BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan magang di PT Jaya Santoso Teknologi, posisi magang ditempatkan di *FrontEnd Developer*. Tanggung jawab utama dari magang ini yaitu difokuskan pada pembangunan tampilan antarmuka admin dashboard untuk website Minyma. Pembangunan antarmuka dilaksanakan dibawah arahan dan bimbingan Bapak Grahana Daffa Herlambang.

Pelaksanaan tugas melibatkan kolaborasi erat dengan berbagai tim terkait, seperti tim pengembang backend dan tim desain *UI/UX*. Koordinasi antar tim dilaksanakan melalui platform komunikasi daring seperti *WhatsApp*, *Discord*. Selain itu, untuk mengatasi tantangan teknis yang muncul selama proses pembangunan, diadakan sesi konsultasi secara berkala bersama supervisor melalui *Google Meet*.

Dalam pengelolaan proyek, digunakan sistem manajemen kode sumber berbasis *Git* dengan repositori yang dihosting di *GitHub*. Untuk setiap fitur atau perubahan, setiap anggota frontEnd harus membuat *branch* baru agar proses pembangunan berjalan secara terpisah dan tidak mengganggu *branch* utama (*master*). Setiap perubahan yang telah selesai akan diajukan melalui *pull request* dan direview oleh rekan tim sebelum dilakukan *merge*. Praktik ini membantu menjaga struktur kode tetap bersih, terorganisir, serta mempermudah kolaborasi antar tim.

3.2 Tugas yang Dilakukan

Tugas yang dikerjakan selama masa magang sebagai *Frontend Developer*, yaitu mengembangkan Frontend Dashboard Admin untuk website Minyma Invitation. Tugas dimulai dengan memahami kebutuhan dan struktur dari desain Figma yang disediakan oleh tim *UI/UX*. Lalu, dilakukan proses implementasi desain dari *Figma* ke dalam bentuk kode menggunakan *framework* NextJS, serta menggunakan Tailwind CSS. Proses implementasi ini melibatkan pembuatan halaman login, dashboard utama, pengelolaan tamu & undangan, manajemen template, dan payment. Untuk memastikan sistem *frontend* dapat berinteraksi dengan layanan *backend*, dilakukan integrasi data menggunakan **endpoint API**.

Komunikasi antar frontend dengan backend diatur melalui mekanisme pengambilan data (*data fetching*) menggunakan **Fetch API**. Selain pembangunan antarmuka, dilakukan juga uji coba terhadap fungsionalitas yang dikembangkan untuk memastikan setiap komponen berjalan dengan benar. Setelah pengujian selesai, proyek akan *di-merge* (digabungkan) ke branch *master*.

3.3 Uraian Pelaksanaan Magang

Tugas yang dikerjakan di PT Jaya Santoso Teknologi pada posisi Frontend Developer selama masa magang telah diuraikan di Tabel 3.1.

Tabel 3.1. Pekerjaan Mingguan Selama Magang

Minggu Ke -	Pekerjaan yang Dilakukan
1	Memahami konsep dasar dari NextJS serta mempelajari alur kerja proyek, Merancang prototipe website profile untuk Minyma, membuat layout dan navbar untuk admin dashboard
2	Membuat komponen Search untuk UserTable, membuat Add User Page untuk halaman User, membuat data Dummy untuk melakukan uji coba pada komponen UserTable, membuat Edit User Page untuk halaman User, membuat library untuk seluruh fungsionalitas di halaman User
3 - 4	Membuat fungsi untuk menampilkan data user yang sedang di edit di halaman User, membuat halaman utama (index) dashboard admin, melakukan revisi pada navbar dashboard admin, membuat View page untuk halaman User, melakukan revisi pada halaman User dan navbar pada admin dashboard, membuat Template Page untuk dashboard admin, membuat Card untuk menampilkan isi dari Template
	Lanjutan di halaman berikutnya...

Minggu Ke -	Pekerjaan yang Dilakukan
5 - 6	Menambahkan fitur paginasi di View User Page dalam halaman User, Membuat komponen DeleteConfirmation, Membuat library untuk seluruh funsionalitas di halaman Template, membuat fungsi di dalam DeleteConfirmation untuk halaman Template, membuat Page Update dan Add untuk halaman Template, membuat komponen PreviewCard untuk page Update di halaman Template, serta menyelesaikan Page Add untuk halaman Template
7	Menyelesaikan page Update untuk halaman Template dan menambahkan komponen PreviewCard ke dalam Page Update, membuat library untuk Update Page di halaman Template
8	Merapihkan Card di page Update, Memperbaiki bug untuk mengecek file dan lightmode di navbar, membuat dropdown menu di navbar admin dashboard, menambahkan search email ke dalam SearchBar untuk UserTable
9	Integrasi backend login ke dashboard admin, belajar tentang CORS untuk memperbaiki bug di dashboard admin, memperbaiki layout admin dashboard
10	Menambahkan error handling di login, menambahkan modal untuk login, inisialisasi template Sangjit dengan membuat layout dan folder untuk template Sangjit
11	Membuat komponen Location, Gallery, Countdown, dan Navbar untuk template Sangjit. Lalu melanjutkan integrasi backend login ke frontend, belajar untuk menyimpan cookie, membuat API frontend untuk menyimpan token ke cookie.
12 - 13	Mengimplementasikan API Frontend untuk menyimpan cookie ke login, melakukan uji coba login dan memperbaiki bug login, melakukan implementasi logout di admin dashboard, belajar menghapus cookie, implementasi penghapusan cookie ke sistem logout
14	Melakukan uji coba sistem logout, mengintegrasikan endpoint API untuk halaman User di dashboard admin, menambahkan parameter Device ke sistem login

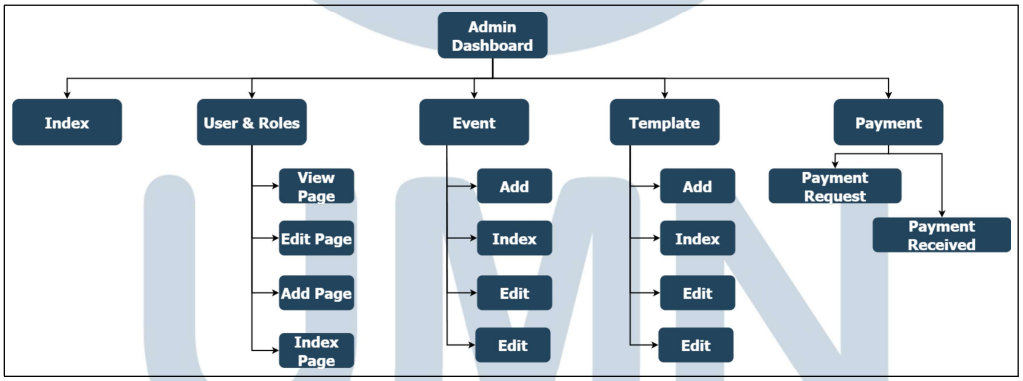
Lanjutan di halaman berikutnya...

Minggu Ke -	Pekerjaan yang Dilakukan
15 - 16	Mengembangkan middleware untuk admin dashboard, memperbaiki bug dalam middleware, menambahkan tombol untuk logout dalam navbar di admin dashboard, memperbaiki layout navbar, menambahkan paginasi untuk UserTable dalam halaman User, memperbaiki layout UserTable, optimasi UserTable, Melakukan inisialisasi untuk login menggunakan akun Google

3.4 Hasil Implementasi Pembangunan Antarmuka

Dibawah ini merupakan hasil dari implementasi pembangunan Dashboard Admin untuk website Minyma menggunakan NextJS.

3.4.1 Sitemap Diagram

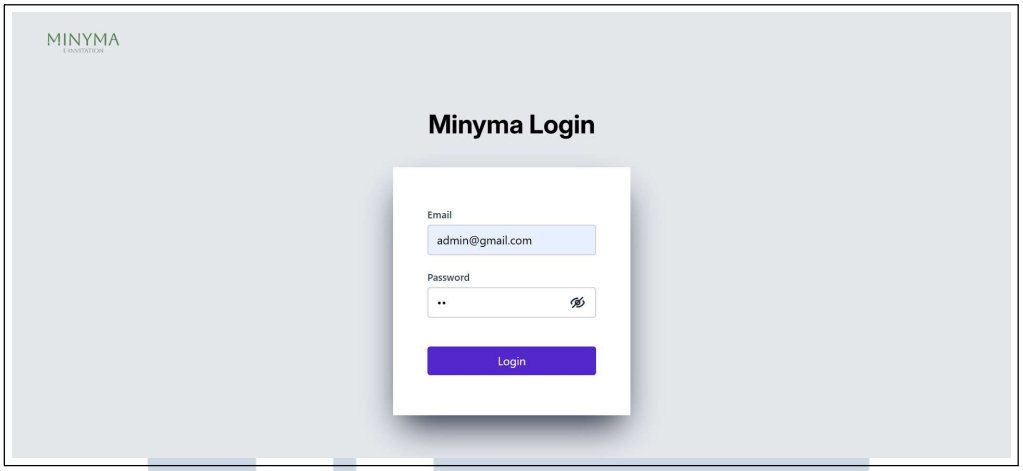


Gambar 3.1. Sitemap Minyma

3.4.2 Hasil Implementasi

A. Login Page

Gambar 3.1 merupakan Page Login untuk Website Minyma. Di dalam Page Login, terdapat dua *input field*. Yang pertama yaitu bertipe email, dan yang kedua adalah password. Setelah pengguna memasukkan email dan password ke dalam masing - masing *input field* lalu menekan tombol **Login**, maka sistem akan otomatis melakukan autentikasi. Jika data yang dimasukkan valid, maka pengunjung akan langsung dialihkan ke halaman admin dashboard.



Gambar 3.2. Login Website Minyma

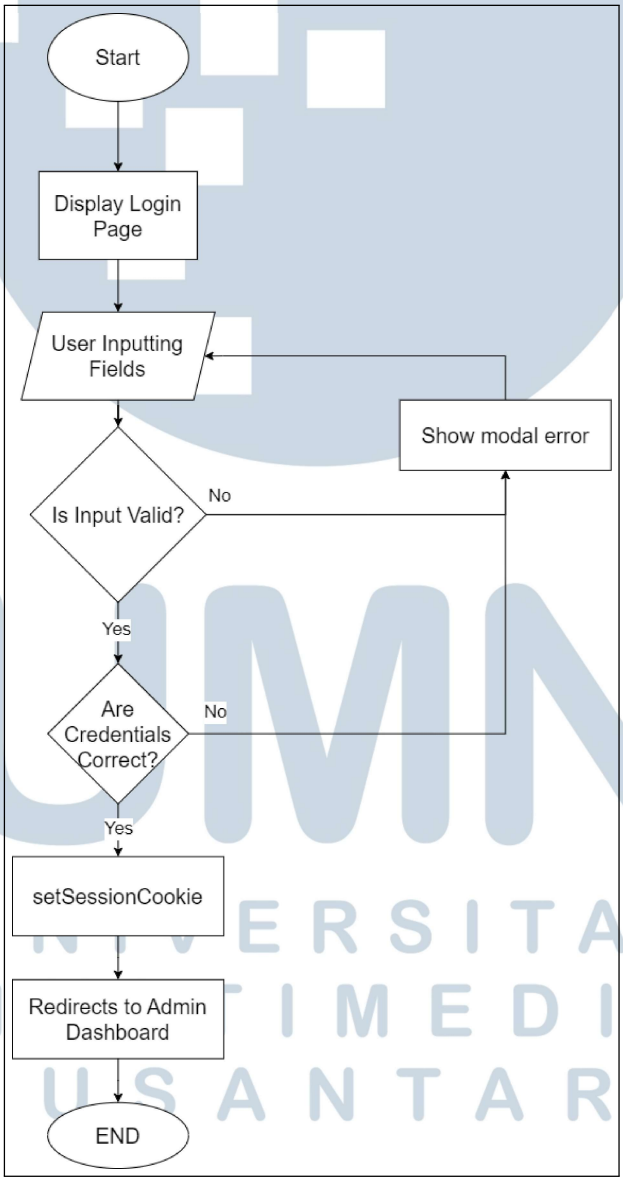
```
export async function login(email, password) {
  try {
    const UUID = getUUID();
    const res = await fetch(url + "/api/v1/auth/login", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        email: email,
        password: password,
        deviceId: UUID,
      }),
    });
    const data = await res.json();

    if (!res.ok) {
      throw {
        message: data.message || res.message,
        status: data.status || res.status || 500,
      };
    }
    setSessionCookie(data.data.accessToken, data.data.refreshToken
    , UUID)
      .then((val) => {
      })
      .catch((err) => {
        throw {
          message: error.message || "Cookie was not saved",
        };
      });
    return data;
  } catch (error) {
    throw {
      message: error.message || "Something went wrong.",
      status: error.status || 500,
    };
  }
}
```

Kode 3.1: Login

Kode 3.1 ini digunakan untuk melakukan autentikasi ketika pengguna menekan tombol *login* di dalam halaman login. Setelah itu, *function* `login`

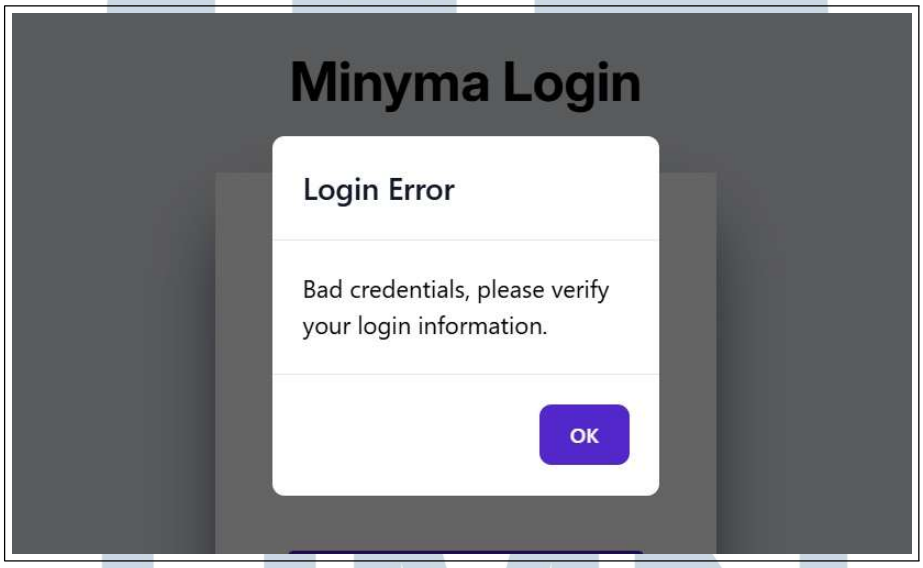
akan mengirim *request* ke API backend beserta dengan data yang berada dalam parameter *function* `login`. Lalu, sistem akan memberikan *respond* dalam bentuk *JSON* yang nantinya akan disimpan ke dalam *res*. Setelah disimpan kedalam *res*, *JSON* dilakukan *parsing* untuk mengambil data yang akan disimpan ke dalam variable. Dan pada akhirnya `setSessionCookie` akan menyimpan token yang dikirim dari *backend* lalu mengalihkan pengguna ke halaman admin dashboard. Dapat dilihat alur dari halaman login pada Gambar 3.3



Gambar 3.3. Alur Flowchart Login

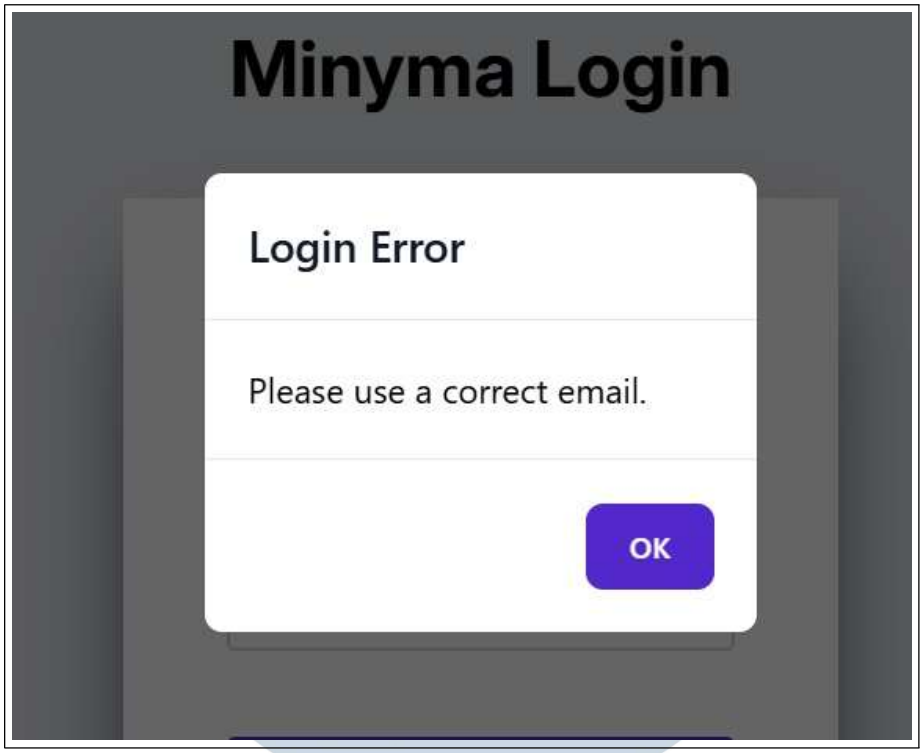
B. Login Modal Pop-up Component

Pada Gambar 3.4 ditampilkan sebuah *modal pop-up* yang berfungsi sebagai notifikasi kesalahan atau *error message*. *Modal* tersebut akan muncul secara otomatis ketika terjadi kesalahan dalam proses autentikasi, yaitu ketika pengguna memasukkan email dengan password yang tidak sesuai dengan data yang disimpan oleh sistem. Kehadiran *modal* ini bertujuan untuk memberikan umpan balik kepada pengguna secara langsung agar pengguna dapat melakukan koreksi terhadap informasi yang dimasukkan.



Gambar 3.4. Login error modal (invalid data)

Seperti yang ditunjukkan pada Gambar 3.4, sistem menampilkan *modal pop-up* sebagai peringatan kesalahan autentikasi. Sementara itu, pada Gambar 3.5, ditampilkan notifikasi kesalahan yang muncul apabila pengguna mengisi *input field* untuk email dengan format yang tidak sesuai dengan standar format alamat email yang valid. Kesalahan ini terjadi ketika format yang dimasukkan tidak mengandung simbol **"@"** atau domain yang benar, sehingga sistem secara otomatis menolak input tersebut dan memberikan umpan balik berupa pesan *error* untuk membantu pengguna memperbaiki data yang dimasukkan.



Gambar 3.5. Login error modal (email)

```
if (email == "" || password == "") {
    setModalTitle("Login Error");
    setModalMessage("Please fill the fields.");
    setModal(true);
    setDebounce(false);
} else if (!email.includes("@")) {
    setModalTitle("Login Error");
    setModalMessage("Please use a correct email.");
    setModal(true);
    setDebounce(false);
```

Kode 3.2: Login Error (Data Invalid)

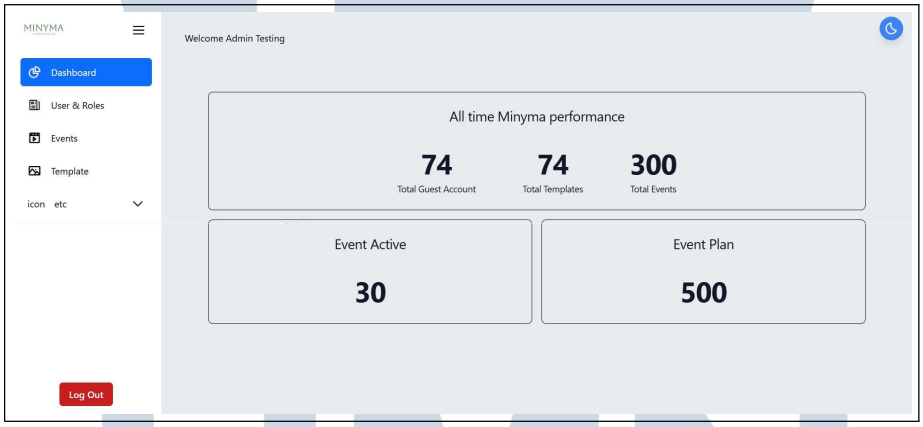

```
setModalTitle("Login Error");
if (err.status == 401) {
    setModalMessage(err.message + ", please verify your login
    information.");
} else {
    setModalMessage("Something unexpected happened. Please try
    again.");
}
setModal(true);
```

```
8 setDebounce(false);
```

Kode 3.3: Login Error (Wrong Password)

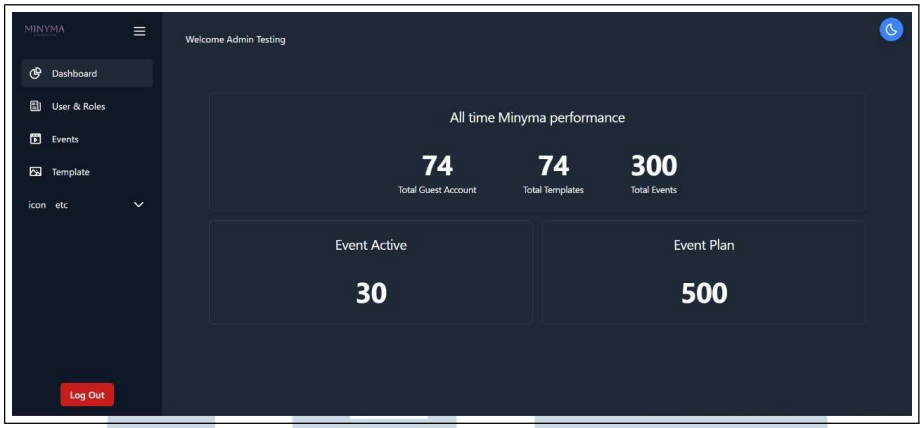
Pada kode 3.2 ini digunakan untuk melakukan pengecekan antar kedua *input field* email dan password. Dan kode 3.3 ini digunakan untuk mengirim informasi error kepada pengguna jika ada kesalahan dalam memasukkan password. Untuk mengatur *title* dari *modal* cukup dengan menggunakan `setModalTitle` dan `setModalMessage` untuk mengubah pesan dari *modal* yang ingin dimunculkan ke pengguna, lalu `setModal` untuk menampilkan *modal* ke pengguna.

C. Admin Dashboard Index



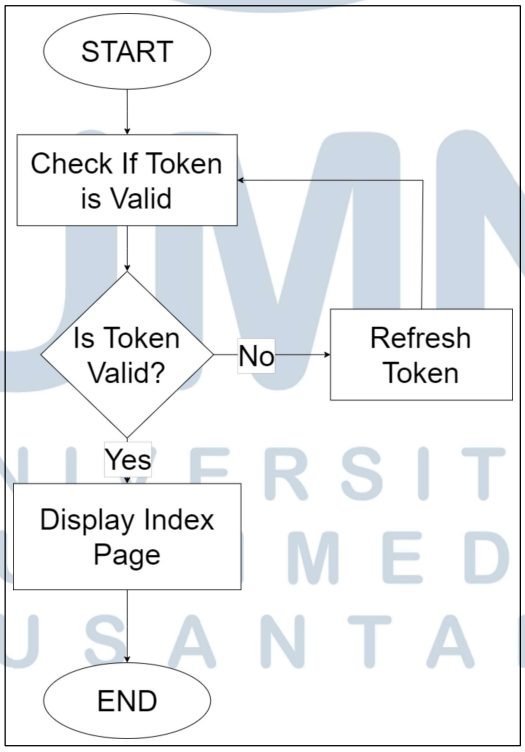
Gambar 3.6. Website utama admin dashboard Minyma

Pada Gambar 3.6 merupakan tampilan halaman utama dari admin dashboard yang berfungsi sebagai pusat informasi bagi admin. Di dalam halaman ini, ditampilkan ringkasan data penting berupa total akun pengguna, jumlah template yang tersedia, serta jumlah event yang sedang aktif. Informasi ini disusun secara terstruktur untuk memudahkan pemantauan dan pengelolaan sistem secara efisien.



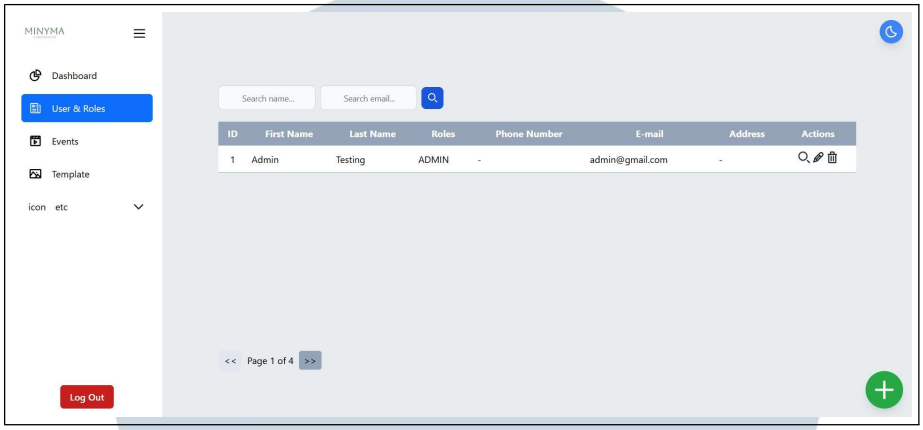
Gambar 3.7. Website utama admin dashboard (*dark mode*)

Selain itu, pada Gambar 3.7 merupakan versi alternatif dari halaman utama yang sama, namun dengan tampilan *dark mode* atau mode gelap. Fitur mode gelap ini disediakan sebagai opsi bagi pengguna yang lebih menyukai tampilan dengan pencahayaan rendah, sekaligus meningkatkan kenyamanan visual saat penggunaan dalam kondisi minim cahaya. Pada gambar 3.8 menunjukkan alur dari tampilan halaman index (beserta halaman lainnya).

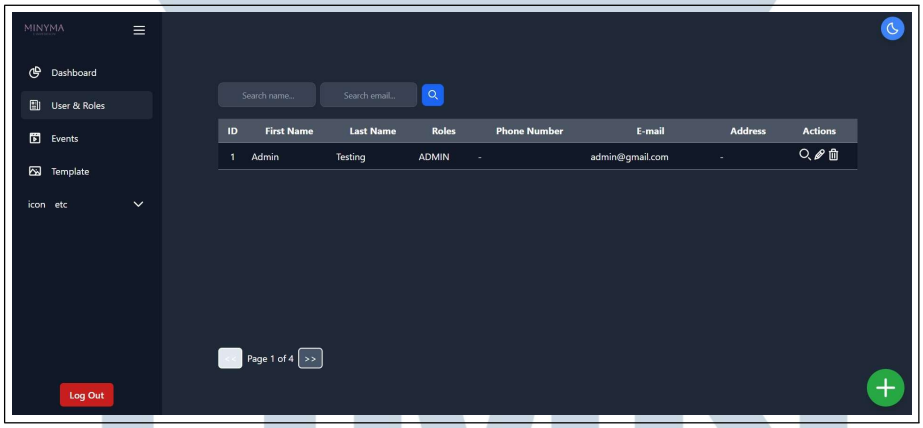


Gambar 3.8. Flowchart Index (dan akses halaman lainnya)

D. User and Roles Page



Gambar 3.9. User & Roles Page



Gambar 3.10. User & Roles Page (Dark Mode)

Gambar 3.9 menampilkan halaman utama dari menu *User & Roles* yang terdapat di dalam admin dashboard. Pada halaman ini, disediakan sebuah tabel interaktif yang menampilkan data - data pengguna yang tersimpan dalam sistem, seperti nama, peran, dan informasi terkait lainnya. Melalui halaman ini, pengguna yang memiliki hak akses sebagai admin dapat melakukan berbagai aksi pengelolaan data, termasuk menambahkan pengguna baru dengan cara menekan tombol tambah (+) terletak dipojok bawah kanan, mengubah informasi pengguna yang sudah ada, maupun menghapus data pengguna. Untuk meningkatkan fleksibilitas tampilan, halaman ini juga dilengkapi dengan fitur *dark mode* atau mode gelap seperti yang ditunjukkan pada Gambar 3.10.

```
getUsers(
    accessToken,
    (pageValue || initialPage) - 1,
    sizeValue || initialSize
    ).then((val) => {
    if (pageValue > val.data.totalPages) {
        const newQuery = {
        ...query,
        page: val.data.totalPages || initialPage,
        size: sizeValue || initialSize,
        };
        router.replace({
        pathname: router.pathname,
        query: newQuery,
        });
    } else if (pageValue <= 0) {
        const newQuery = {
        ...query,
        page: initialPage,
        size: sizeValue || initialSize,
        };
        router.replace({
        pathname: router.pathname,
        query: newQuery,
        });
    } else {
        isLastPage(val.data.last);
        totalPages(val.data.totalPages);
        setListOfUsers(val.data.content);
        setIsLoading(false);
        paginationLoading(false);
        setRefreshTable(false);
    }
    console.log(val.data);
})
```

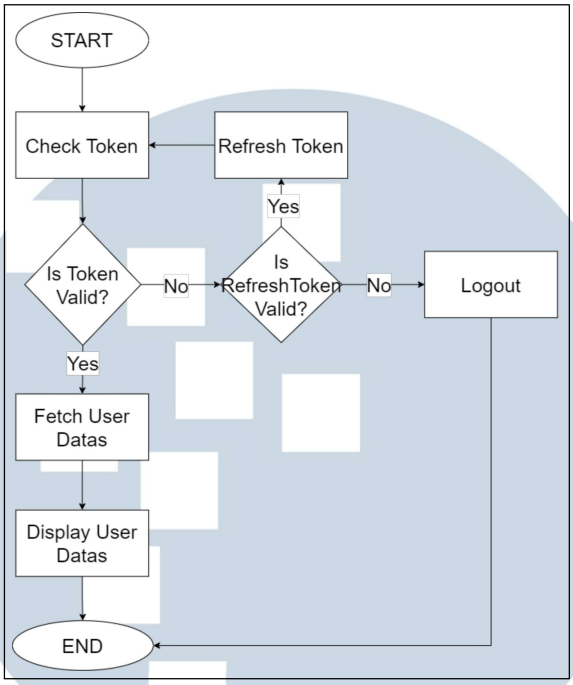
Kode 3.4: User & Roles Code


```
function handlePaging(inc) {
    const newQuery = {
      ...query,
      page: Number(query.page) + inc || initialPage + inc,
      size: query.size || initialSize,
    };
```

```
    router.push(
      {
        pathname: router.pathname,
        query: newQuery,
      },
      undefined,
      { shallow: true }
    );
}
```

Kode 3.5: User & Roles Pagination Handler

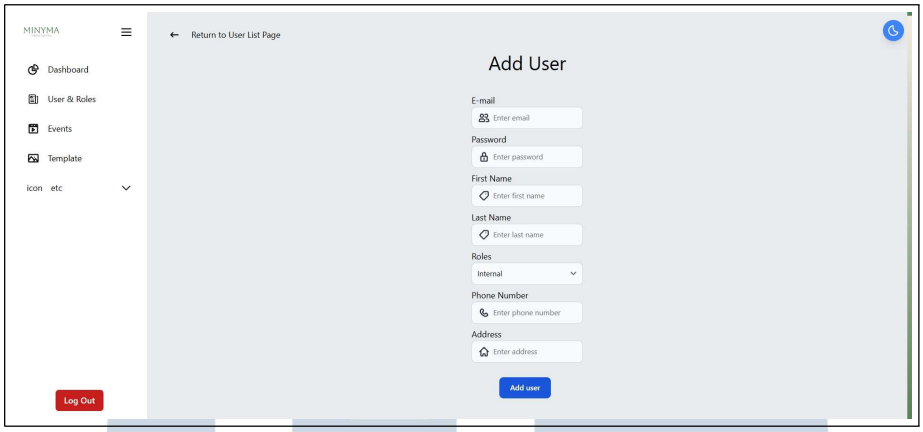
Kode 3.4 ini digunakan untuk menjalankan fungsi `getUsers()` yang akan melakukan request ke *API backend* dengan mengirim `accessToken`, `pageValue || initialPage` dan `sizeValue || initialSize` untuk mengambil data dari *backend*, lalu data tersebut disimpan ke dalam fungsi *setListOfUsers*, lalu menampilkannya ke dalam tabel *user*. Kode 3.4 juga digunakan untuk mengatur paginasi dari tabel *user* dengan melakukan inisialisasi *query* `page` dan `size` ke *url* secara otomatis. Kode 3.5 digunakan untuk mengelola paginasi dari tabel *user*. Ketika pengguna menekan tombol *next*, maka fungsi `handlePaging` berjalan untuk mengubah *query* dari *url* dengan menambahkan `page`, sehingga tabel yang ditampilkan berubah sesuai dengan urutan. Pada gambar 3.11 menunjukkan alur dari halaman User & Roles.



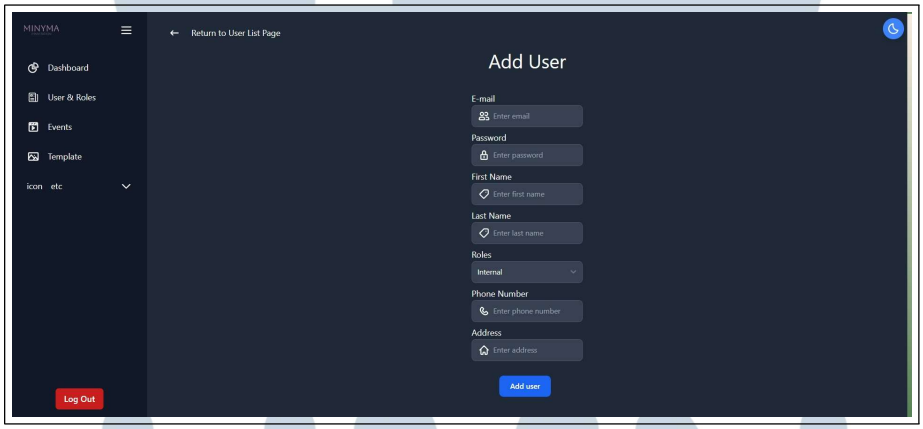
Gambar 3.11. Flowchart User & Roles

E. Add User Page

Gambar 3.12 menampilkan tampilan halaman Add User yang muncul setelah tombol tambah pengguna ditekan pada halaman User & Roles. Halaman ini digunakan untuk menambahkan data pengguna baru ke dalam sistem melalui formulir isian yang telah disediakan, seperti nama, email, kata sandi, peran, nomor telepon, dan juga alamat. Setelah seluruh data yang dibutuhkan diisi, informasi tersebut kemudian disimpan ke dalam sistem. Halaman ini juga dilengkapi dengan fitur *dark mode* atau mode gelap seperti yang ditampilkan pada gambar 3.13.



Gambar 3.12. Add User



Gambar 3.13. Add User (Dark Mode)

```
const addPage = async (prevState, formData) => {
let func = await addUser(
    formData.get("email"),
    formData.get("password"),
    formData.get("first_name"),
    formData.get("last_name"),
    formData.get("roles"),
    formData.get("phone_number"),
    formData.get("address")
)
```

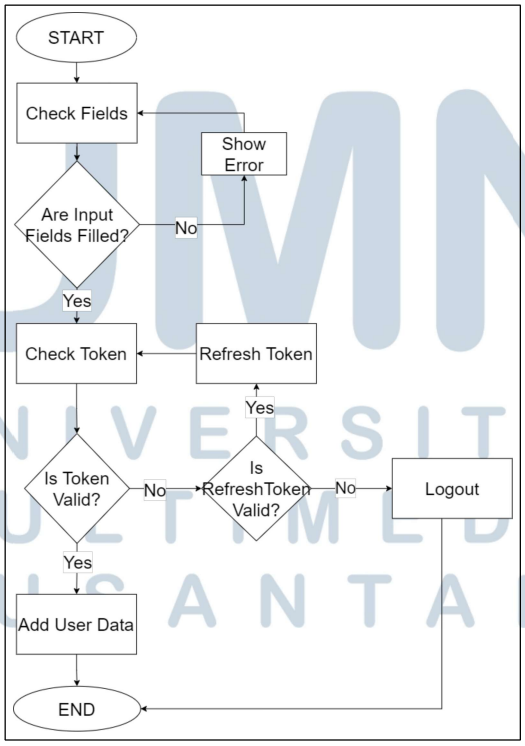
Kode 3.6: Add User Code

```
export async function addUser(
  email,
  password,
```

```
  first_name,
  last_name,
  roles,
  phone_number,
  address
) {
  let concat =
    email + password + first_name + last_name + roles +
   phone_number + address;
  return concat;
}
```

Kode 3.7: Add User Library

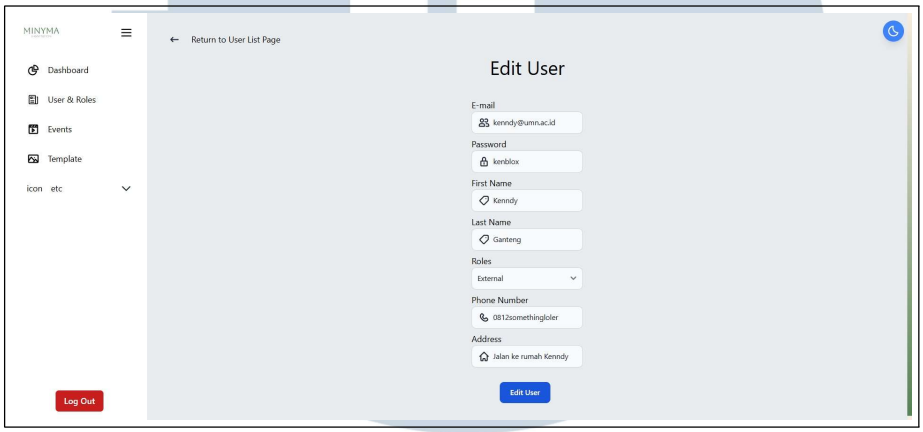
Kode 3.6 ini berfungsi untuk mengirim seluruh data melalui *input field* ketika pengguna menekan tombol *Add User*. Setelah menekan tombol *Add User*, fungsi `addPage()` akan dijalankan, yang di dalamnya memanggil fungsi `addUser()`, fungsi `addPage()` kemudian mengirim semua data dari *formData* ke dalam parameter fungsi `addUser()`. Kode 3.7 merupakan fungsi `addUser()` yang akan bertugas untuk menangkap data yang dikirim dari kode 3.6. Terdapat juga alur dari halaman Add User pada Gambar 3.14



Gambar 3.14. Flowchart Add User

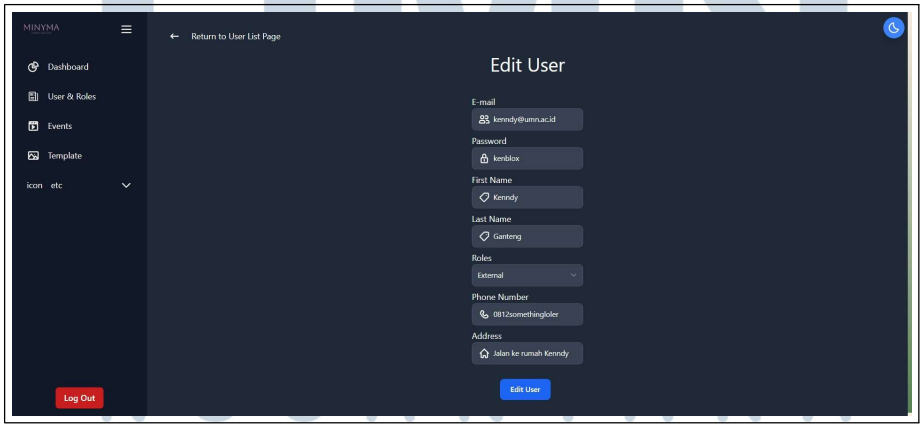
F. Edit User Page

Gambar 3.15 menampilkan halaman Edit User yang muncul ketika tombol edit ditekan pada tabel di halaman User & Roles. Halaman ini digunakan untuk mengubah atau memperbarui data pengguna yang telah tersimpan di dalam sistem. Setelah isi *field* diubah, pengguna hanya perlu menekan tombol Edit User untuk mengubah data pengguna sesuai dengan informasi yang dimasukkan oleh pengguna.



Gambar 3.15. Edit User Page

Halaman Edit User juga dilengkapi dengan fitur *dark mode* atau mode gelap, sehingga tampilan tetap optimal dan mudah dibaca dalam berbagai kondisi pencahayaan. *dark mode* atau mode gelap dapat dilihat pada Gambar 3.16.



Gambar 3.16. Edit User Page (Dark Mode)

```
1 const editPage = (prevState, formData) => {
2     editUser(
```

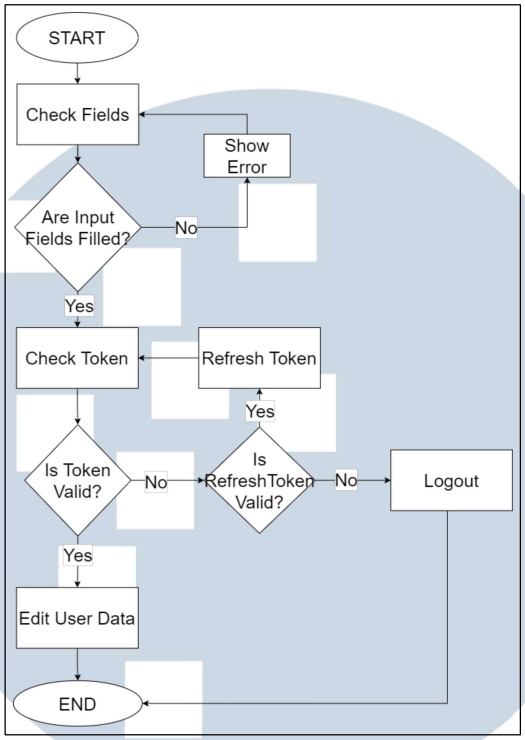
```
    id,
    formData.get("email"),
    formData.get("password"),
    formData.get("first_name"),
    formData.get("last_name"),
    formData.get("roles"),
    formData.get("phone_number"),
    formData.get("address")
    )
```

Kode 3.8: Edit User Code

```
export async function editUser(
  id,
  email,
  password,
  first_name,
  last_name,
  roles,
  phone_number,
  address
) {
  return "Edited " + id;
}
```

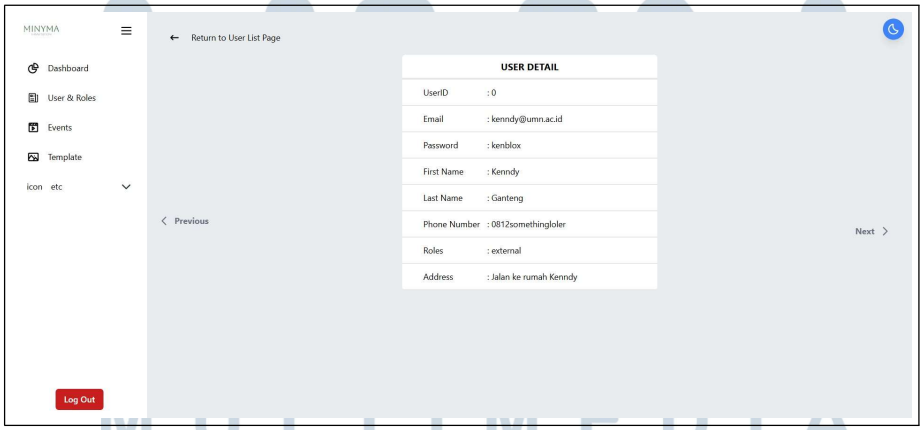
Kode 3.9: Edit User Library

Kode 3.8 bertujuan untuk mengirim data dari user yang sudah di*edit* oleh pengguna ketika tombol *Edit User* ditekan. Setelah tombol tersebut ditekan, fungsi `editPage()` akan dijalankan, yang didalamnya terdapat fungsi `editUser()`. Kode 3.9 merupakan fungsi `editUser()` untuk menangkap data yang dikirim dari fungsi `editPage()` pada kode 3.8. Berikut merupakan alur dari halaman Edit User pada Gambar 3.17



Gambar 3.17. Flowchart Edit User

G. View User Page



Gambar 3.18. View User Page

Gambar 3.19. View User Page (Dark Mode)

Gambar 3.18 menampilkan halaman View User yang muncul ketika tombol view ditekan pada halaman User & Roles. Halaman ini berfungsi untuk menampilkan detail informasi dari masing - masing pengguna yang telah terdaftar dalam sistem. Selain menampilkan data secara rinci, halaman ini juga dilengkapi dengan fitur paginasi, yang digunakan untuk berpindah antar data pengguna, baik pengguna sebelumnya maupun berikutnya. Selain itu, halaman View User juga dilengkapi dengan fitur *dark mode* atau mode gelap seperti yang di tampilkan pada Gambar 3.19.

```
getUser(id).then((val) =>
    setUserData([
    {
        email: val.email,
        password: val.password,
        first_name: val.first_name,
        last_name: val.last_name,
        roles: val.roles,
        phone_number: val.phone_number,
        address: val.address,
    },])
);
```

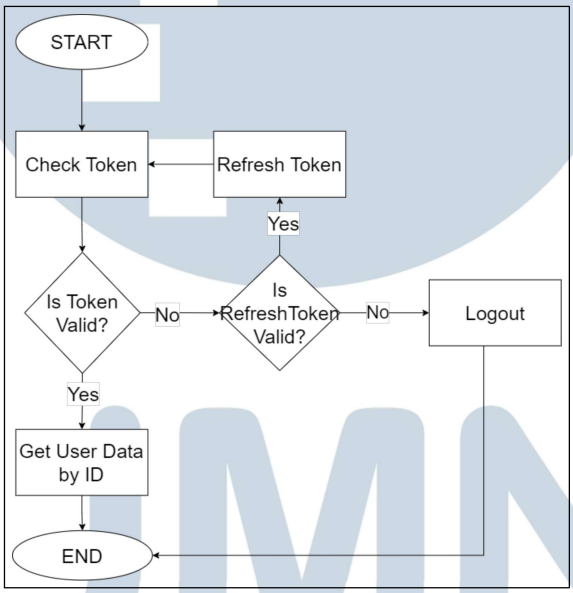
Kode 3.10: View User Code

```
export async function getUser(id) {
  let data = await fetch("http://localhost:3000/api/hello");
  let res = await data.json();
  let userData = [];
  res.users.forEach((user) => {
    if (user.id == id) {
```

```
      userData = user;
    }
  });
  return userData;
}
```

Kode 3.11: View User Library

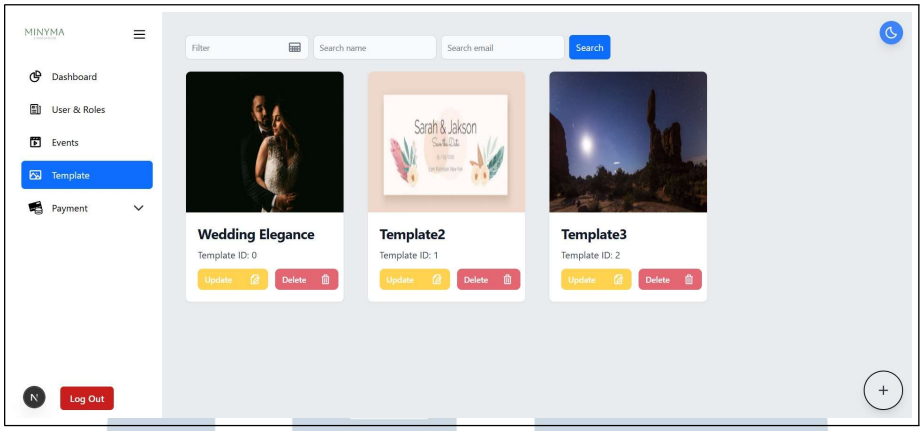
Kode 3.10 ini berfungsi untuk mengambil data pengguna yang kemudian disimpan menggunakan fungsi `setUserData()`, lalu ditampilkan ke halaman view user. Kode 3.11 merupakan fungsi `getUser()` untuk mengambil data dari *endpoint API*, lalu mengirim data tersebut ke Kode 3.10 untuk ditampilkan di halaman view user. 3.20 merupakan Gambar dari alur halaman View User.



Gambar 3.20. Flowchart View User

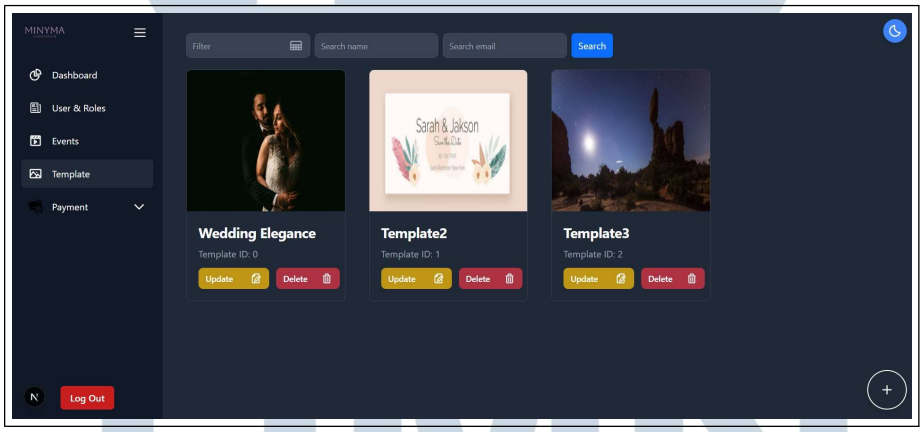
H. Template Page

Halaman template, seperti yang ditampilkan pada Gambar 3.21, digunakan untuk menampilkan daftar template dalam bentuk kartu yang telah disimpan dalam sistem. Pada halaman ini, pengguna dengan hak akses yang sesuai dapat melakukan pengelolaan data template, termasuk mengubah, menghapus, serta menambahkan template baru sesuai kebutuhan. Seluruh fitur tersebut disajikan dalam antarmuka yang terstruktur agar memudahkan proses pengelolaan.



Gambar 3.21. Template Page

Selain itu, halaman template juga dilengkapi dengan fitur *dark mode* atau mode gelap pada Gambar 3.22.



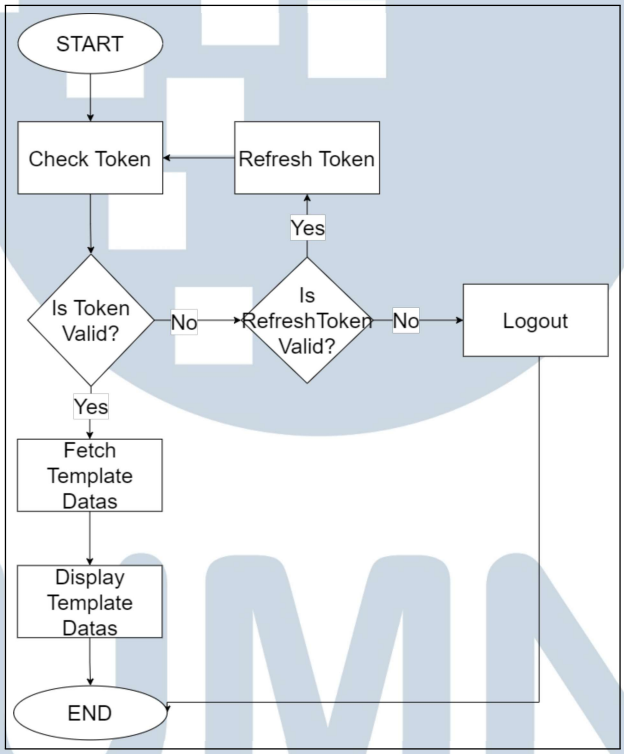
Gambar 3.22. Template Page (Dark Mode)

```
getTemplates()
    .then((val) => {
        console.log(val);
        setListOfTemplates(val);
        setIsLoading(false);
        setRefreshList(false);
        })
    .catch((err) => {
        errorCode = err;
        setIsError(true);
        setIsLoading(false);
        setRefreshList(false);
```

```
13 });
```

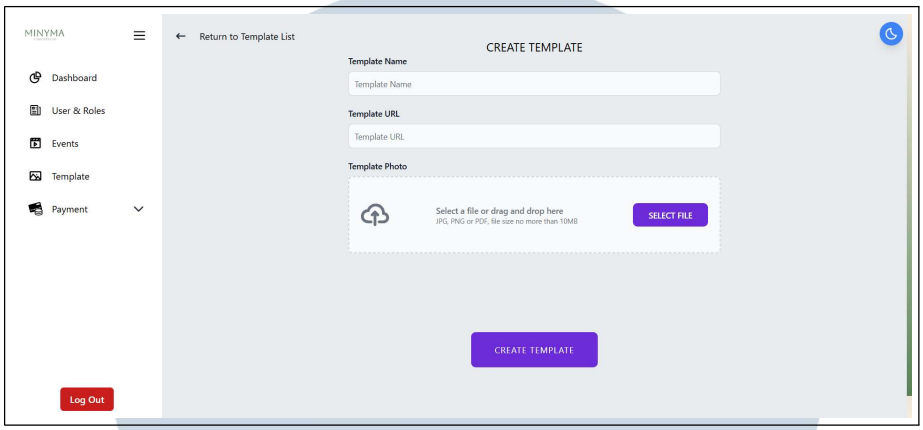
Kode 3.12: Template List Code

Guna dari kode 3.12 yaitu untuk mengambil data template yang tersimpan di dalam sistem *backend*. Data yang diambil lalu disimpan ke dalam function `setListOfTemplates()`. Setelah data disimpan, data yang sudah tersimpan di dalam function akan ditampilkan dalam bentuk komponen *Card*. Gambar 3.23 berikut merupakan alur kode halaman Index Template.



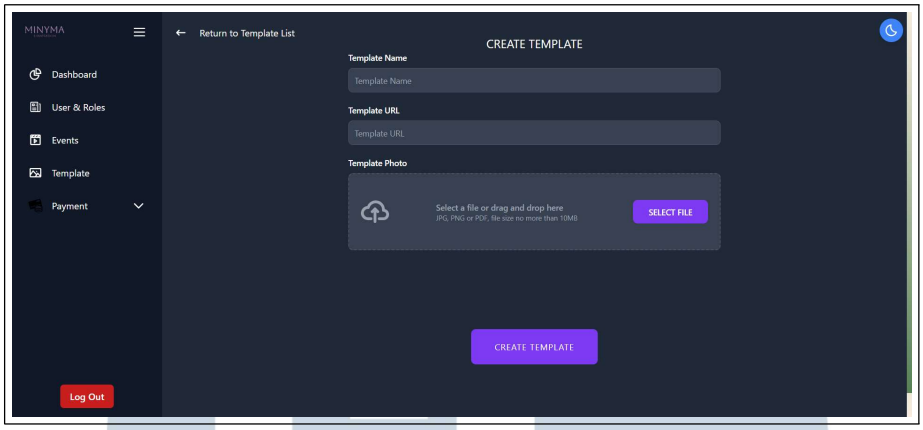
Gambar 3.23. Flowchart Index Template

I. Create Template



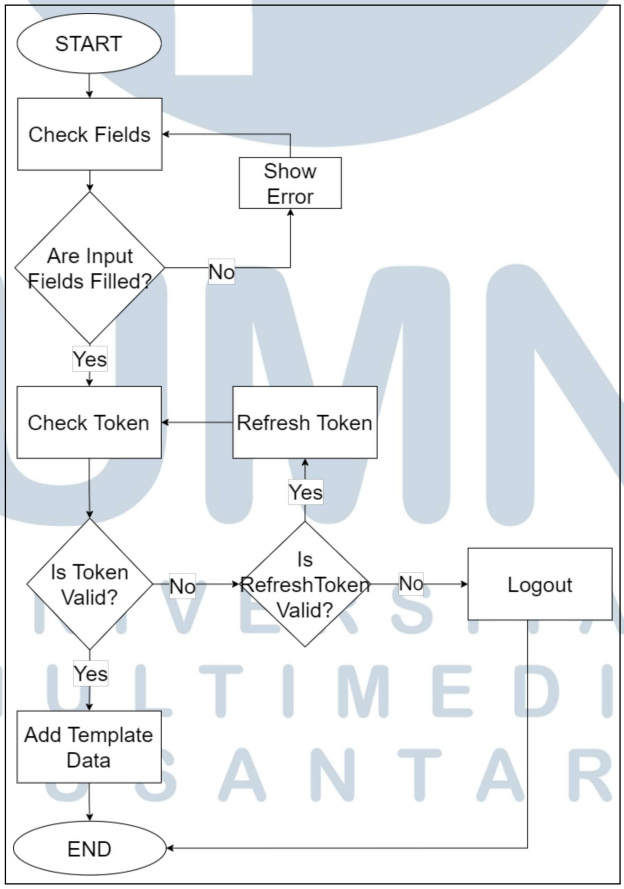
Gambar 3.24. Create Template

Halaman Create Template, seperti yang ditampilkan pada Gambar 3.24, muncul ketika tombol tambah (+) ditekan pada halaman Template. Pada halaman ini, ditampilkan formulir yang digunakan untuk mengisi data terkait template baru, seperti nama template, *URL* template, dan gambar yang akan mewakili tampilan visual dari template tersebut. Setelah seluruh data yang dibutuhkan dimasukkan, proses penyimpanan dapat dilakukan dengan menekan tombol *Create Template*, sehingga informasi yang telah diisi akan tersimpan ke dalam sistem.



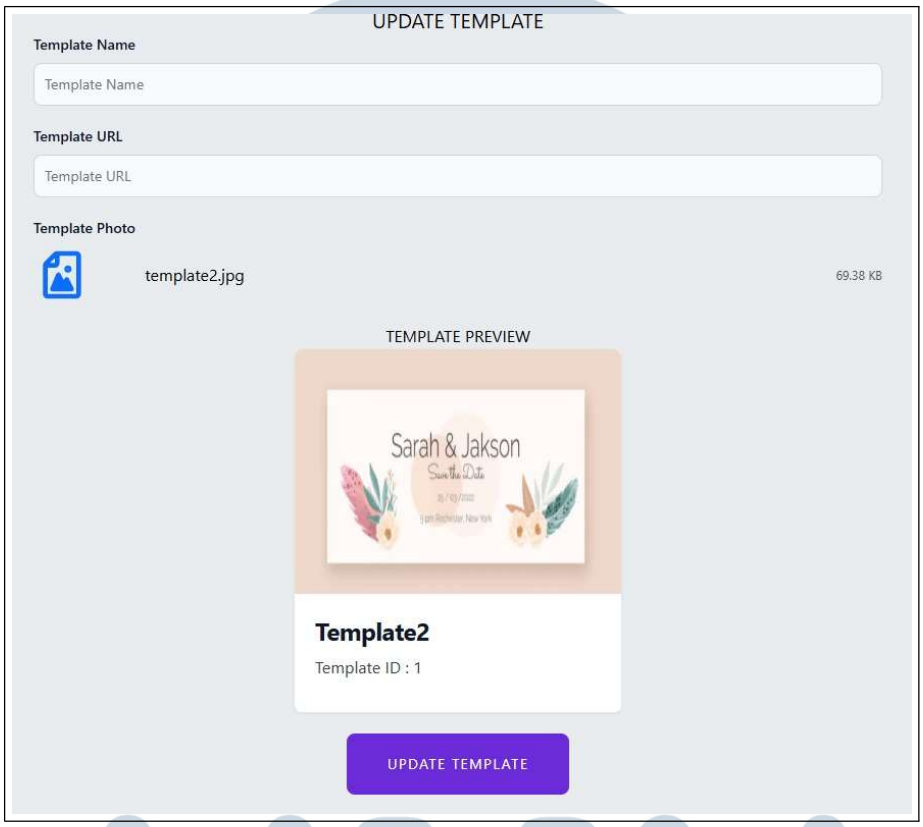
Gambar 3.25. Create Template (Dark Mode)

Untuk menjaga kenyamanan tampilan dan konsistensi antarmuka, halaman Create Template ini juga dilengkapi dengan fitur *dark mode* atau mode gelap pada Gambar 3.25. Gambar 3.26 berikut merupakan alur kode halaman Add Template.



Gambar 3.26. Flowchart Add Template

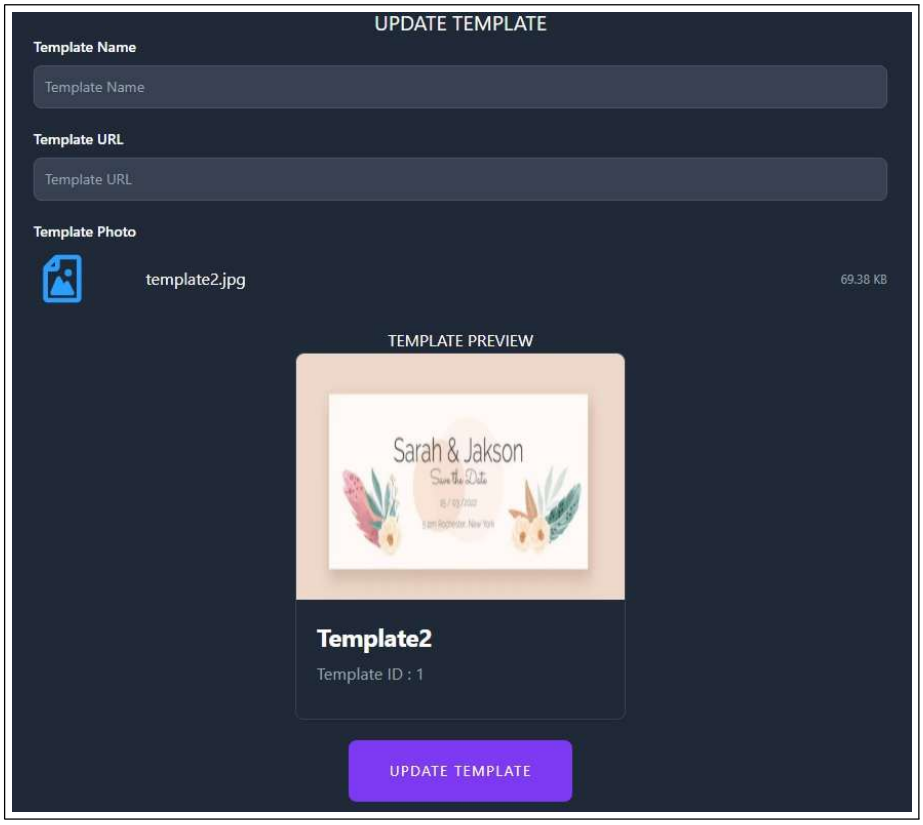
J. Update Template



Gambar 3.27. Update Template

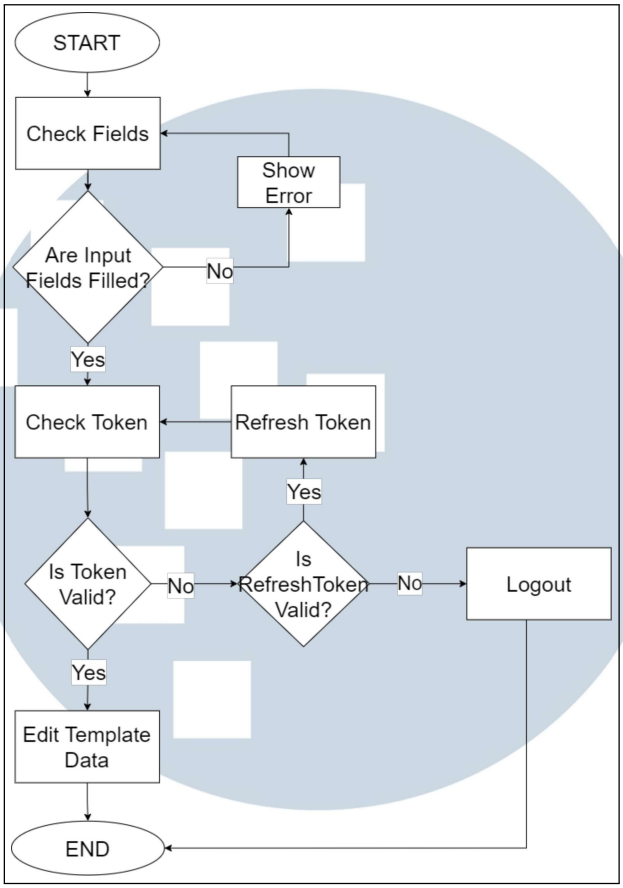
Halaman Update Template, seperti yang ditampilkan pada Gambar 3.27, akan muncul ketika tombol update ditekan pada salah satu kartu template di halaman Template. Pada halaman ini, disediakan formulir yang telah terisi dengan data dari template yang ingin diubah.

Pengguna dapat melakukan pengeditan terhadap informasi yang ada di dalam formulir, seperti nama, URL, atau gambar template. Setelah perubahan dilakukan, tombol Update Template dapat ditekan untuk menyimpan perubahan tersebut, dan sistem akan secara otomatis memperbarui data template di dalam basis data.



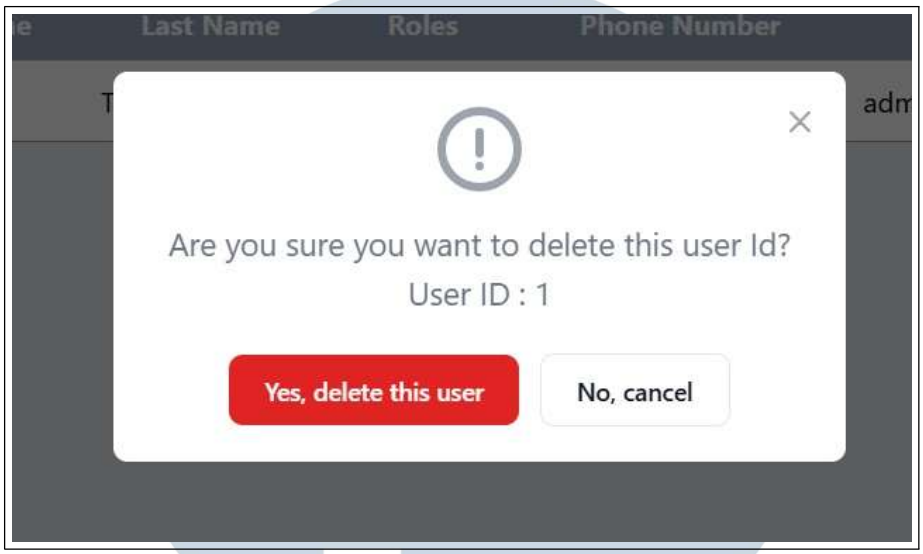
Gambar 3.28. Update Template (Dark Mode)

Tampilan halaman *Update Template* juga dilengkapi dengan fitur *dark mode* atau mode gelap untuk menjaga kenyamanan visual dan konsistensi desain seperti Gambar 3.28. Gambar 3.29 berikut merupakan alur kode halaman Update Template.

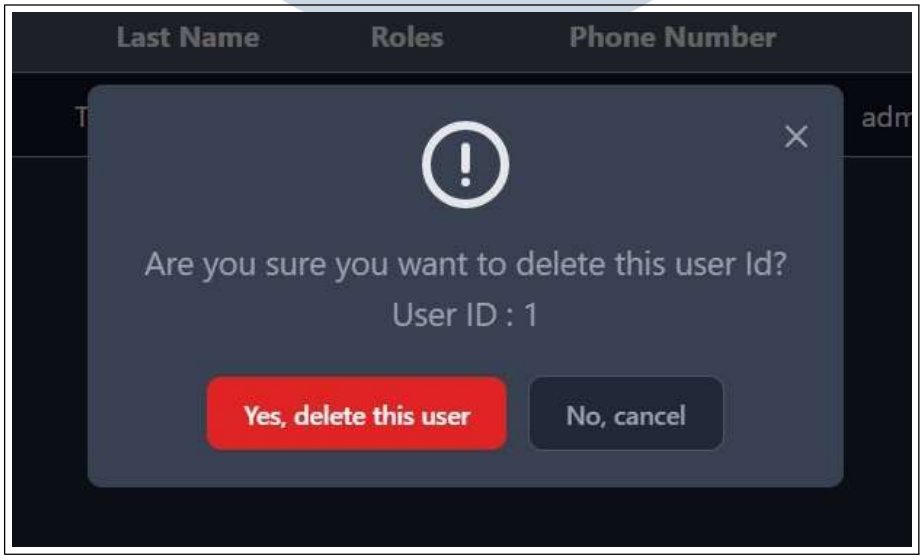


Gambar 3.29. Flowchart Update Template

K. Delete Confirmation Component



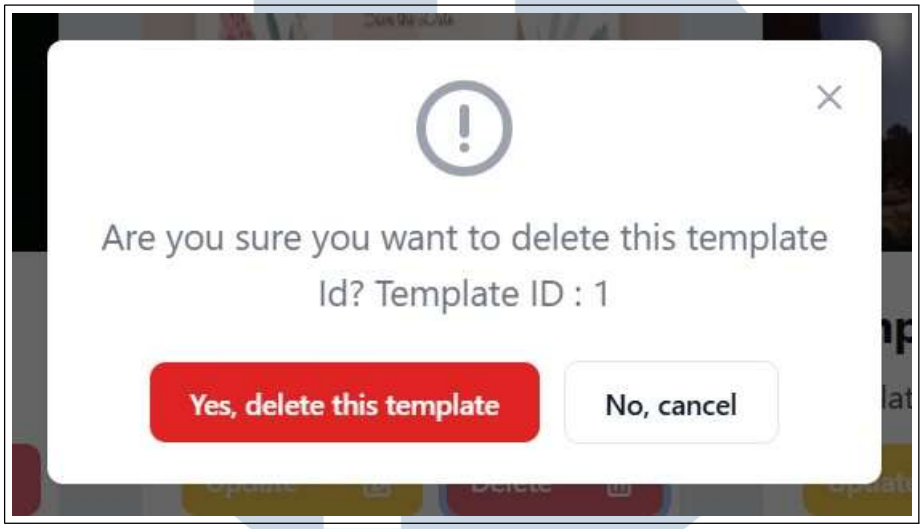
Gambar 3.30. Delete User



Gambar 3.31. Delete User (Dark Mode)

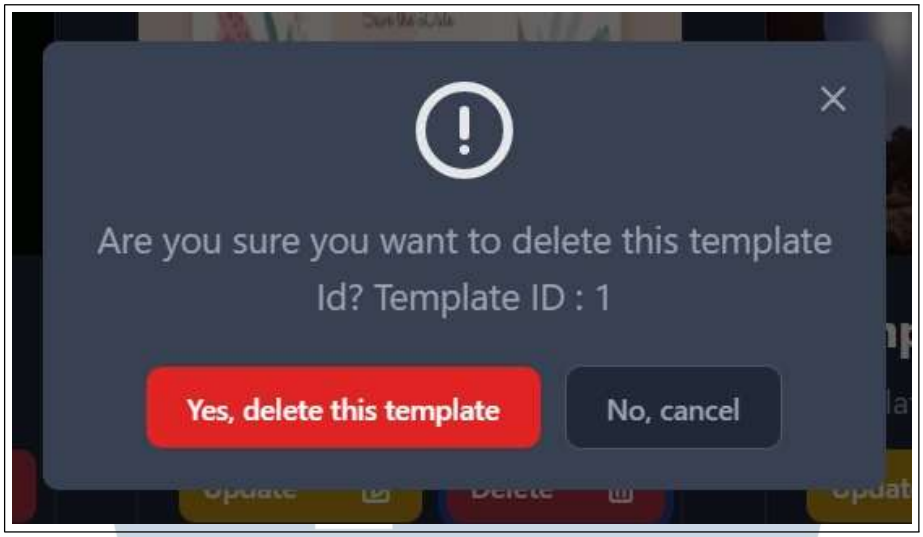
Modal pop-up Delete Confirmation, seperti yang ditampilkan pada Gambar 3.30, akan muncul ketika tombol delete ditekan pada halaman User & Roles. *Modal* ini ditampilkan sebagai bentuk umpan balik sekaligus peringatan kepada pengguna sebelum melakukan penghapusan data pengguna lain dari sistem. Tujuan dari keberadaan *modal* ini adalah untuk mencegah tindakan penghapusan yang tidak

disengaja, dengan cara meminta konfirmasi ulang apakah data yang akan dihapus sudah sesuai dengan keinginan. Dengan demikian, proses penghapusan dapat dilakukan secara lebih hati-hati dan terkontrol. Selain itu, modal ini juga dilengkapi dengan fitur *dark mode* atau mode gelap seperti pada tampilan pada Gambar 3.31.



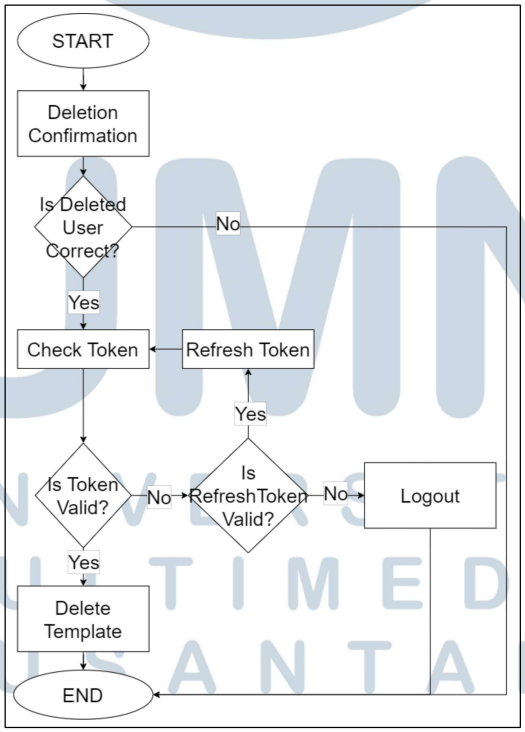
Gambar 3.32. Delete Template

Modal pop-up Delete Confirmation pada Gambar 3.32 muncul ketika pengguna menekan tombol delete pada halaman Template. *Modal* ini berperan sebagai umpan balik sebagai peringatan kepada pengguna sebelum melakukan penghapusan data template dari sistem. Tujuan dari modal ini agar mencegah tindakan penghapusan yang tidak disengaja.



Gambar 3.33. Delete Template (Dark Mode)

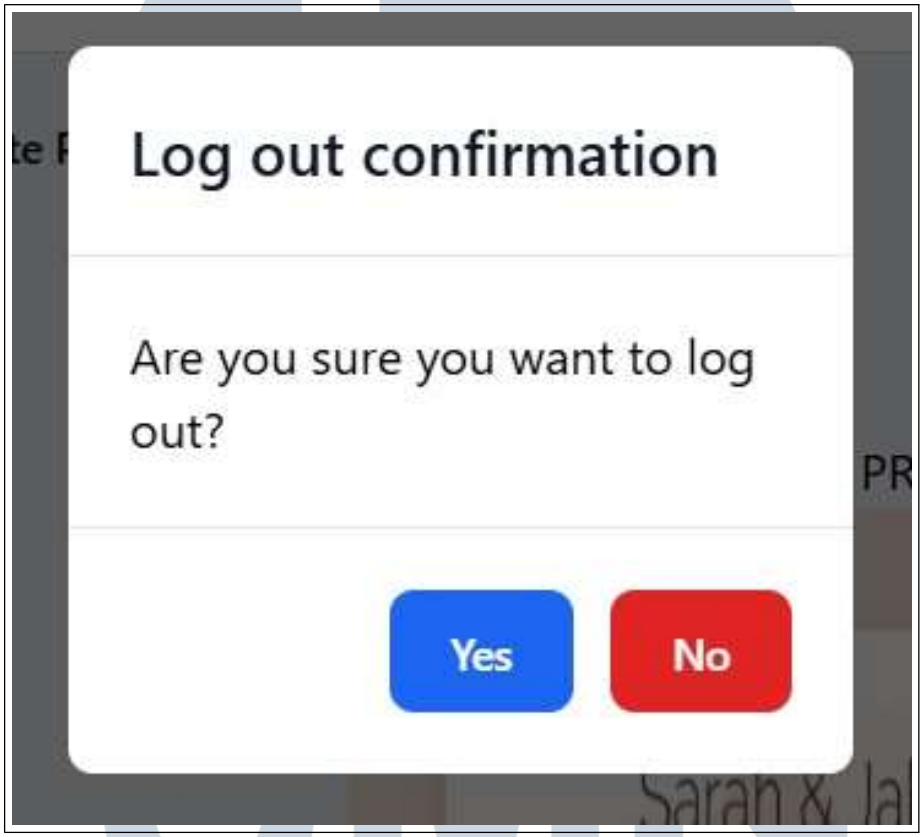
Selain itu, *modal* ini juga disajikan dengan fitur *dark mode* atau mode gelap untuk mengikuti konsistensi pada halaman Template. Dapat dilihat mode gelap dari *modal* ini di Gambar 3.33. Gambar 3.34 merupakan alur dari komponen *delete*.



Gambar 3.34. Delete Component

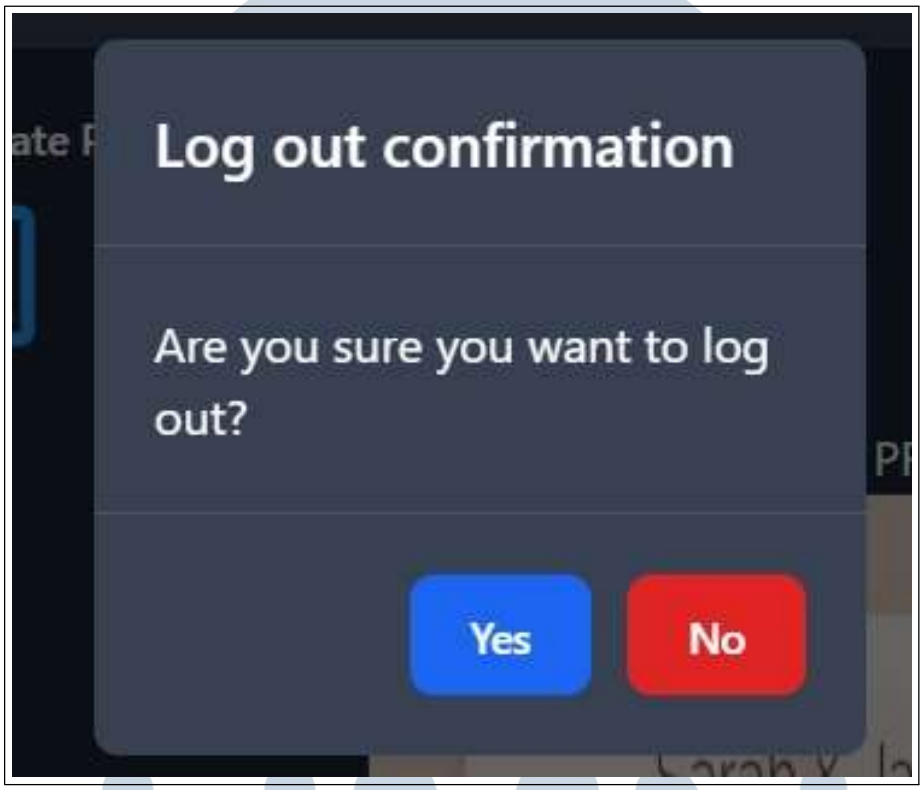
L. Logout Confirmation

Modal pop-up Logout Confirmation seperti pada gambar 3.35 muncul ketika pengguna menekan tombol *Logout* pada navbar di halaman admin dashboard. Fungsi dari tombol ini yaitu untuk menghapus *session* dari sistem lalu menghapus token dari *cookie*. Setelah itu, pengguna akan dikembalikan ke halaman Login.



Gambar 3.35. Logout

Modal ini juga disediakan dengan fitur *dark mode* atau mode gelap untuk mengikuti konsistensi seperti pada halaman lainnya. Mode gelap ditampilkan seperti pada Gambar 3.36



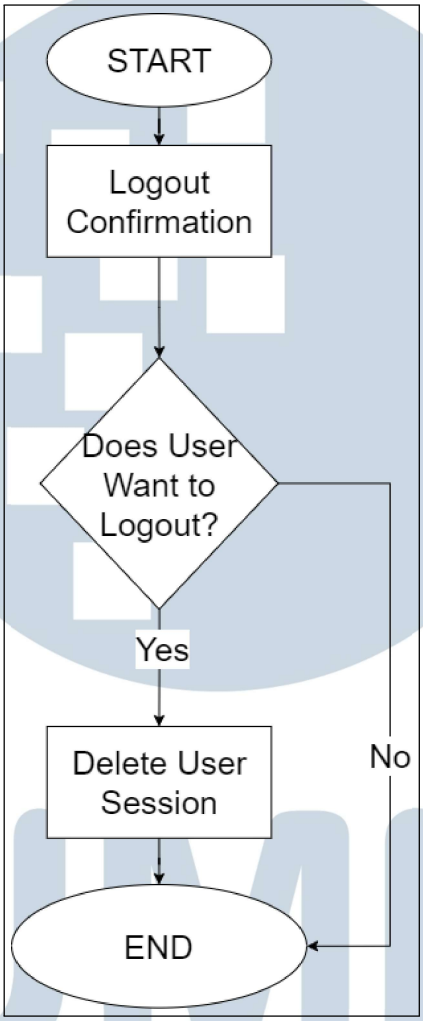
Gambar 3.36. Logout (Dark Mode)

```
const res = await fetch(url + "/api/v1/auth/logout", {
    method: "POST",
    headers: {
        Authorization: "Bearer " + Tokens.accessToken,
        "Content-Type": "application/json",
    },
    body: JSON.stringify({ refreshToken: Tokens.refreshToken }),
    credentials: "include",
});
```

Kode 3.13: Logout Library

Setelah pengguna menekan tombol *yes*, sistem akan langsung menjalankan kode 3.13 dengan melakukan *request* ke *backend* untuk menghapus sesi pengguna dari *backend*. Setelah sesi dihapus, sistem juga akan menghapus *cookie* dari *browser*. Setelah seluruh proses tersebut selesai, pengguna akan secara otomatis dialihkan

kembali ke halaman Login sebagai langkah akhir dari proses logout. Gambar 3.37 merupakan alur dari komponen *delete*.



Gambar 3.37. Flowchart Logout

3.5 Kendala dan Solusi yang Ditemukan

Berikut merupakan kendala yang dialami selama masa magang beserta dengan solusi ditemukan:

A. Kendala

- Kurangnya komunikasi antar divisi, khususnya antara *frontend*, *backend*, dan *UI/UX*, yang mengakibatkan banyak informasi penting tidak tersampaikan

dengan jelas dan menimbulkan miskomunikasi saat pengembangan berlangsung.

- Sering terjadi perubahan kebutuhan dari *backend* yang menyebabkan *developer frontend* harus menyesuaikan ulang implementasi yang sudah selesai, dan ini menghambat efisiensi waktu serta memperpanjang proses pembangunan.
- Kurangnya komunikasi antar divisi, khususnya antara *frontend*, *backend*, dan *UI/UX*, yang mengakibatkan banyak informasi penting tidak tersampaikan dengan jelas dan menimbulkan miskomunikasi saat pembangunan berlangsung.

B. Solusi

- Mengadakan koordinasi rutin (misalnya, weekly meeting) antar divisi, khususnya *frontend*, *backend*, dan *UI/UX*, agar setiap perubahan atau kendala dapat segera dikomunikasikan dan diselesaikan bersama.
- Melakukan dokumentasi kebutuhan secara lebih formal dan konsisten di awal *sprint*, serta menetapkan batas perubahan setelah tahap perencanaan agar scope pekerjaan lebih jelas dan tidak berubah-ubah di tengah pengerjaan.