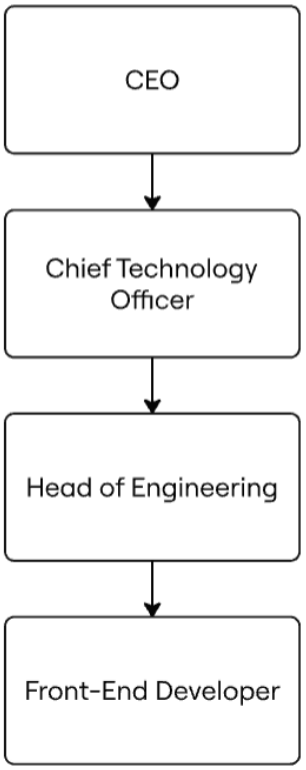
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

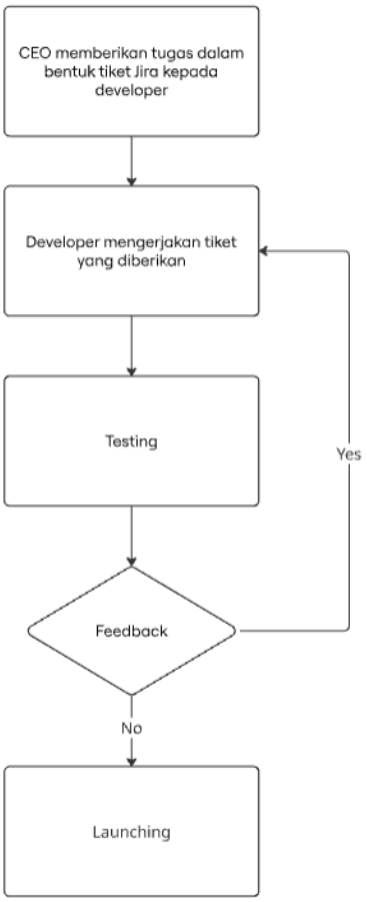


Gambar 3.1 Kedudukan dan Organisasi

CEO bertanggung jawab atas penentuan arah strategis perusahaan dan pengawasan seluruh tim operasional perusahaan. *Chief Technology Officer* fokus pada pengembangan dan pengelolaan strategi teknologi perusahaan. *Head of Engineering* bertanggung jawab atas koordinasi tim *IT* dan memastikan proyek pengembangan *software* berjalan sesuai rencana. Terakhir, *Frontend Developer* berfokus pada pengembangan antarmuka menggunakan teknologi *web* modern, memastikan *user experience* yang optimal dan performa aplikasi yang efisien. Selama menjalani magang di PT Ahli Karya, penulis menempati posisi sebagai

Frontend Developer Intern. Dalam peran ini, penulis dibimbing oleh Ibu Nadila Setiabudiarto, yang bertindak sebagai *supervisor*. Beliau mengkoordinasikan tugas-tugas yang diberikan dan memberikan arahan serta bimbingan terkait pekerjaan yang harus diselesaikan.

3.1.1 Alur Pengembangan Platform Smplhr



Gambar 3.2 Alur Pengembangan Platform Smplhr

Alur pembuatan *platform* Smplhr dapat dilihat pada gambar diatas. Proses dimulai ketika *CEO* memberikan tugas kepada *developer* dalam bentuk tiket yang dibuat di *Jira*, sebuah *platform* untuk mengelola proyek dan pelacakan tugas secara kolaboratif. Setiap tiket umumnya berisi deskripsi, referensi, atau informasi lain yang membantu *developer* dalam memahami dan melaksanakan tugas yang diberikan. Setelah menerima tiket, *developer* akan

mulai mengerjakan tugas sesuai dengan instruksi yang tercantum. Setelah implementasi selesai, hasil pekerjaan akan masuk ke tahap *testing* untuk di *review*. Apabila ditemukan kekurangan atau hal yang perlu diperbaiki, maka *feedback* akan diberikan dan tugas akan dikembalikan kepada *developer* untuk disesuaikan hingga memenuhi ekspektasi. Jika tidak ada revisi lebih lanjut, maka tiket dinyatakan selesai dan dapat dilanjutkan ke tahap *launching*.

3.2 Tugas dan Uraian Kerja Magang

Selama menjalani masa magang di PT Ahli Karya Improovi Teknologi, penulis berkontribusi dalam proses pengembangan *frontend* dari *platform* Smplhr. Pekerjaan yang dilakukan mencakup berbagai aspek, mulai dari *slicing* desain antarmuka berdasarkan file *Figma*, implementasi fitur interaktif, integrasi *API*, hingga *bug fixing*. *Slicing* merupakan proses mengubah desain *UI* menjadi struktur *HTML* dan *CSS* yang dapat diimplementasikan dalam kode. Dalam pengembangannya, penulis merujuk pada *existing code platform* Smplhr yang sudah tersedia sebelum penulis bergabung dengan perusahaan, terutama saat melakukan penyesuaian tampilan halaman. Namun, untuk fitur atau halaman yang belum diimplementasikan sebelumnya, penulis mengerjakannya dari awal berdasarkan desain yang diberikan oleh tim *UI/UX*. Penulis juga terlibat dalam peningkatan *user experience* dan optimasi performa melalui pengembangan fitur seperti *popup* konfirmasi, *lazy loading*, serta penyusunan *email template* dinamis menggunakan *Handlebars*. Rincian tugas mingguan secara lengkap dapat dilihat pada tabel berikut.

Tabel 3.1 Rangkuman Pekerjaan Magang Tiap Minggu

Minggu Ke-	Pekerjaan yang dilakukan
1	● *Explore Mockoon* dan *Tailwind* ● Memahami desain *website* di *Figma*
2	● Mempelajari kode halaman *candidate* ● *Slicing menu task* ● Penyesuaian proses *sign-up*

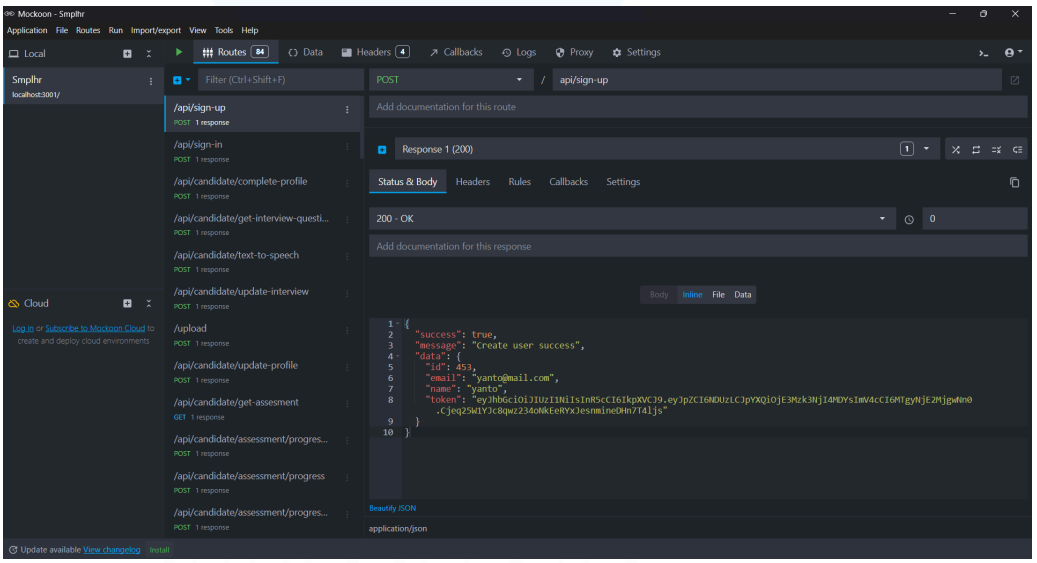
3	● Mempelajari kode halaman *company* ● Memperbaiki tampilan halaman *company*
4	● Mengerjakan halaman 404 dan *routing* ● Melakukan penyesuaian pada *quill editor*
5	● Integrasi *API* untuk menampilkan data di *modal preview job posting*
6	● Mengerjakan *career page* ● Integrasi *job detail API* ● Memperbaiki *bug sorting*
7	● Menambahkan *popup* konfirmasi di halaman *edit profile*
8	● *Slicing menu schedule* ● Menambahkan fitur *pagination, cancel,* dan *search* di *menu schedule*
9	● Implementasi *lazy loading* pada *menu talent search* ● *Slicing talent search card*
10	● Menambahkan fitur pada *menu jobs* serta *tooltip* di beberapa *field*
11	● Mengerjakan *modal quick tour* ● *Slicing menu Asessment*
12	● Membuat *template email Handlebars*
13	● Slicing halaman *add employee*
14	● Slicing halaman *onboarding*

1. Explore Mockoon dan Tailwind serta Mempelajari Design di Figma

   Mockoon adalah aplikasi *open-source* yang digunakan untuk membuat *mock API* secara lokal, yang memudahkan proses pengembangan tanpa harus bergantung pada *API* eksternal [2]. Penulis belum pernah menggunakan *Mockoon* sebelumnya, sehingga perlu waktu untuk memahami dasar penggunaannya. Penulis belajar melakukan simulasi *API* pada proyek *React* dengan menggunakan data *dummy* melalui konfigurasi di *Mockoon*. Setiap *route endpoint* yang disimulasikan dikonfigurasi dengan menentukan metode *HTTP, path*

URL, status response, serta isi dari *body response*. Untuk mendapatkan struktur data pada *body response*, penulis menggunakan dua cara:

1. Jika *endpoint API* sudah tersedia, penulis mengambil data dari *website* internal perusahaan (*staging*) yang digunakan untuk *testing* hasil pengembangan sebelum di *launching*. Melalui *tab Network* pada *browser developer tools*, penulis mengamati *API request* yang dikirim dari *frontend* ke *server* dan menyalin struktur *response JSON* yang ditampilkan sebagai referensi untuk konfigurasi *route Mockoon*.
2. Jika *endpoint* yang dibutuhkan belum tersedia, penulis berkoordinasi dengan tim *backend* untuk menyusun struktur dan isi *response API* berdasarkan kebutuhan *frontend*, sebelum membuat konfigurasi *route* baru di *Mockoon*.



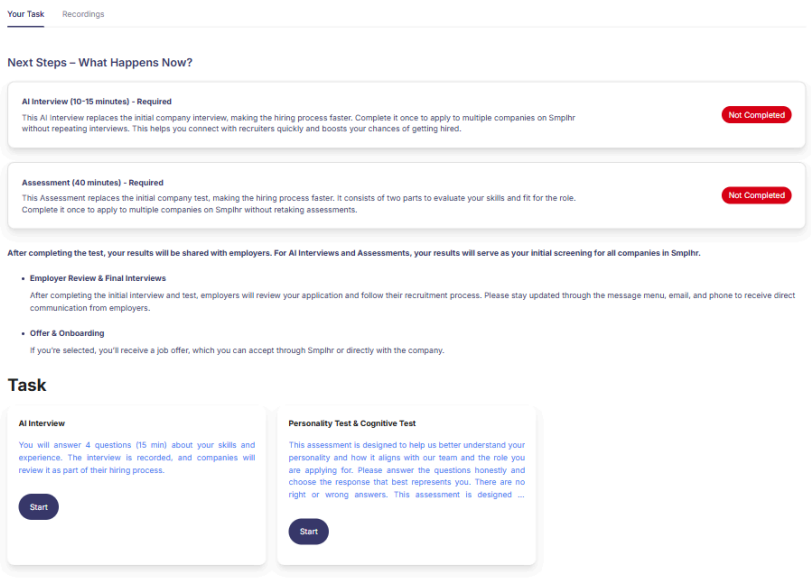
Gambar 3.3 Tampilan Aplikasi Mockoon

Untuk pengembangan *website*, *tech stack* yang digunakan adalah *React*, dan *Tailwind* sebagai *CSS framework*. Penulis telah memiliki pengalaman sebelumnya dengan *React* dan *Tailwind* di perkuliahan, sehingga proses adaptasi berjalan cukup cepat. Penulis juga diberi akses ke desain *UI* di *Figma* dan diminta untuk memahami

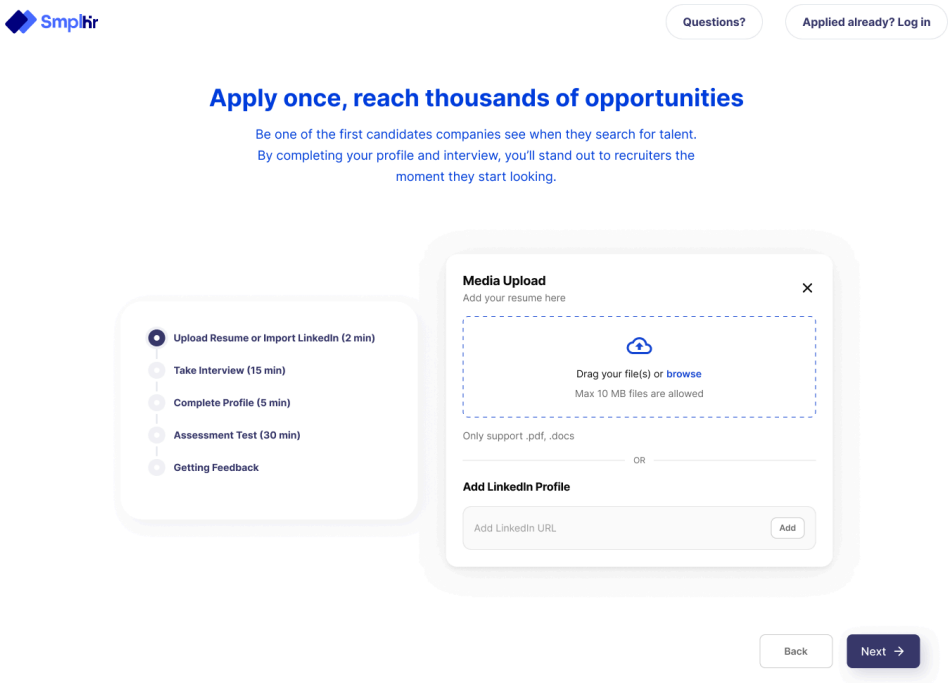
layout serta struktur tampilan sebelum mengimplementasikannya ke dalam kode.

2. Slicing Menu Task dan Penyesuaian Proses Sign-Up

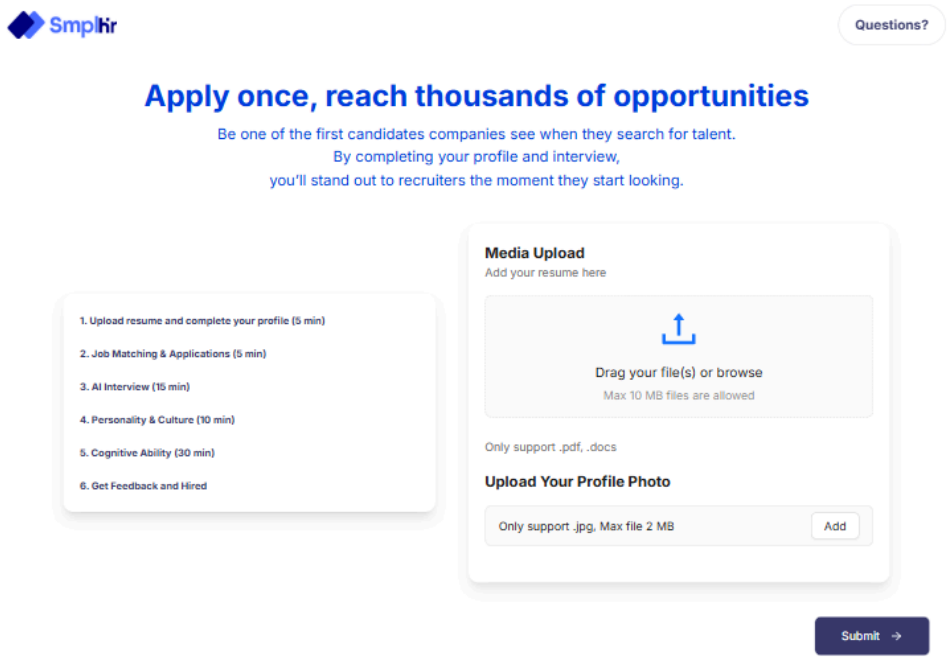
   Penulis mulai mempelajari *existing code* untuk halaman *candidate*. Penulis membantu proses *slicing UI* untuk bagian *menu task* serta melakukan perubahan pada proses saat kandidat melakukan *sign-up*. Proses *sign-up* menggunakan konsep *step-by-step*, di mana *user* perlu menyelesaikan setiap tahapan secara berurutan. Alur ini dikontrol menggunakan sebuah variabel *step* untuk menampilkan komponen sesuai dengan tahapan yang sedang dijalani. Setiap nilai pada *step* mewakili satu tahap tertentu, dan tiap tahap memiliki komponen tersendiri yang sesuai dengan isi dan fungsinya. Ketika *user* menyelesaikan satu tahap, nilai *step* akan di *increment* untuk menampilkan komponen tahap selanjutnya. Perubahan tampilan proses *sign-up* dapat dilihat pada gambar 3.5 dan 3.6.



Gambar 3.4 Tampilan Menu Task



Gambar 3.5 Tampilan Proses Registrasi Sebelum diubah



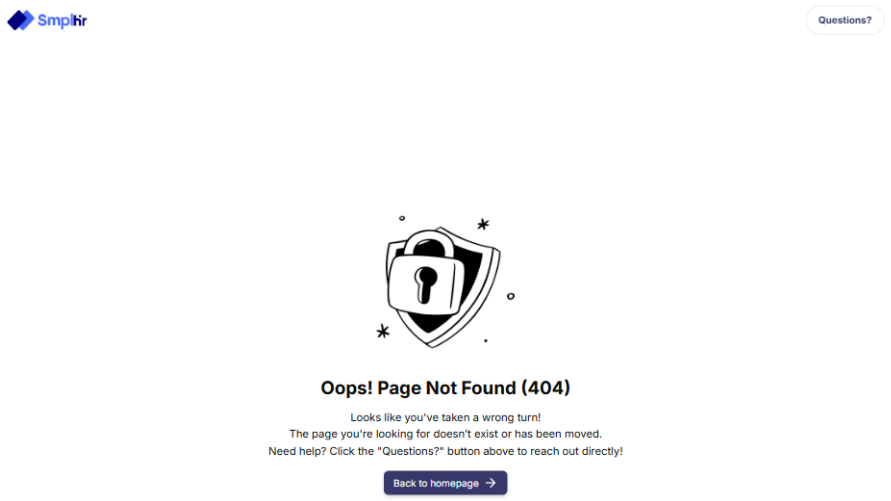
Gambar 3.6 Tampilan Proses Registrasi Setelah diubah

3. Perbaikan Tampilan Halaman Company

Penulis dipindahkan untuk membantu pengembangan pada halaman *company*. Fokus pekerjaan meliputi perbaikan *typo* dan penyesuaian *spacing* elemen agar sesuai dengan desain yang ada. Sambil melakukan ini, penulis juga mempelajari struktur kode halaman *company* untuk memahami konteks pengembangan yang lebih luas.

4. Mengerjakan Halaman 404

Penulis ditugaskan untuk membuat halaman 404 lengkap dengan *routing*-nya, agar *user* diarahkan ke halaman *error* saat mencoba mengakses *path URL* yang tidak tersedia. *Routing* ini diatur menggunakan *createBrowserRouter*, sebuah fungsi dari *library react-router-dom* yang mengelola rute aplikasi modern *react* serta mendukung *nested routing* dan *data fetching* melalui *loader* [3].



Gambar 3.7 Tampilan Halaman 404

Selain itu, penulis juga diminta untuk menyesuaikan tampilan komponen *Quill Editor*. Awalnya, penulis hanya diminta untuk menaikkan tinggi *default editor* agar tampilan teks menjadi lebih luas. Namun, setelah berdiskusi lebih lanjut dengan *supervisor*, penulis

menyarankan agar tinggi *editor* tidak statis, melainkan dapat menyesuaikan secara otomatis mengikuti jumlah konten yang dimasukkan oleh *user*. Hal ini bertujuan untuk meningkatkan kenyamanan dalam mengetik dengan menghindari kemunculan *scrollbar* pada *editor*. Penulis menggunakan *useRef* dan *useEffect*, dua *hook* bawaan *React*. *useRef* digunakan untuk mengambil referensi langsung ke elemen *DOM*, sementara *useEffect* digunakan untuk mendeteksi perubahan konten dan memperbarui tinggi elemen saat konten melebihi tinggi awal *editor* (300px). Kombinasi kedua *hook* ini memungkinkan penyesuaian ukuran *editor* secara responsif tanpa mengganggu performa aplikasi.

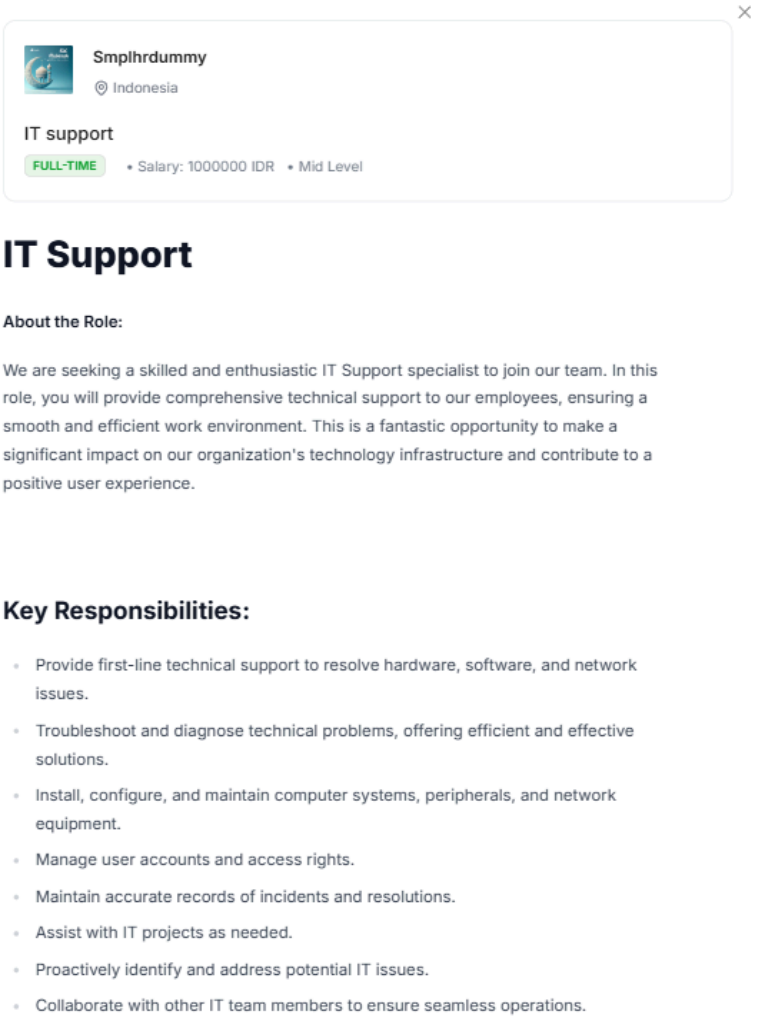
5. Integrasi API

Penulis diminta untuk menampilkan data lowongan pekerjaan ke dalam *modal preview job*. Langkah pertama yang dilakukan adalah *slicing* tampilan *modal* berdasarkan desain di *Figma*, kemudian dilanjutkan dengan proses integrasi *API*. Untuk *fetching* data, penulis menggunakan *tanstack react-query,* sebuah *library* manajemen *state server* dan *data fetching* di *React* yang menyediakan fitur seperti *caching* otomatis, *refetching data*, manajemen *loading* dan *error state*, serta deduplikasi *request*. Sebelumnya, penulis hanya memiliki pengalaman menggunakan *fetch API* bawaan dari *JavaScript*. Oleh karena itu, penggunaan *react-query* merupakan hal baru yang membutuhkan pemahaman tambahan, terutama dalam membedakan fungsi antara *useQuery* dan *useMutation*. *useMutation* umumnya digunakan untuk operasi *write* atau mengubah data seperti *POST, PUT,* dan *DELETE*, sedangkan *useQuery* digunakan untuk operasi *read* data seperti *GET*. Gambar 3.7 menunjukkan kode yang dipakai oleh penulis untuk mendapatkan *response* data dari *API* menggunakan *useMutation*.

```
export default function PreviewModal({ isOpen, onClose, jobData }) {
    const [data, setData] = useState({});

    const { isPending, mutateAsync } = useMutation({
        mutationFn: () => careerSiteData(),
    });

    const fetchData = async () => {
        try {
            const response = await mutateAsync();
            setData(response?.data);
        } catch (error) {
            message.error(error);
        }
    };

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

Gambar 3.8 Kode untuk Integrasi API

Dalam kasus ini, penulis menggunakan *hook useMutation* karena *endpoint API* untuk menampilkan *preview job* menggunakan metode *HTTP POST*. Untuk penanganan *error*, penulis menerapkan blok *try-catch*, merujuk pada struktur *existing code* sebelumnya.

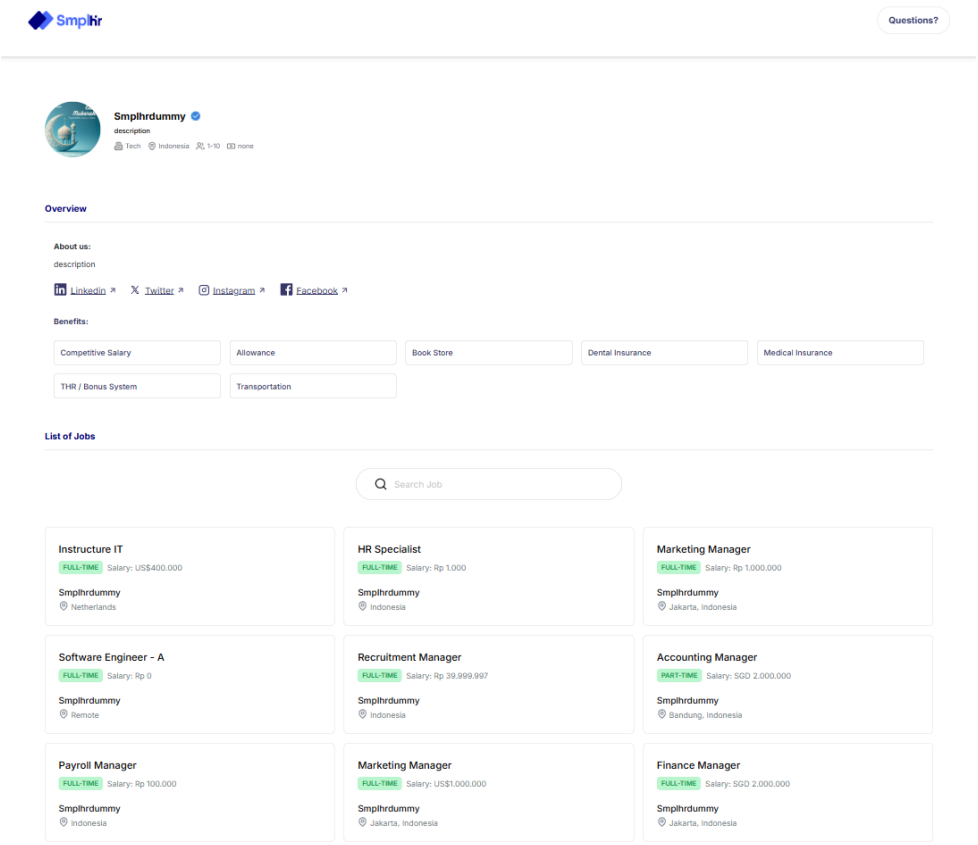


Gambar 3.9 Tampilan Modal Preview Job

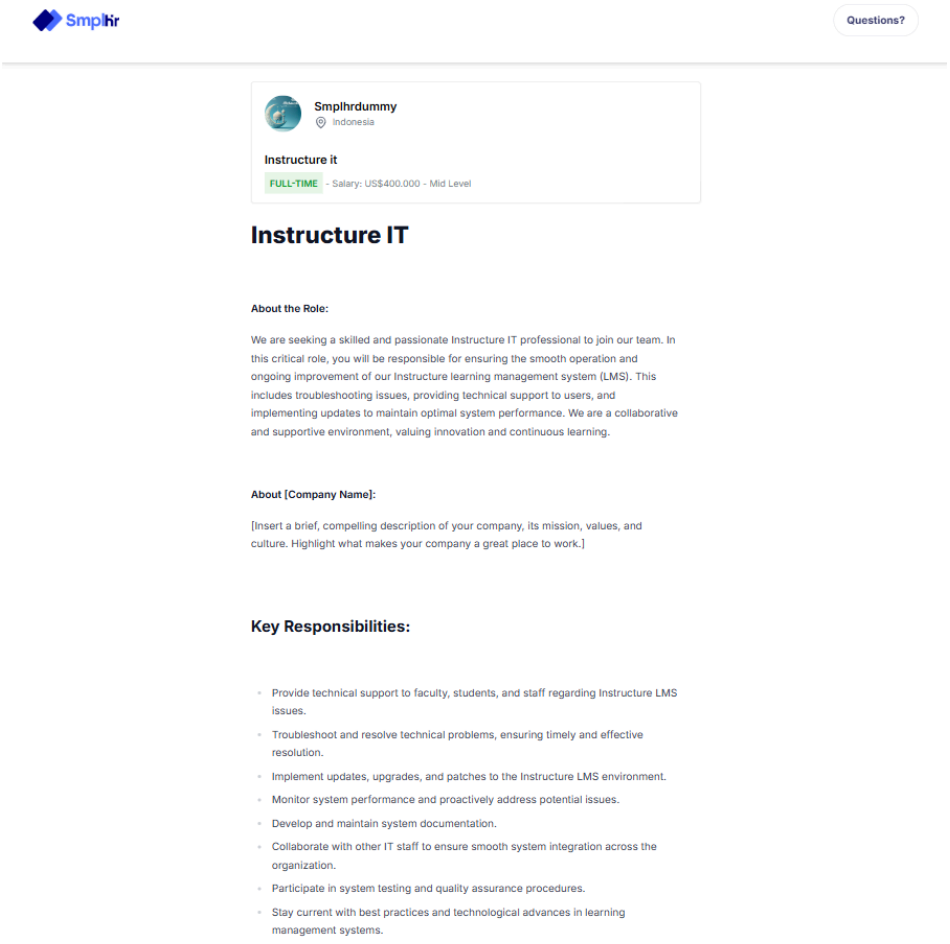
6. Career Page, Halaman Job Detail dan Fix Bug Sorting

Penulis diminta untuk mengerjakan *career page* yang menampilkan *detail* sebuah perusahaan. Halaman ini dibagi menjadi dua bagian utama. Bagian pertama berisi *overview* dari perusahaan, seperti nama, deskripsi, *benefit*, dan informasi umum lainnya. Bagian kedua menampilkan daftar lowongan pekerjaan yang dibuka oleh perusahaan tersebut. Di bagian lowongan, penulis mengimplementasikan beberapa fitur tambahan seperti *search bar, pagination*, dan juga navigasi ke halaman *job detail* yang sesuai ketika

salah satu lowongan diklik oleh *user*. Penulis juga memperbaiki *bug* terkait *sorting* tanggal, dengan mengubah tipe data dari *string* ke *date*.

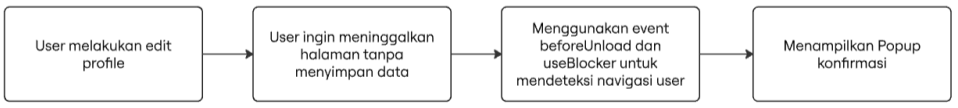


Gambar 3.10 Tampilan Career Page



Gambar 3.11 Tampilan Halaman Job Detail

7. Implementasi Popup Konfirmasi pada Halaman Edit Profile



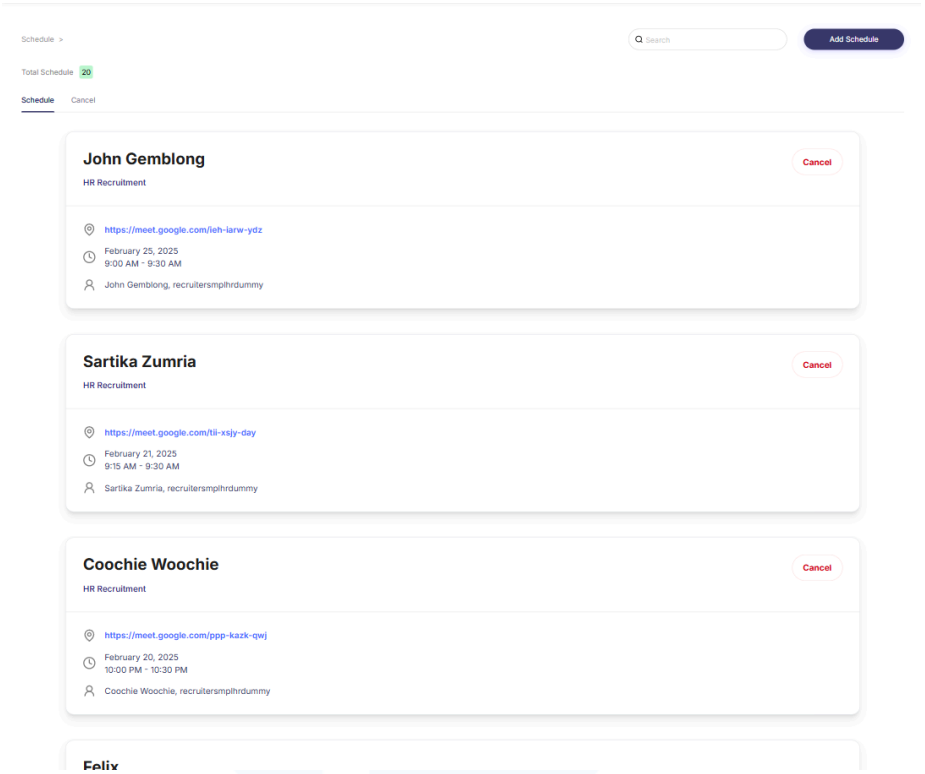
Gambar 3.12 Alur Implementasi Popup Konfirmasi

Penulis diberikan tugas untuk meningkatkan *user experience* pada halaman *edit profile.* Salah satu fitur yang diminta adalah *popup* konfirmasi ketika *user* mencoba meninggalkan halaman (*refresh, close tab,* atau berpindah *menu*) tanpa menyimpan perubahan data profil.

Awalnya penulis menggunakan *event beforeUnload*, sebuah event dalam *JavaScript* yang umumnya digunakan untuk mencegah *user* meninggalkan halaman secara tidak sengaja dengan menampilkan konfirmasi. Namun ternyata *event* tersebut tidak berfungsi saat user hanya berpindah *menu* dikarenakan *react* menggunakan *client side routing*, di mana navigasi antar halaman dilakukan tanpa *reload* halaman. Untuk mengatasi hal ini, penulis melakukan *searching* dan menemukan solusi menggunakan *hook useBlocker* dari *library react-router-dom*. *Hook* ini memungkinkan pemblokiran navigasi dan menampilkan *popup* konfirmasi saat *user* mencoba meninggalkan halaman.

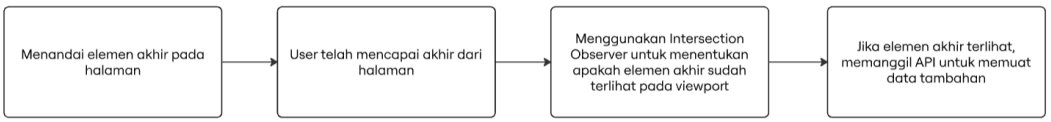
8. Slicing menu schedule

   Penulis diminta untuk menyesuaikan tampilan *menu schedule* berdasarkan desain baru. Dikarenakan layout yang baru tidak terlalu berbeda dengan yang lama, maka penulis dapat menggunakan referensi dari kode sebelumnya. Penulis juga diminta untuk mengembangkan fitur *pagination, search,* dan *cancel schedule* pada *menu schedule*. Pengerjaan ini relatif mudah karena penulis sudah pernah membuat fitur serupa sebelumnya.



Gambar 3.13 Tampilan Menu Schedule

9. Implementasi Lazy Loading pada Menu Talent Search



Gambar 3.14 Alur Implementasi Lazy Loading

Penulis mengerjakan berbagai tugas yang bertujuan untuk mengoptimalkan performa *website*. Salah satunya adalah implementasi *lazy loading* pada *menu talent search*. Menu ini memungkinkan *user* untuk mencari kandidat berdasarkan kriteria tertentu. Karena data yang ditampilkan bisa sangat banyak, maka untuk menghindari beban berlebih pada performa *website*, penulis menggunakan *Intersection Observer*, sebuah *API* untuk mendeteksi saat suatu elemen terlihat di *viewport*. Ketika elemen akhir sudah terdeteksi, *API* akan dipanggil

untuk memuat data tambahan. Penulis juga melakukan *slicing card talent search* sesuai dengan desain yang diinginkan

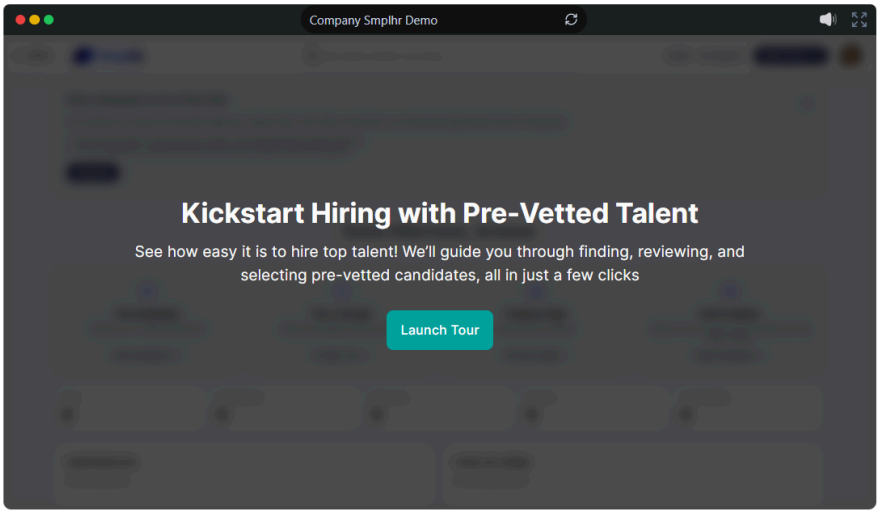
Gambar 3.15 Tampilan Talent Search Card

10. Implementasi Multi-Selection pada Tabel Menu Job

Pada *menu jobs*, penulis mengimplementasikan fitur *multi-selection* pada tabel data lowongan pekerjaan. Tabel ini menggunakan *library Ant Design (antd)* yang sebelumnya belum dikenal oleh penulis, sehingga diperlukan waktu untuk membaca dokumentasi dan memahami cara kerja komponen tabel dan fitur seperti *row selection.* Salah satu tujuan dari *multi-selection* ini adalah agar *user* bisa melakukan *bulk action,* seperti menghapus beberapa *job* sekaligus atau mengubah status *job* menjadi *active* atau *inactive*. Namun, penulis menghadapi kendala saat melakukan tes di lokal, di mana meskipun status sudah diubah di sisi *frontend*, data dari *endpoint Mockoon* tetap tidak berubah dan selalu kembali ke kondisi awal. Hal ini terjadi karena *Mockoon* hanya menyediakan data statis, sehingga data awal pada *route endpoint* di *Mockoon* tidak bisa diubah. Akibatnya, fitur tidak dapat diverifikasi langsung di lingkungan lokal. Sebagai solusi, penulis melihat contoh implementasi kode sebelumnya untuk *action update status* pada satu data, lalu mengadaptasinya agar bisa digunakan pada banyak data sekaligus. Validasi akhir tetap dilakukan setelah fitur tersebut di *deploy* ke *staging site*. Selain itu, penulis juga menambahkan *tooltip* pada beberapa *field* untuk menampilkan deskripsi singkat atau instruksi kepada *user*.

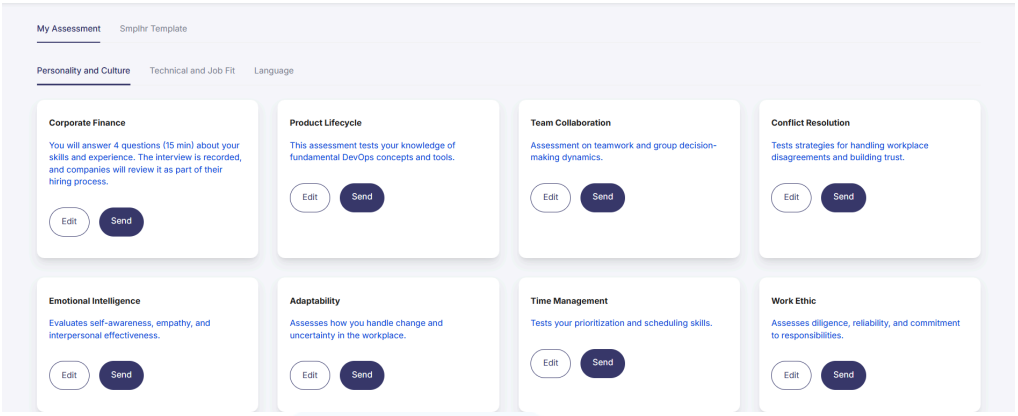
11. Modal Quick Tour dan Slicing Menu Assessment

Untuk membantu user memahami cara penggunaan *website*, penulis diminta untuk membuat *modal quick tour* yang menampilkan *demo* singkat. *Demo* yang ditampilkan sudah disediakan oleh *Layerpath*, sebuah *platform* yang digunakan untuk membuat *walkthrough* interaktif. *Demo* dari *Layerpath* ditampilkan dalam *modal* menggunakan elemen *iframe*, yang berfungsi untuk menampilkan halaman *web* lain di dalam *page*. Dengan menggunakan *iframe*, *demo* bisa langsung ditampilkan tanpa perlu integrasi *API* tambahan. *Modal* ini akan muncul secara otomatis saat sesi pertama *user*, atau saat user mengklik *icon* bantuan.



Gambar 3.16 Tampilan Modal Quick Tour

Penulis juga melakukan penyesuaian *layout* pada *menu assessment* berdasarkan desain terbaru.



Gambar 3.17 Tampilan Menu Assessment

12. Membuat Template Email dengan File Handlebars

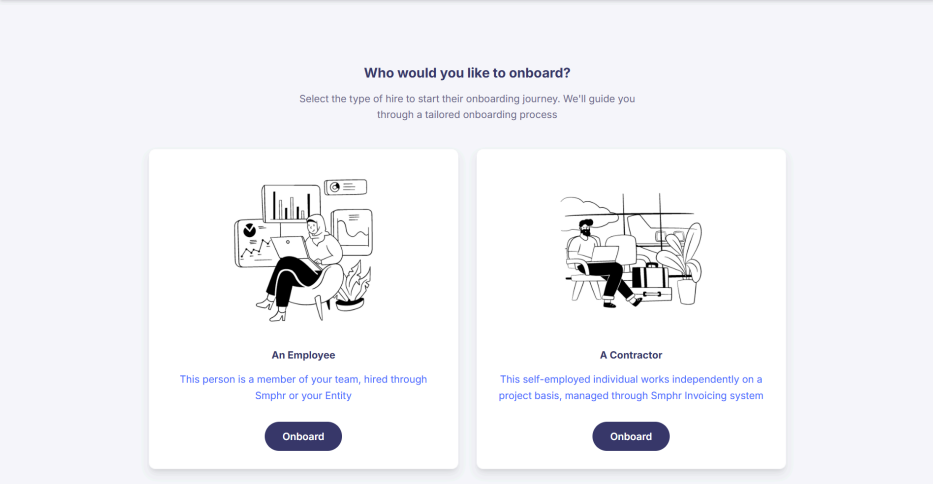
Penulis diminta untuk membuat beberapa *template email* menggunakan *Handlebars*. *Handlebars* adalah sebuah *templating language* yang memungkinkan pembuatan *HTML* dinamis berdasarkan input data tertentu [4]. Salah satu ciri khas *Handlebars* yang membedakannya dari struktur *HTML* biasa adalah penggunaan tanda kurung ganda {{}} yang digunakan untuk menyisipkan data ke dalam *template*. *Template email* yang dibuat mencakup berbagai skenario, seperti konfirmasi *sign-up,* undangan *interview*, hingga notifikasi status lamaran pekerjaan. Kode file *handlebars* untuk *template email* dapat dilihat pada gambar 3.17.

```
<html
  <body bgcolor="#ffffff">
    <table cellpadding="0" cellspacing="0" width="100%" bgcolor="#ffffff">
      <tr>
        <td align="center">
                <h1>Interview Invitation for
                  {{jobName}}
                  at
                  {{companyName}}</h1>
                <p>Dear {{interviewerName}},</p>
                <p>
                  I hope this email finds you well. We have scheduled an
                  interview with
                  {{candidateName}}
                  for the
                  {{jobName}}
                  position at
                  {{companyName}}. Please find the interview details below:
                </p>

                <h3>Interview Details:</h3>
                <ul>
                  <li><strong>Candidate </strong> {{candidateName}}</li>
                  <li><strong>Position </strong> {{jobName}}</li>
                  <li><strong>Day/Date: </strong> {{interviewerDate}}</li>
                  <li><strong>Time: </strong> {{interviewTime}}</li>
                  <li><strong>
                      {{#if (eq locationType "VIRTUAL")}}
                        Meeting Link:
                      {{else}}
                        Location:
                      {{/if}}</strong>
                    {{#if (eq locationType "VIRTUAL")}}
                      <a href="{{location}}">{{location}}</a>
                    {{else}}
                      {{location}}
                    {{/if}}</li>
```

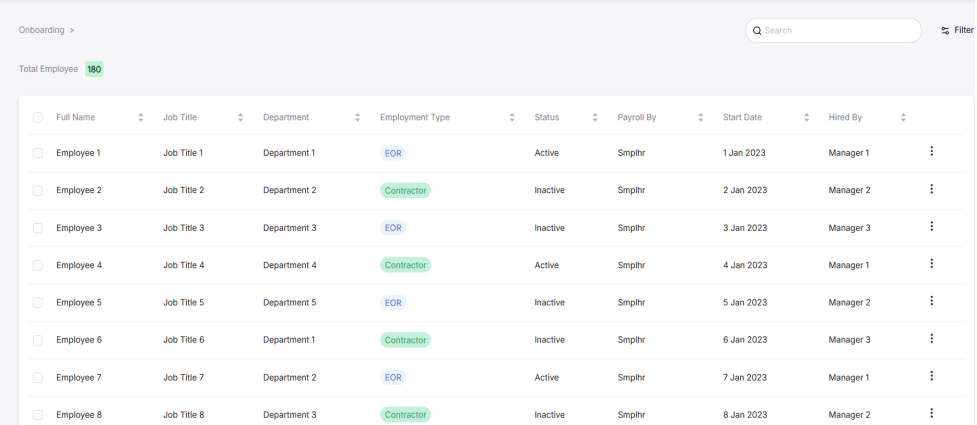
Gambar 3.18 Kode File Handlebars

13. Slicing Halaman Add Employee dan Halaman Onboarding

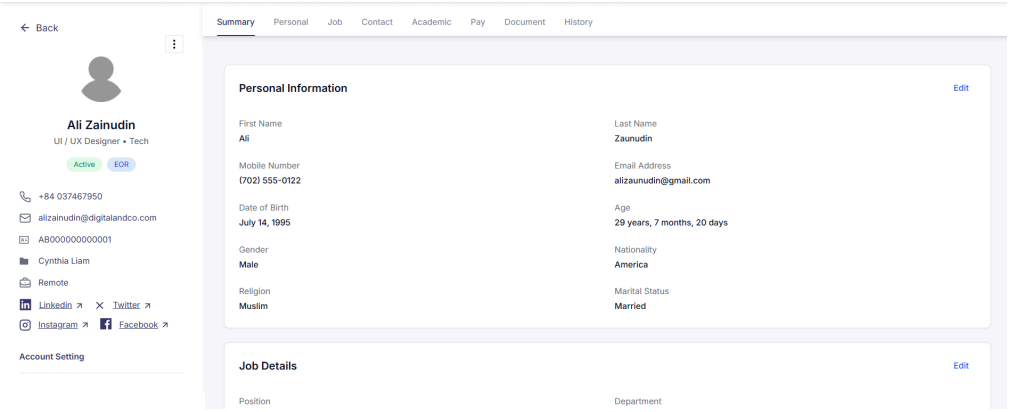
Pada masa akhir magang, penulis mengerjakan *slicing* tampilan untuk halaman *Add Employee* dan *Onboarding* berdasarkan desain yang telah disiapkan oleh tim *UI/UX*. Karena kedua halaman ini merupakan halaman baru, penulis hanya diminta untuk fokus pada proses *slicing* sesuai desain, tanpa perlu melakukan integrasi *API*. Setelah proses *slicing* selesai, penulis memastikan kembali kepada tim *UI/UX* bahwa tampilan serta seluruh fungsi visual dari fitur yang diinginkan sudah berjalan dengan baik dan sesuai ekspektasi.



Gambar 3.19 Tampilan Halaman Add Employee



Gambar 3.20 Tampilan Halaman Onboarding



Gambar 3.21 Tampilan Halaman Onboarding Detail

3.3 Kendala yang Ditemukan

- Penulis mengalami kendala dalam penggunaan beberapa *tools* dan *library* yang belum pernah digunakan sebelumnya, seperti *Mockoon, Antd, dan react-query.*
- *Event beforeUnload* tidak berfungsi saat user melakukan navigasi antar *menu* dalam aplikasi *React*, karena *React* menggunakan *client-side routing* yang tidak melakukan *reload* halaman.
- *Response body* pada *route* yang dikonfigurasi di *Mockoon* bersifat statis, sehingga *testing* untuk fitur tertentu tidak bisa dilakukan secara lokal.
- Dokumentasi *endpoint API* yang dibutuhkan kurang lengkap, sehingga penulis mengalami kesulitan dalam memahami struktur data yang diperlukan untuk proses integrasi maupun simulasi menggunakan *Mockoon*.

3.4 Solusi atas Kendala yang Ditemukan

- Penulis membaca dokumentasi resmi, mencari referensi dari internet, serta menonton video *tutorial* untuk mempercepat proses pemahaman dan implementasi.
- Penulis menggabungkan *event beforeUnload* dengan *hook useBlocker* dari *react-router-dom*, yang memungkinkan pemblokiran navigasi internal dan menampilkan *popup* konfirmasi saat user mencoba meninggalkan halaman.
- Penulis melakukan pengujian di *staging site*, di mana data dapat dimodifikasi secara dinamis. Penulis juga merujuk pada implementasi kode sebelumnya yang memiliki fungsi serupa, lalu mengadaptasinya agar fitur tetap dapat berjalan dengan baik meskipun dijalankan di lokal.
- Penulis melakukan inspeksi langsung melalui *Network tab* di *browser* untuk menyalin *response* data dari *staging site*, serta berkoordinasi

dengan tim *backend* guna memperoleh struktur data yang sesuai dan menyusunnya sebagai dokumentasi internal pribadi.