BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Pelaksanaan magang diselenggarakan pada PT Kalbe Farma Tbk, perusahaan bergerak di sektor farmasi yang berfokus pada inovasi berbasis teknologi kesehatan dan data. Kegiatan magang berlangsung di fungsi CDT yang berperan dalam pengembangan infrastruktur sistem internal guna mendukung efisiensi operasional dan pengelolaan pengetahuan perusahaan. Dalam struktur departemen, pembimbing lapangan ditunjuk dari tim Software Engineer dengan peran sebagai lead software engineer, Russell Otniel Tjakra, untuk proyek pengembangan ELN. Pelaksanaan magang berlangsung penuh secara daring menggunakan platform kolaborasi terpadu serta manajemen proyek berbasis web. Seluruh komunikasi operasional dilakukan melalui aplikasi pesan instan *real-time*, sementara diskusi teknis dan evaluasi berkala dilaksanakan dalam format pertemuan jarak jauh (*remote meeting*). Koordinasi harian berpusat pada sistem manajemen proyek yang mengintegrasikan *backlog*, *progress tracking*, serta dokumentasi teknis sehingga supervisi dapat dilaksanakan secara konsisten meskipun tanpa tatap muka langsung. Pendekatan kolaborasi daring juga memfasilitasi partisipasi pada *retrospective* dan *sprint planning* yang memastikan penyelarasan tugas magang dengan prioritas pengembangan proyek secara berkelanjutan. Keterlibatan aktif pada diskusi teknis, tinjauan kode, serta testing kolaboratif juga menjadi bagian dari proses koordinasi yang dilakukan selama masa magang.

3.2 Tugas yang Dilakukan

Selama masa pelaksanaan kerja magang, berbagai tugas telah dilaksanakan sesuai dengan fase dan proyek yang ditugaskan, terutama pada pengembangan sistem ELN dan AI Chatbot Internal di PT Kalbe Farma Tbk. Setiap tugas bersifat eksploratif dan iteratif, dengan tujuan memastikan bahwa solusi yang dirancang dapat diimplementasikan secara teknis serta memenuhi kebutuhan user. Adapun uraian tugas yang dilaksanakan adalah sebagai berikut:

1. Kontribusi pada Electronic Lab Notebook (ELN)
   Pada proyek ELN, kontribusi berfokus pada pengembangan dan penyempurnaan

modul-modul konfigurasi sistem, meliputi User & Roles Management, Admin & Tenant Management, serta Formula List. Selain modul utama, terdapat pula beberapa pekerjaan eksploratif berupa *proof of concept* (POC) dan fitur yang tidak dilanjutkan ke tahap final, seperti text editor kolaborasi, POC sistem konfigurasi Operator Management, POC View-Only Instrument Booking Calendar, serta POC document editor baru. Seluruh pekerjaan ini dilakukan secara iteratif dengan memperhatikan konsistensi arsitektur, keamanan akses berdasarkan peran, serta efisiensi pengelolaan data dalam lingkungan multi-tenant.

2. Kontribusi pada AI Chatbot Internal
   Pada proyek AI Chatbot Internal, kontribusi mencakup pengembangan UI dan integrasi fungsional sistem chatbot. Lingkup pekerjaan meliputi perbaikan dan penyempurnaan UI pada halaman percakapan, integrasi fitur manual prompt antara frontend dan backend, serta pengembangan halaman admin beserta Prompt Form Editor untuk mendukung konfigurasi internal. Pekerjaan dilakukan secara bertahap melalui perancangan UI, definisi skema data, pembuatan RPC dan struktur database, hingga integrasi dan validasi akhir untuk memastikan fungsi chatbot dapat dikonfigurasi dan dijalankan secara konsisten.

Penjabaran teknis mengenai kontribusi yang dilakukan disajikan pada subbab berikutnya secara terstruktur berdasarkan *timeline* dan modul pengembangan.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama magang

Minggu Ke-	Pekerjaan yang Dilakukan
1	28 Juli – 1 Agustus 2025 • Inisialisasi modul User & Roles Management dan struktur konfigurasi awal. • Implementasi *server-side pagination* dan column filtering dasar. • Integrasi awal kolaborasi *real-time* pada text editor.
2	4 Agustus – 8 Agustus 2025 • Melanjutkan implementasi inisialisasi modul User & Roles Management. • Perbaikan *scrollbar* bug pada chat page AI chatbot internal. • Perbaikan bug lanjutan pada integrasi text editor kolaborasi.
3	11 Agustus – 15 Agustus 2025 • Implementasi fitur manual prompt pada AI chatbot internal. • Perbaikan *scroll-down button* bug pada chat page AI chatbot internal. • Pengembangan dasar konfigurasi sistem POC Operator Management.
4	18 Agustus – 22 Agustus 2025 • Implementasi fitur pembuatan user baru pada User & Roles Management. • Penyelarasan UI konfigurasi untuk konsistensi modul. • Melanjutkan penyusunan konfigurasi sistem POC Operator Management.
	Lanjut pada halaman berikutnya

Tabel 3.1 Pekerjaan yang dilakukan tiap minggu selama magang (lanjutan)

Minggu Ke-	Pekerjaan yang Dilakukan
5	25 Agustus – 29 Agustus 2025 • Penyelesaian halaman pembuatan user baru. • Implementasi *role* filtering, optimasi pengambilan data, dan perbaikan performa *server actions*. • Pembersihan RPC, *trigger*, dan tabel yang tidak digunakan.
6	1 September – 5 September 2025 • Inisialisasi modul Admin & Tenant Management. • Penyesuaian code convention untuk konsistensi antar modul. • Pembuatan layout awal Admin Page pada AI chatbot internal.
7	8 September – 12 September 2025 • Penamaan ulang modul User & Roles serta Admin & Tenant Management sesuai fungsinya, *refactoring server actions*, serta implementasi fitur disabling user. • Penyusunan UI *draggable* serta interaktivitas dasar awal untuk Prompt Form Editor.
8	15 September – 19 September 2025 • Penyempurnaan UI *draggable* dan interaktivitas Prompt Form Editor. • Perancangan skema data Prompt Form yang bersifat dinamis. • Penyusunan types dan RPC awal untuk Prompt Form Editor. • Refactoring tipe data pada modul User & Roles Management.
9	22 September – 26 September 2025
	Lanjut pada halaman berikutnya

Tabel 3.1 Pekerjaan yang dilakukan tiap minggu selama magang (lanjutan)

Minggu Ke-	Pekerjaan yang Dilakukan
	• Implementasi fitur user editing dan *refactoring* form pada User & Roles serta Admin & Tenant Management. • Penyusunan DDL dan DML awal untuk data Prompt Form dinamis. • Eksplorasi awal tampilan kalender untuk *View-Only* Instrument Booking Calendar (POC).
10	29 September – 3 Oktober 2025 • Integrasi database dengan frontend Prompt Form Editor (pengambilan *section*, *field*, dan opsi dinamis). • Implementasi awal kalender *view-only* booking management (POC) serta integrasi dengan database. • Eksplorasi fitur dan opsi untuk POC document editor baru.
11	6 Oktober – 10 Oktober 2025 • Melanjutkan integrasi database dengan Prompt Form Editor. • Penyempurnaan UI Prompt Form Editor agar konsisten dengan struktur dinamis. • Implementasi dan setup lanjutan POC document editor.
12	13 Oktober – 17 Oktober 2025 • Penyesuaian lanjutan Prompt Form Editor dan konsolidasi struktur frontend-backend. • Eksplorasi dan perancangan struktur Formula List untuk manajemen formula.
13	20 Oktober – 24 Oktober 2025
	Lanjut pada halaman berikutnya

Tabel 3.1 Pekerjaan yang dilakukan tiap minggu selama magang (lanjutan)

Minggu Ke-	Pekerjaan yang Dilakukan
	• Perancangan dan implementasi mekanisme saving berbasis *tracking* perubahan pada Prompt Form Editor. • Penanganan bug dan penyesuaian perilaku saving. • Implementasi tampilan list formula per proyek.
14	27 Oktober – 31 Oktober 2025 • Integrasi Prompt Form Editor dengan Prompt Form dinamis pada AI chatbot internal. • Penyempurnaan UI dan perbaikan bug integrasi Prompt Form. • Refactoring fitur copy formula pada Formula List.
15	3 November – 7 November 2025 • Audit skema, optimasi struktur, dan konsolidasi Prompt Form Editor. • Perancangan *validation schema* berbasis konfigurasi field. • Perbaikan bug lanjutan pada Prompt Form Editor.
16	10 November – 14 November 2025 • Integrasi *validation schema* ke Prompt Form Editor dan Prompt Form dinamis. • Bug fixing generation chat pada Prompt Form baru. • Penyesuaian implementasi Formula List menjadi tampilan per product di Product Management.

3.3.1 Kontribusi pada ELN

Proyek ELN berada pada tahap pengembangan lanjutan dengan integrasi bertahap ke lingkungan *staging*. Setiap *backlog* ditetapkan oleh lead software engineer sesuai prioritas kebutuhan user yang terus berkembang. Pekerjaan pada proyek ini berfokus pada pengembangan modul-modul konfigurasi sistem

yang menjadi pusat pengelolaan berbagai entitas dan hanya dapat diakses oleh user dengan hak admin. Modul-modul tersebut mendukung operasi pembuatan, pembaruan, pengarsipan, dan penghapusan entitas yang diperlukan dalam pengelolaan data pada sistem ELN. Kontribusi utama meliputi pengembangan modul User & Roles Management, Admin & Tenant Management, serta Formula List. Selain itu, beberapa pekerjaan eksploratif berupa POC juga dilakukan, seperti integrasi awal text editor kolaboratif, modul Operator Management, implementasi *View-Only* Instrument Booking Calendar, dan integrasi document editor, meskipun tidak dilanjutkan ke tahap final. Setiap fitur yang telah diselesaikan melalui proses evaluasi oleh tim SWE dan PM sebelum digabungkan ke dalam lingkungan *staging* untuk pengujian lebih lanjut.

A User & Roles Management (Week 1–9)

Modul User & Roles Management merupakan bagian dari konfigurasi sistem yang memungkinkan admin untuk mengelola data user beserta *roles* masing-masing. Pengelolaan ini dibatasi pada tenant yang menjadi ruang lingkup otoritas admin. Modul ini dikembangkan dalam bentuk tabel data interaktif yang dilengkapi dengan *pagination* dan filtering kolom untuk memudahkan navigasi serta pencarian data user. Implementasi berfokus pada pembuatan struktur tabel, pengaturan *server actions*, serta integrasi API requests untuk mendukung pemrosesan data secara *server-side* agar sistem tetap efisien dan konsisten. Melalui modul ini, pengelolaan akun user di setiap tenant dapat dilakukan secara terpusat dan aman.

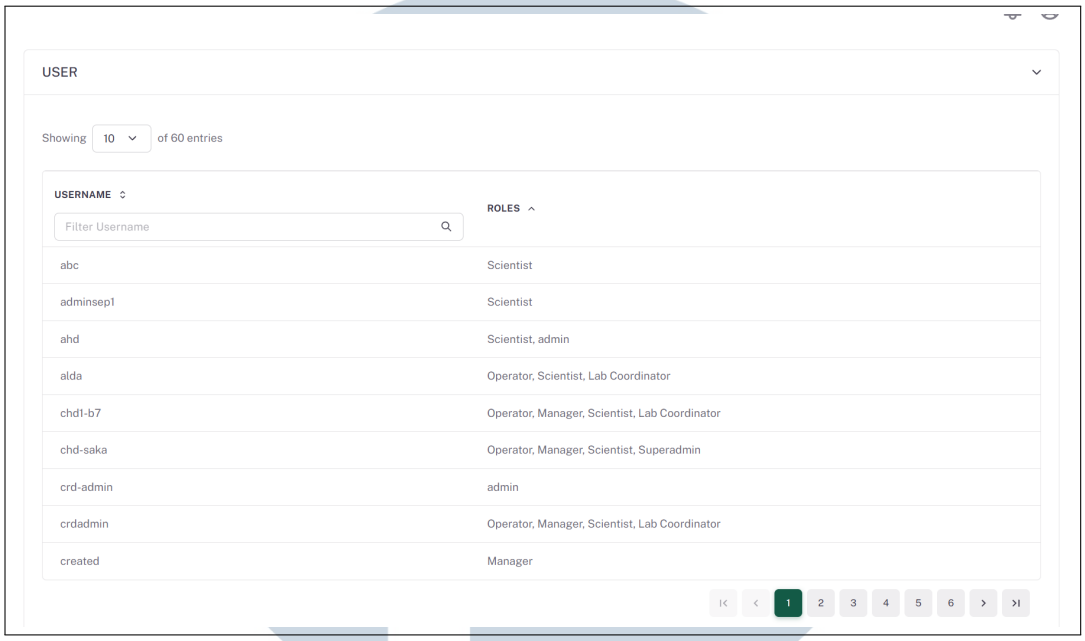
A.1 Kontribusi Berdasarkan Timeline

Berikut adalah kontribusi pada modul User & Roles Management berdasarkan timeline:

A.1.1 Week 1–2: Inisialisasi Modul

Tahap awal pengembangan difokuskan pada pembuatan tabel data interaktif dengan dua kolom utama, yaitu `username` dan `roles`. Nilai `roles` direpresentasikan sebagai *array* `string` yang ditampilkan sebagai *comma-separated list* untuk menjaga *readability*. Filtering kolom diterapkan pada `username` untuk mendukung pencarian *server-side*, sementara mekanisme *pagination* diterapkan untuk membatasi jumlah data yang ditampilkan setiap

halaman. Tampilan tabel tersebut ditunjukkan pada Gambar 3.1, dengan judul *USER* di atas tabel.



Gambar 3.1. Tabel User & Roles Management dengan kolom `username` dan `roles`

Seluruh proses pengambilan data mengikuti pola *server-side* dengan alur sebagai berikut:

1. Melakukan autentikasi ke *identity provider* eksternal.
2. Mengambil seluruh daftar user melalui *endpoint* API.
3. Melakukan pemanggilan tambahan untuk setiap user guna memperoleh informasi tenant dan *roles* (N+1 request).
4. Melakukan filtering *in-memory* berdasarkan hak akses tenant dan filter kolom.
5. Menghitung jumlah total data secara manual karena tidak tersedia *endpoint* bawaan.
6. Memotong data sesuai parameter halaman sebelum dikirim ke UI.

Alur pemrosesan ini ditunjukkan pada Kode 3.1.

```
const fetchUsersRoles = async (params: FetchParams): Promise<{
    rows: User[], totalCount: number }> => {
```

```
  const { pageIndex, pageSize, usernameFilter, currentAdminTenant
   } = params;

  // Fetch all users
  const users: User[] = await getAllUsers(usernameFilter);

  // Enrich users with roles and tenants
  for (const user of users) {
    user.roles = await getUserRoles(user.id);
    user.tenant = await getUserTenant(user.id);
  }

  // Apply tenant filter and get total count
  const filtered = users.filter(user => user.tenant ===
   currentAdminTenant);
  const totalCount = filtered.length;

  // Paginate
  const rows = filtered.slice(pageIndex * pageSize, (pageIndex +
   1) * pageSize);

  return { rows, totalCount };
}
```

Kode 3.1: Alur pengambilan data modul User & Roles Management

Role assignment juga diinisialisasi pada tahap ini, yang dapat dilakukan melalui dialog *Assign Role* yang menampilkan satu *field* `username` *read-only* dan *multiselect* untuk memilih `roles`, dengan tombol konfirmasi yang menjalankan *server action* dan memuat ulang tabel setelah perubahan tersimpan. Perubahan `roles` tidak dilakukan dengan menghapus dan menulis ulang seluruh daftar *roles*, namun dengan menghitung selisih antara *roles* lama dan *roles* baru sehingga hanya perubahan yang dikirimkan.

A.1.2 Week 4–5: Penambahan Role Filtering, Optimasi Mekanisme Fetching, Pembaruan UI, dan Fitur Pembuatan User Baru

Pada tahap ini dilakukan perluasan fitur filtering dengan menambahkan filter berbasis *role* pada kolom `roles`, sehingga admin dapat melakukan pencarian user tidak hanya berdasarkan `username` tetapi juga berdasarkan *role* yang dimiliki. Perubahan ini berdampak langsung pada mekanisme pengambilan data, karena *identity provider* tidak menyediakan *endpoint* bawaan untuk melakukan *pagination*

sekaligus filtering berdasarkan *role*. Akibatnya, logika *server actions* harus diperluas agar dapat menangani dua mode pengambilan data yang berbeda.

Mode pertama berlaku ketika hanya kolom `username` yang difilter; pada kasus ini masih dapat digunakan *endpoint* pencarian bawaan dengan *server-side pagination* yang efisien. Mode kedua berlaku ketika terdapat filter *role*; pada kondisi tersebut, pendekatan berubah menjadi *progressive batched fetching* untuk mengambil user dalam *batch* kecil secara bertahap, memperkaya (*enrich*) tiap entri dengan informasi *role* dan tenant, kemudian melakukan filtering *in-memory*. Pendekatan ini memungkinkan penghentian dini (*early termination*) segera setelah jumlah hasil yang dibutuhkan tercapai, sehingga mengurangi beban memori dan biaya permintaan API yang tidak relevan dibandingkan mengambil seluruh user sekaligus.

Implementasi *progressive batched fetching* menggunakan *offset-based pagination* dalam *batch* kecil. Pendekatan ini menghindari pemuatan seluruh dataset ke memori sekaligus dan memungkinkan penghentian dini segera setelah jumlah hasil yang dibutuhkan untuk halaman aktif terpenuhi. Dengan strategi ini, sistem hanya memroses data yang diperlukan tanpa melakukan *enrichment* dan filtering terhadap seluruh dataset. *Progressive batched fetching* ini diperjelas pada Kode 3.2.

```
async function progressiveBasedFetch(
  filters: FilterParams,
  pageIndex: number,
  pageSize: number
): Promise<{ users: User[]; totalCount: number }> {
  const results: User[] = [];
  let offset = 0;
  const batchSize = 50;
  const targetEnd = (pageIndex + 1) * pageSize;

  while (results.length < targetEnd) {
    const batch = await fetchUsers(offset, batchSize);

    if (batch.length === 0) break;

    const enriched = await enrichWithRolesAndTenants(batch);
    const filtered = applyFilters(enriched, filters);

    results.push(...filtered);
    offset += batchSize;
```

```
21
22      if (batch.length < batchSize) break;
23    }
24
25    const paginatedUsers = results.slice(
26      pageIndex * pageSize,
27      targetEnd
28    );
29
30    return { users: paginatedUsers, totalCount: results.length };
31  }
```

Kode 3.2: *Progressive batched fetching* dengan *early termination*

Dari sisi UI, kolom `roles` diperbarui agar tidak hanya menampilkan teks berisi daftar *role*, tetapi menampilkan komponen *expandable chips* dengan dua *chip* pertama terlihat dan sisanya tersembunyi. Selain meningkatkan keterbacaan, pola penyajian ini juga konsisten dengan komponen tabel pada modul konfigurasi lain. Dilakukan juga pembaruan *layout* modul konfigurasi, di mana struktur sebelumnya menggunakan komponen *accordion* sehingga tabel hanya terlihat ketika dibuka karena mengikuti UI preseden lama, tetapi *accordion* tersebut tidak memberikan tujuan yang jelas. Komponen tersebut diganti menjadi tampilan statis dengan judul dan kontainer tabel agar UI lebih konsisten dengan modul lain, termasuk Operator Management yang masih dalam tahap awal pengembangan. Tampilan tabel setelah pembaruan ini ditunjukkan pada Gambar 3.2.

Selain fitur filtering, ditambahkan pula fitur pembuatan user baru langsung dari modul konfigurasi tanpa perlu menggunakan platform eksternal. Fitur ini tersedia melalui tombol *Add User* dan membuka dialog berisi *field* seperti `username`, `email`, `password`, serta *role* yang ingin ditugaskan. Selama proses submit, *server action* membuat user baru di *identity provider*, menetapkan user ke tenant milik admin, lalu menambahkan *role* yang dipilih. Dialog pembuatan user juga memberi preview tenant user akan ditempatkan melalui sebuah disabled *field*, dan melakukan validasi dasar seperti keunikan `username` serta keberadaan minimal satu *role* sebelum submit.

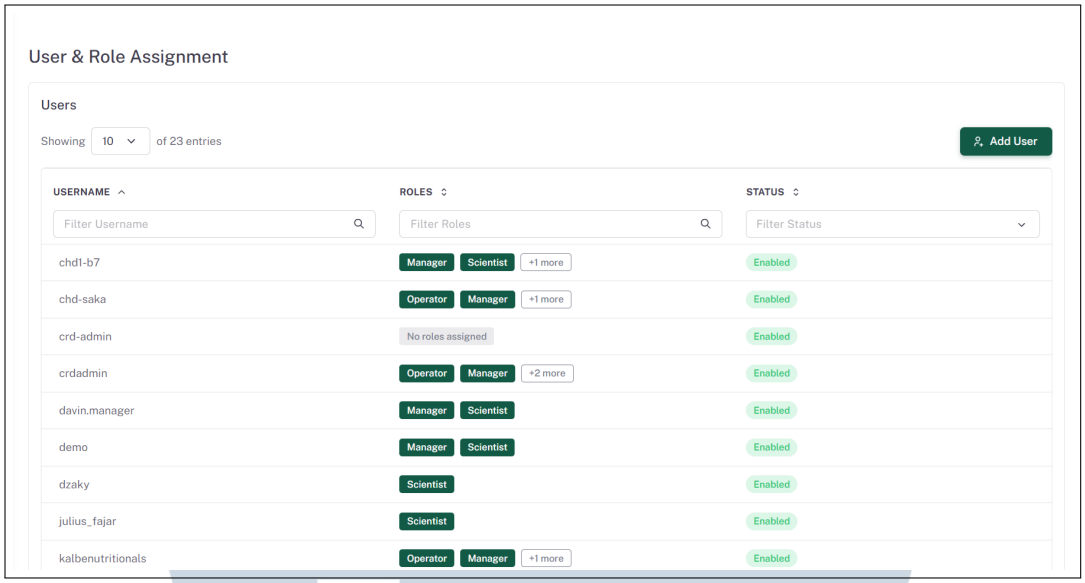
A.1.3 Week 7–8: Penyesuaian Code Convention, Penamaan Ulang Modul, Refactoring Server Action dan Type, serta Penambahan Fitur Disabling User

Pada tahap ini dilakukan penyesuaian menyeluruh terhadap struktur kode dan penamaan modul agar modul konfigurasi lebih konsisten dengan pola yang telah digunakan pada konfigurasi sistem lain. Modul *Role Assignment* diubah menjadi *User & Roles Management* karena fungsinya tidak lagi terbatas pada penetapan *role*, melainkan mencakup pengelolaan user secara penuh. Demikian pula, modul *Tenant Assignment* diubah menjadi *Admin & Tenant Management* karena modul ini telah berkembang menjadi modul manajemen tenant dan admin, bukan hanya penugasan tenant.

Penyesuaian code *convention* dilakukan pada beberapa aspek utama. Pertama, struktur *server actions* dipisahkan ke dalam *directory* terdedikasi agar setiap fungsi memiliki tanggung jawab tunggal dan dapat digunakan lintas modul apabila diperlukan. Kedua, komponen form yang lama diganti dengan komponen baru seperti untuk *field* yang mendukung aturan validasi terpusat, termasuk integrasi dengan *hook* `useFormDialog` untuk penanganan dialog yang lebih terstruktur. Ketiga, dilakukan penyelarasan penamaan, style UI, dan struktur *directory* agar konsisten dengan modul konfigurasi lain seperti Material dan Product Management.

Selain penataan struktur kode, dilakukan refactoring tipe data pada modul User & Roles Management. Sebelumnya, tipe user yang digunakan merupakan tipe buatan sendiri. Refactoring dilakukan dengan mengganti tipe tersebut menggunakan tipe representasi user bawaan *identity provider* agar lebih selaras dengan struktur respons API dan menyediakan atribut tambahan seperti `enabled`, yang diperlukan untuk fitur disabling user. Refactoring ini mencakup pembaruan tipe pada komponen, fungsi *server actions*, serta penghapusan definisi tipe lama yang sudah tidak relevan.

Pada tahap ini juga ditambahkan fitur disable user untuk memungkinkan admin menonaktifkan akun user tanpa perlu menghapus data. Fitur ini disertai penambahan kolom `status` pada tabel agar informasi `enabled` atau `disabled` terlihat langsung di UI. Aksi disabling dilakukan melalui *server action* khusus yang mengubah status `enabled` dan kemudian me-reload tabel. User yang berstatus disabled juga tidak dapat diedit melalui dialog form untuk menjaga integritas data. Tampilan tabel setelah penambahan kolom `status` dan tindakan disabling ditunjukkan pada Gambar 3.2 berikut.



Gambar 3.2. Tabel User Management dengan kolom `username`, `roles` berbentuk *chips*, dan `status`, masing-masing kolom dengan filter

A.1.4 Week 9: User Editing dan Refactoring Form (Bersama Admin & Tenant Management)

Fitur pengeditan user ditambahkan agar admin dapat memperbarui informasi user langsung melalui modul konfigurasi tanpa perlu menggunakan platform eksternal. Implementasi dilakukan secara seragam pada modul User & Roles Management dan Admin & Tenant Management sehingga kedua modul memiliki perilaku UI dan alur pemrosesan data yang konsisten. Fitur ini melengkapi siklus penuh manajemen user, yakni pembuatan akun, penyesuaian *roles*, disabling akun, dan kini pembaruan atribut user.

Alur pengeditan dilakukan melalui dialog form yang ditampilkan setelah admin memilih satu baris user pada tabel dan menekan tombol *Edit User*. Data awal form diisi menggunakan informasi user yang sudah ada, sementara *field* `password` bersifat opsional sehingga admin dapat memperbarui data profil tanpa harus mengubah kata sandi. Setelah dikirim, perubahan diteruskan ke *server action* khusus yang hanya mengirim atribut yang dimodifikasi ke *identity provider* sehingga tidak seluruh data user ditimpa kembali.

Agar form dapat digunakan baik untuk operasi pembuatan maupun pengeditan, dialog yang sebelumnya dipisahkan menjadi dua komponen digabung menjadi satu komponen `UserForm` dengan parameter `mode` (`add` atau `edit`). Pendekatan ini menghilangkan duplikasi logika validasi dan mengurangi

ketergantungan antar komponen. Struktur yang sudah di-*refactor* ini dapat dilihat pada Kode 3.3 berikut:

```
// BEFORE: Separate user forms
<AddUserForm />
<EditUserForm />

// AFTER: Unified user forms with mode prop
const UserForm = ({ mode, initialData }: UserFormProps) => {
  const schema = mode === 'add' ? addUserSchema : editUserSchema;

  const handleSubmit = async (data: UserFormData) => {
    if (mode === 'add') {
      await createUser(data);
    } else {
      await editUserDetails({ ...data, userId: initialData.id });
    }
  };

  return (
    // Form UI
  );
}
```

Kode 3.3: Struktur *unified form component* dengan mode `add` dan `edit`

A.2 Outcome

Modul telah dinilai stabil dan konsisten dengan standar konvensi konfigurasi sistem, dengan fungsi yang mencakup pengelolaan user dan *roles* secara terpusat. Modul telah diintegrasikan ke lingkungan *staging* dan siap mendukung pengelolaan user dan *roles* dalam ELN.

B Admin & Tenant Management (Week 6–9)

Modul Admin & Tenant Management dirancang untuk mendukung fungsi admin tingkat tinggi yang hanya dapat diakses oleh superadmin, yakni peran dengan hak akses penuh terhadap seluruh tenant. Modul ini memungkinkan pengelolaan tenant dan admin dari semua tenant. Sebagaimana modul konfigurasi sistem lainnya, UI dikembangkan dalam bentuk tabel data dengan fitur pagination dan filtering kolom. Implementasi dilakukan dengan membangun mekanisme *server*

actions dan API requests yang mendukung operasi *create*, *update*, dan disable baik untuk akun admin maupun entitas tenant.

B.1 Kontribusi Berdasarkan Timeline

Berikut adalah kontribusi pada modul Admin & Tenant Management berdasarkan timeline:

B.1.1 Week 6: Inisialisasi Modul

Tahap awal pengembangan modul Admin & Tenant Management difokuskan pada pembuatan tabel data untuk menampilkan daftar admin beserta tenant yang mereka kelola. Dua kolom utama yang ditampilkan adalah `username` dan `tenant`, dilengkapi dengan mekanisme filtering kolom dan pagination. Seluruh proses pengambilan data dilakukan melalui *server actions* karena API yang tersedia tidak menyediakan *endpoint pagination* ataupun filtering terstruktur untuk kebutuhan ini. Tampilan tabel ini secara final dapat dilihat pada Gambar 3.3.

Alur pengambilan data dilakukan secara *server-side* melalui fungsi pemrosesan yang menerima parameter `pageIndex`, `pageSize`, dan filter kolom. Fungsi tersebut mengambil daftar admin, memetakan tenant yang terkait, menerapkan filter berdasarkan *username* dan tenant, menghitung total data yang memenuhi kriteria, kemudian memotong *array* hasil filtering sesuai *offset* dan `pageSize`. Alur ini ditunjukkan pada Kode 3.4 berikut.

```
1  const fetchAdminsTenants = async (params: FetchParams): Promise<{
       rows: Admin[]; totalCount: number }> => {
2    const { pageIndex, pageSize, usernameFilter, currentAdminTenant
       } = params;
3
4    const admins = await fetchAllAdmins(usernameFilter);
5
6    for (const admin of admins) {
7      admin.tenants = await fetchTenants(admin);
8    }
9
10   const filtered = admins.filter(admin => admin.tenants.includes(
       currentAdminTenant));
11   const totalCount = filtered.length;
12
```

```
13    const rows = filtered.slice(pageIndex * pageSize, (pageIndex +
        1) * pageSize);
14
15    return { rows, totalCount };
16  }
```

Kode 3.4: Alur pemrosesan *server-side* untuk modul Admin & Tenant

Pada tahap ini juga diimplementasikan fitur pembuatan admin user baru melalui dialog yang dapat diakses dari tombol *Add Admin*. Dialog ini memuat *field* untuk `username`, `email`, `password`, serta pemilihan tenant yang akan di-*assign* kepada admin tersebut. Validasi dasar diterapkan untuk memastikan keunikan `username` dan kelengkapan data sebelum submit.

Selain itu, fitur pembuatan tenant baru juga diimplementasikan pada tahap ini melalui dialog terpisah yang dapat diakses dari tombol `Add Tenant`. Dialog ini hanya berisi *field* untuk `tenant name`. Setelah tenant berhasil dibuat, tenant tersebut dapat langsung di-*assign* ke admin melalui dialog pengeditan admin atau dialog pembuatan admin baru.

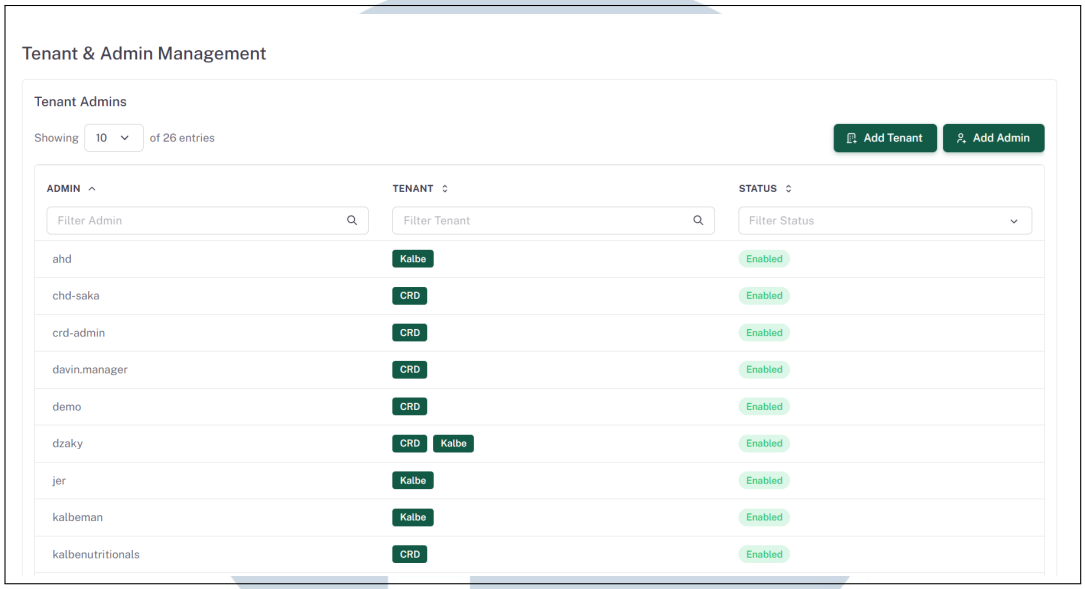
B.1.2 Week 7–8: Penyesuaian Code Convention, Penamaan Ulang Modul, Refactoring Server Action dan Type, Disabling User (Bersama User & Roles Management)

Penamaan ulang modul dilakukan dari *Tenant Assignment* menjadi *Admin & Tenant Management* untuk mencerminkan ruang lingkup yang meluas, yaitu dari sekadar penugasan tenant menjadi pengelolaan entitas admin dan tenant secara penuh. Pada tahap ini dilakukan penyesuaian konvensi kode, pemindahan ulang *server actions* ke dalam *directory* terpisah berdasarkan tanggung jawab, serta penyelarasan struktur direktori agar identik dengan pola yang telah diterapkan pada modul User & Roles Management.

Refactoring tipe juga dilakukan untuk menghapus tipe user buatan sebelumnya dan menggantinya dengan `StandardizedUser` standar yang sudah mendukung atribut `enabled`. Perubahan ini diperlukan agar fungsi disabling user dapat diterapkan secara seragam pada kedua modul.

Fitur disable user ditambahkan dengan pola yang sama seperti pada modul User & Roles Management: status user ditentukan oleh atribut `enabled`, sebuah kolom `status` ditampilkan pada tabel, dan baris user yang berada pada keadaan nonaktif dibatasi dari operasi pengeditan. Dengan pendekatan ini, kedua modul berbagi sistem perilaku yang konsisten tanpa duplikasi implementasi. Tampilan

tabel Admin & Tenant Management setelah penamaan ulang dan penambahan kolom `status` beserta fitur disable/enable user ditunjukkan pada Gambar 3.3.



Gambar 3.3. Tabel Admin & Tenant Management dengan kolom `username`, `tenants`, dan `status` serta menu aksi Disable/Enable

B.1.3 Week 9: User Editing & Refactoring Form (bersama User & Roles Management)

Fitur pengeditan admin ditambahkan untuk memungkinkan pembaruan data user dan tenant secara langsung melalui tabel konfigurasi. Implementasinya mengikuti pola umum yang telah digunakan pada modul User & Roles Management, yaitu penggunaan komponen form tunggal dengan mode `add` dan `edit` yang diatur melalui properti `mode`. Pendekatan ini menghilangkan kebutuhan untuk memelihara dua komponen terpisah dan memastikan validasi, struktur *field*, dan perilaku dialog tetap konsisten.

Refactoring dilakukan terhadap keseluruhan dialog form, termasuk pemanfaatan `useFormDialog` untuk pengendalian state dan konfirmasi, serta penggunaan komponen *field* yang telah diperbarui untuk mendukung validasi skema dan pesan kesalahan terpusat. Perbedaan utama dibandingkan modul User & Roles Management terletak pada kebutuhan pemilihan tenant. Pada mode pengeditan, nilai tenant awal dipetakan kembali ke pilihan *dropdown* dan selalu ditampilkan secara sinkron dengan data user yang sedang diedit.

Selain itu, seperti di User & Roles Management, setelah melakukan seleksi

admin, pengeditan informasi admin dapat dilakukan dengan mengeklik tombol *Edit Admin* untuk membuka dialog form dengan *field* `username` (*read-only*), `email`, `name`, `password` opsional, dan `tenant` *select*.

Dengan perubahan ini, kedua modul kini berbagi pola form yang identik, dapat dipelihara secara paralel, dan memungkinkan penambahan fitur lanjutan seperti perubahan `tenant`, perubahan password opsional, serta penyelarasan tombol aksi tanpa memengaruhi konsistensi UI.

B.2 Outcome

Modul telah dinilai stabil dan konsisten dengan standar konvensi konfigurasi sistem, dengan fungsi yang mencakup pengelolaan admin dan tenant secara terpusat. Modul telah diintegrasikan ke lingkungan *staging* dan siap mendukung pengelolaan admin dan tenant dalam ELN.

C Formula List (Week 12–16)

Fitur Formula List merupakan komponen konfigurasi sistem yang dirancang untuk memudahkan user dalam meninjau seluruh formula yang terkait dengan suatu produk tertentu. Dalam konteks logika bisnis, formula dapat dipahami sebagai variasi atau susunan tahapan pembuatan suatu produk beserta material yang digunakan pada setiap tahapnya. Secara struktural, sebuah formula berada dalam suatu eksperimen, yang kemudian berada dalam sebuah proyek, yang kemudian berada dalam sebuah produk. Pada tahap awal, kebutuhan user mengarah pada penyajian daftar formula dalam cakupan proyek. Selanjutnya, terjadi penyesuaian kebutuhan lanjutan yang mengarahkan pengembangan pada tampilan daftar formula per produk untuk memberikan akses cepat terhadap seluruh formula yang relevan dalam satu tempat. Penyesuaian berlapis ini menjadi dasar pengembangan Formula List sehingga fitur yang dihasilkan selaras dengan kebutuhan user dan mendukung efisiensi proses peninjauan formula.

C.1 Kontribusi Berdasarkan Timeline

Berikut adalah kontribusi pada Formula List berdasarkan timeline:

C.1.1 Week 12: Eksplorasi dan Definisi Struktur

Tahap ini difokuskan pada eksplorasi kebutuhan bisnis dan penentuan struktur tampilan untuk fitur Formula List. Proses dimulai melalui diskusi untuk memastikan alur data yang harus ditampilkan: produk → proyek → eksperimen → formula → tahap → material. Tahap ini juga berfungsi sebagai fase pembelajaran domain, karena pemahaman struktur hierarki tersebut menjadi syarat utama sebelum merancang tampilan dan pola interaksi.

Dari sisi kebutuhan UI, daftar formula tidak ditampilkan sebagai halaman tersendiri, melainkan muncul sebagai dialog ketika user memilih sebuah proyek pada halaman daftar proyek. Dialog ini harus menampilkan seluruh formula dalam ruang lingkup proyek tersebut tanpa kehilangan konteks asalnya. Berdasarkan hasil evaluasi beberapa alternatif tampilan (tabel *nested*, *tab-based navigation*, baris *expandable*), struktur *nested accordion* dipilih sebagai solusi paling sesuai karena mampu menampilkan hierarki multi-level tanpa membebani ruang tampilan utama. Struktur akhir yang disepakati adalah sebagai berikut: lapisan pertama adalah daftar eksperimen dalam proyek, setiap eksperimen menampilkan daftar formula, setiap formula berisi daftar tahapan, dan setiap tahap menampilkan tabel material yang digunakan.

C.1.2 Week 13: Implementasi Tampilan List Formula Per Proyek

Struktur *nested accordion* yang telah dikonfirmasi pada tahap sebelumnya mulai diimplementasikan pada minggu ini. Implementasi mencakup pembuatan *remote procedure call* (RPC) baru untuk mengambil data formula dalam lingkup proyek secara hierarkis, penyusunan *server action* dengan pemetaan tipe data berlapis, serta pembangunan dialog UI yang menampilkan data tersebut dalam bentuk *nested accordion*. Seluruh proses ditujukan agar struktur bisnis formula dapat ditampilkan secara lengkap dalam satu tampilan tanpa perlu membuka halaman detail eksperimen.

Pada lapisan database, ditambahkan RPC `get_project_formula_list` yang menghasilkan sebuah struktur `JSONB` berisi daftar eksperimen dan seluruh formula beserta tahap serta materialnya. Untuk menghindari masalah N+1 query, seluruh struktur dihimpun dalam satu eksekusi fungsi tersebut. Pseudocode struktur logika pengambilan data ditunjukkan pada Kode 3.5.

```sql
CREATE OR REPLACE FUNCTION get_project_formula_list(p_project_id
    UUID)
RETURNS JSONB AS $$
BEGIN
  RETURN (
    -- Build project object
    SELECT jsonb_build_object(
      'project_id', pr.project_id,
      'experiments', (
        -- Build experiments array
        SELECT jsonb_agg(exp_data ORDER BY e.created_at DESC)
        FROM (
          SELECT jsonb_build_object(
            'experiment_id', e.experiment_id,
            'formulas', (
              -- Build formulas array, then stages, then materials
    ...
            )
          ) AS exp_data
          FROM experiment e
          WHERE e.project_id = pr.project_id
        ) experiments
      )
    )
    FROM project pr
    WHERE pr.project_id = p_project_id
  );
END;
$$ LANGUAGE plpgsql;
```

Kode 3.5: RPC `get_project_formula_list` untuk struktur hierarkis

Pada halaman daftar proyek, setelah user memilih satu baris proyek, muncul tombol aksi `View Formula List` yang ketika ditekan membuka dialog tersebut. Dialog Formula List yang muncul menampilkan struktur *nested accordion* yang sudah disepakati sebelumnya: eksperimen sebagai *accordion* tingkat pertama, formula pada tingkat kedua, tahap pada tingkat ketiga, dan tabel material pada bagian terdalam. Struktur tabel dipilih untuk material karena data material memiliki kolom numerik, unit, status stok, serta informasi batch yang lebih mudah dipindai dalam bentuk tabel. Dialog ini secara final ditunjukkan pada Gambar 3.4.

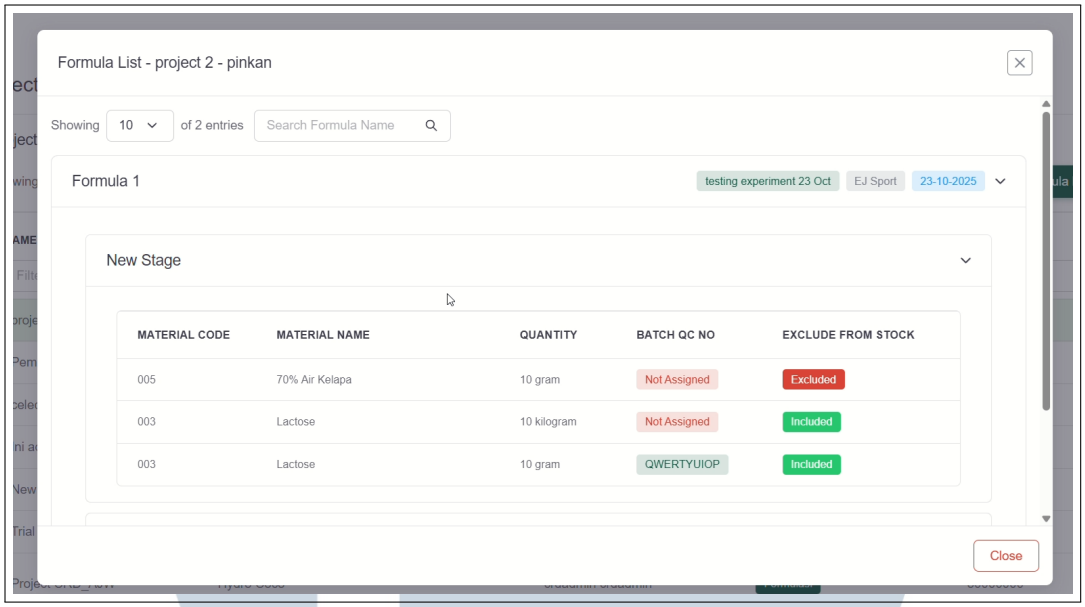
C.1.3 Week 14: Refactoring Fitur Copy Formula

Fitur *Copy Formula* pada modul lain dalam sistem memiliki struktur tampilan dan hierarki data yang serupa dengan Formula List, yaitu menampilkan formula beserta tahap dan material dalam susunan *nested accordion*. Perbedaannya terletak pada kemampuan pemilihan formula sebagai *template*, serta adanya *pagination* dan filtering untuk membantu memilih formula dalam jumlah besar. Karena terdapat tumpang tindih yang signifikan dalam struktur UI serta pemetaan data, dilakukan *refactoring* menyeluruh agar logika tampilan, pengelolaan *state accordion*, dan mekanisme pengambilan data dapat digunakan kembali secara konsisten oleh kedua fitur.

Proses *refactoring* dilakukan dengan mengekstraksi seluruh logika utama menjadi satu komponen dasar `BaseFormulaDialog` yang dapat dikonfigurasi, sementara kedua dialog hanya berperan sebagai *wrapper* yang menentukan mode dan sumber data. Dengan pendekatan ini, komponen Formula List dalam konteks proyek menggunakan mode tampilan *read-only*, sedangkan fitur *Copy Formula* tetap mendukung pemilihan dan aksi lanjutan. Secara keseluruhan, `BaseFormulaDialog` tersebut mengelola hal-hal berikut secara terpusat:

1. State ekspansi *accordion* pada tingkat formula dan tahap.
2. State pemilihan formula (hanya aktif pada mode *copy*).
3. Pagination (`pageIndex`, `pageSize`, `totalCount`).
4. Filtering berdasarkan nama formula.

Dengan abstraksi ini, baik komponen Formula List per proyek maupun *Copy Formula* hanya melakukan konfigurasi cukup dengan menggunakan komponen `BaseFormulaListDialog` dan menyesuaikan melalui *props*. *Refactoring* ini menurunkan ukuran kode pada kedua dialog secara signifikan, menghilangkan duplikasi logika, memastikan konsistensi UI serupa antar fitur, serta membawa fitur *pagination* dan filtering ke Formula List per proyek. Selain itu, komponen hasil *refactoring* tetap kompatibel dengan struktur respons RPC sebelumnya sehingga tidak memerlukan perubahan pada sisi database. Tampilan dialog setelah *refactoring* ditunjukkan pada Gambar 3.4, tetap dengan struktur *nested accordions* namun sekarang dengan tambahan filtering dan *pagination*.



Gambar 3.4. Dialog Formula List berbentuk *nested accordions* setelah refactoring dengan *pagination* dan search *field*

C.1.4 Week 16: Revisi Akhir ke Tampilan Per Produk di Konfigurasi Sistem Product Management

Perubahan kebutuhan user menyebabkan daftar formula tidak lagi ditampilkan per proyek, melainkan langsung pada konteks produk untuk mempermudah akses informasi tanpa harus membuka halaman eksperimen atau proyek terlebih dahulu. Oleh karena itu, integrasi Formula List dipindahkan ke modul Product Management dan diimplementasikan sebagai halaman terpisah, bukan dialog. Dengan pendekatan ini, user dapat memilih satu baris produk dan memilih aksi View Formula List, kemudian diarahkan ke halaman dengan breadcrumb yang menunjukkan konteks navigasi konfigurasi sistem.

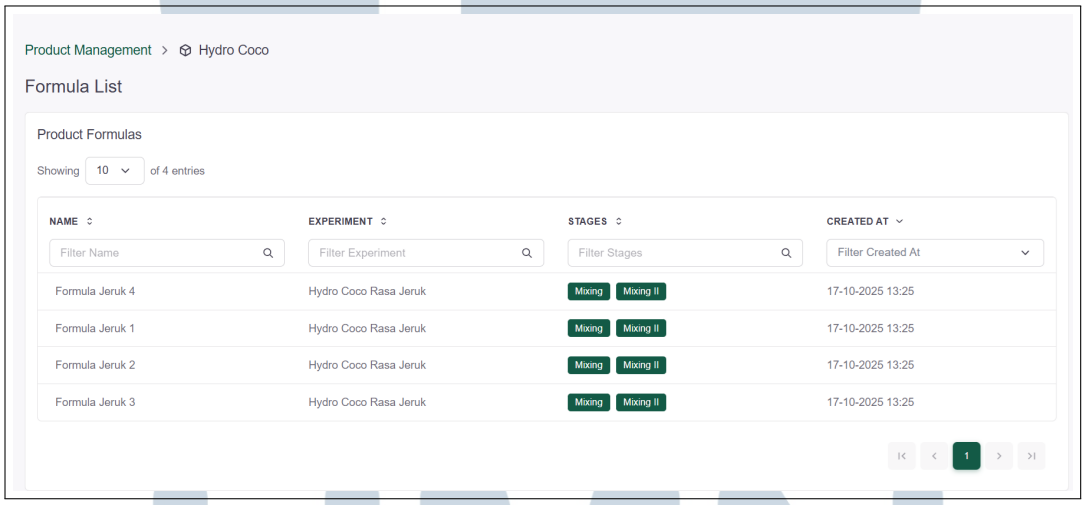
Tampilan baru menggunakan tabel data dengan server-side pagination, filtering, dan sorting berdasarkan nama formula, nama eksperimen, serta tanggal pembuatan. Seluruh data diperoleh melalui RPC yang telah tersedia sebelumnya, sehingga tidak diperlukan perubahan struktur basis data. Struktur halaman dirancang sebagai berikut:

1. Aksi View Formula List pada modul Product Management.
2. Routing menuju halaman `/product-management/[product-id]/formula`.

3. Penampilan tabel daftar formula disertai kolom nama formula, eksperimen, dan `created_at`.

4. Filtering per kolom dan pagination berbasis server.

Pada halaman Product Management, setelah user memilih satu baris produk, muncul tombol aksi `View Formula List`. Setelah tombol ditekan, user diarahkan ke halaman Formula List dengan *breadcrumb* yang menunjukkan navigasi dari Product Management ke produk spesifik. Halaman menampilkan tabel formula dengan *pagination* dan filtering. Tampilan halaman ini ditunjukkan pada Gambar 3.5 berikut.



Gambar 3.5. Halaman Formula List dengan *breadcrumbs*, tabel formula, pagination, dan filter

Tahap berikutnya adalah untuk menampilkan rincian tahap dan material dalam bentuk grid setelah user memilih suatu baris formula pada tabel. Bagian ini belum diimplementasikan pada saat penulisan ini, namun struktur data penelitian telah disiapkan sehingga integrasi lanjutan dapat dilakukan tanpa perubahan signifikan pada data flow di sisi server maupun *client*.

Pendekatan ini menutup pengembangan Formula List pada konteks proyek, menyatukan sumber informasi ke dalam satu modul konfigurasi sistem, serta menyederhanakan proses pemeriksaan hubungan antara produk dan seluruh formula yang memanfaatkannya.

C.2 Outcome

Fitur ini masih berada dalam tahap pengembangan lanjutan, namun struktur kebutuhan dan cakupan fungsional telah dikonfirmasi melalui evaluasi bersama user. Perubahan iteratif yang terjadi merupakan hasil penyesuaian kebutuhan yang masih dalam batas wajar pengembangan fitur.

D POC/Fitur yang Dihentikan atau Ditahan

Selain pengembangan fitur utama, terdapat beberapa pekerjaan eksploratif berupa POC dan pengembangan modul awal yang tidak dilanjutkan hingga tahap final. Seluruh POC ini dikembangkan sebagai bagian dari proses evaluasi kebutuhan sistem, baik untuk menguji kelayakan solusi teknis maupun untuk menilai kesesuaian dengan kebutuhan user. Penghentian masing-masing POC umumnya dipengaruhi perubahan prioritas pengembangan akibat adanya penyesuaian kebutuhan user serta ditemukannya redundansi fitur dengan modul yang telah ada. Meskipun tidak dirilis sebagai fitur akhir, rangkaian pekerjaan ini memberikan kontribusi penting dalam memahami batasan teknis, arah pengembangan modul di tahap selanjutnya, serta keputusan sistematis terkait fitur yang layak dilanjutkan.

Terkecuali untuk POC Document Editor (Week 10–11), dokumentasi visual untuk fitur-fitur POC berikut tidak dapat disertakan karena seluruh implementasi telah mengalami *breaking changes* dan tidak lagi dapat diakses pada lingkungan pengembangan saat penulisan dokumentasi dilakukan.

D.1 Text Editor Kolaborasi (Week 1–2)

Text Editor Kolaborasi merupakan POC editor teks *real-time* yang telah dikembangkan sebelumnya sebagai modul terpisah. Pada tahap ini, fokus pekerjaan bukan pada pembangunan editor dari awal, melainkan pada proses integrasi editor tersebut ke dalam sistem ELN dan penyelarasan mekanisme sinkronisasi agar sesuai dengan struktur data yang telah digunakan pada platform.

Pada Week 1 dilakukan penyesuaian skema integrasi, termasuk format penamaan dokumen kolaboratif dan pemisahan ruang dokumen per bagian laporan. Setiap instansi editor diidentifikasi dengan pola `<section>:<entityId>`.

Alur integrasi kolaborasi dirancang sebagai berikut:

1. Inisialisasi client text editor.
2. Inisialisasi objek `collabDoc` untuk menyimpan state dokumen yang akan disinkronisasi.
3. Penghubungan dengan server kolaborasi (WebSocket server) untuk sinkronisasi dokumen melalui objek `collabDoc` dan nama `<section>:<entityId>`.
4. Pengkaitan editor dengan objek `collabDoc` untuk sinkronisasi konten.
5. Pelacakan dan sinkronisasi *state* kehadiran dan posisi user ke `collabDoc` melalui editor.
6. Penyimpanan *state* dokumen secara berkala ke server kolaborasi setiap terjadi pembaruan di editor.

Pada Week 2 ditemukan beberapa kendala teknis yang muncul setelah proses penyatuan dengan backend ELN, antara lain:

1. Mekanisme penyimpanan pada server kolaborasi belum disesuaikan untuk identifikasi dengan pola baru (dipisah per bagian laporan), sehingga persistensi di database belum terjalin.
2. Objek sinkronisasi dan koneksi *real-time* kembali dibuat setiap kali komponen di-*render* ulang, menyebabkan kebocoran koneksi, konsumsi memori berlebih, dan gagal terjadinya sinkronisasi *multi-user*.

Permasalahan tersebut diselesaikan dengan penyesuaian kode di server kolaborasi untuk penyimpanan dan penambahan mekanisme memoisasi objek dokumen dan provider kolaborasi. Contoh penyelarasan penyimpanan ditunjukkan berikut:

Sebelum pengerjaan dihentikan, editor kolaborasi telah berfungsi stabil, termasuk tampilan user aktif yang sedang mengedit secara bersamaan. Pekerjaan dihentikan karena modul-modul lanjutan yang direncanakan menggunakan editor kolaboratif mengalami penjadwalan ulang. Dengan terselesaikannya perbaikan integrasi, penyelarasan format penyimpanan, dan sinkronisasi state, POC dinyatakan siap dipakai kembali ketika fase pengembangan berikutnya dimulai.

D.2 POC Operator Management (Week 3–4)

Operator Management dirancang sebagai modul konfigurasi sistem yang menampilkan daftar operator beserta eksperimen yang pernah atau sedang ditugaskan kepada masing-masing operator dalam bentuk tabel data, seperti konfigurasi sistem lainnya, yang memiliki kolom `operator_name`, yang menampilkan nama user operator, serta kolom `assigned_experiments`, yang menampilkan daftar eksperimen yang ditugaskan kepada operator dalam bentuk *array* `string` terpisah koma. Tujuan awal pengembangan adalah menyediakan tampilan terpusat yang berorientasi pada operator (*operator-centric view*). Operator Management ini juga hanya bersifat *view-only*.

Alur pengambilan data mengikuti pola *server-side* sebagai berikut:

1. Mengambil seluruh daftar operator dengan eksperimen yang ditugaskannya melalui *server action* yang memanggil dan mengolah RPC `fetch_operators_with_experiments`.
2. Melakukan filtering berdasarkan filter kolom, jika ada.
3. Menghitung jumlah total data.
4. Memotong data sesuai parameter tabel sebelum dikirim ke UI.

UI dirancang menggunakan tabel *server-side* dengan filtering pada kolom `operator_name`. Karena sifat modul ini hanya menampilkan informasi, tidak terdapat aksi penugasan ulang maupun penghapusan.

Setelah melalui evaluasi fungsional, modul dinilai mengalami redundansi. Informasi yang ditampilkan identik dengan data yang sudah tersedia pada modul eksperimen, hanya berbeda sudut pandang (*operator-centric* vs. *experiment-centric*). Selain itu, seluruh mekanisme penugasan operator terhadap eksperimen telah sepenuhnya dikelola oleh modul eksperimen sehingga modul ini hanya bersifat tampilan pasif tanpa nilai tambah operasional. Dengan pertimbangan tersebut, Operator Management dihentikan pada tahap POC dan tidak diteruskan ke tahap final.

D.3 POC View-Only Instrument Booking Calendar (Week 9–10)

POC ini dikembangkan untuk menyediakan tampilan kalender terpusat yang menampilkan jadwal penggunaan instrumen penelitian secara visual. Sebelum POC

ini dibuat, informasi jadwal hanya tersedia dalam bentuk daftar tabel sehingga menyulitkan user untuk memahami keterpakaian instrumen dan mendeteksi konflik waktu.

Struktur data dirancang agar setiap entri pemakaian instrumen memuat rentang waktu, identitas instrumen, serta konteks eksperimen dan tahapnya, menjadi sebuah *event* kalender. Transformasi data dilakukan di sisi server melalui RPC dengan mengambil daftar *booking* dan mengubahnya menjadi struktur umum kalender dengan kolom:

1. `start_time` dan `end_time` sebagai penanda rentang reservasi.
2. `instrument` sebagai objek ringkas berisi informasi instrumen.
3. `experiment_name`, `stage_name`, dan `procedure_name` sebagai informasi tambahan yang ditampilkan ketika *event* diklik.

Komponen kalender dibangun menggunakan *library* kalender *open-source* yang mendukung tampilan bulanan dan mingguan, navigasi tanggal, serta interaksi klik pada *event*, lalu di-*styling* menyesuaikan *template* dan *style guideline* ELN. Karena modul ini hanya bersifat *view-only*, seluruh komponen interaktif seperti pembuatan *event* baru, *drag-and-drop*, ataupun perubahan jadwal dinonaktifkan. Kalender dibuat menjadi komponen *reusable* yang dibuat dari berbagai subkomponen yang ditunjukkan pada Kode 3.6.

```
const ReusableCalendar = (props: ReusableCalendarProps) => {
  // ...

  return (
    <>
      <SidebarComponent />
      <Calendar />
      <AddEventSidebar />
    </>
  );
}
```

Kode 3.6: Komponen kalender custom yang dibuat di atas *library open-source*

State data kalender pun dikelola menggunakan Redux karena struktur kalender dan interaksi antar komponen yang kompleks, terutama apabila ingin dikembangkan untuk CRUD. *Hook* `useInstrumentCalendar` dibuat khusus untuk

digunakan untuk kalender *booking*, dengan penggunaannya yang dapat dilihat di Kode 3.7.

```
const InstrumentBookingCalendarPage = () => {
  const { calendarStore, loading, error } = useInstrumentCalendar
    ();

  return (
    <>
      // Other UI...

      {!loading && !error && (
        <ReusableCalendar
          storeSelector={() => calendarStore}
          // Other props...
        />
      )}
    </>
  );
}
```

Kode 3.7: Komponen kalender *custom* yang dibuat di atas *library open-source*

Evaluasi internal menunjukkan bahwa tampilan kalender memberikan peningkatan keterbacaan dibandingkan daftar berbasis teks, terutama dalam melihat tumpang tindih jadwal instrumen. Namun, pengembangan modul diberhentikan sementara karena fitur *booking* instrumen tidak termasuk dalam prioritas pengembangan fase ini. Komponen kalender dan struktur transformasi data tetap dipertahankan sebagai referensi untuk pengembangan fitur *booking* interaktif pada tahap mendatang.

D.4 POC Document Editor (Week 10–11)

POC ini dikembangkan sebagai respons terhadap kebutuhan user yang menginginkan pengalaman pengeditan dokumen yang lebih menyerupai aplikasi pengolah dokumen konvensional. Berbeda dengan editor teks ringan yang sebelumnya digunakan, solusi ini menggunakan sebuah *document engine* lengkap, terdiri dari server yang dijalankan sendiri dan *client editor*-nya, yang menyediakan fitur pengaturan paragraf, tabel, gambar, riwayat perubahan dokumen, serta ekspor ke format dokumen umum. Selain itu, *object storage* eksternal juga diintegrasikan untuk menyimpan dokumen. Alur pengambilan dan penyimpanan dokumen di POC

ini dapat diringkas sebagai berikut:

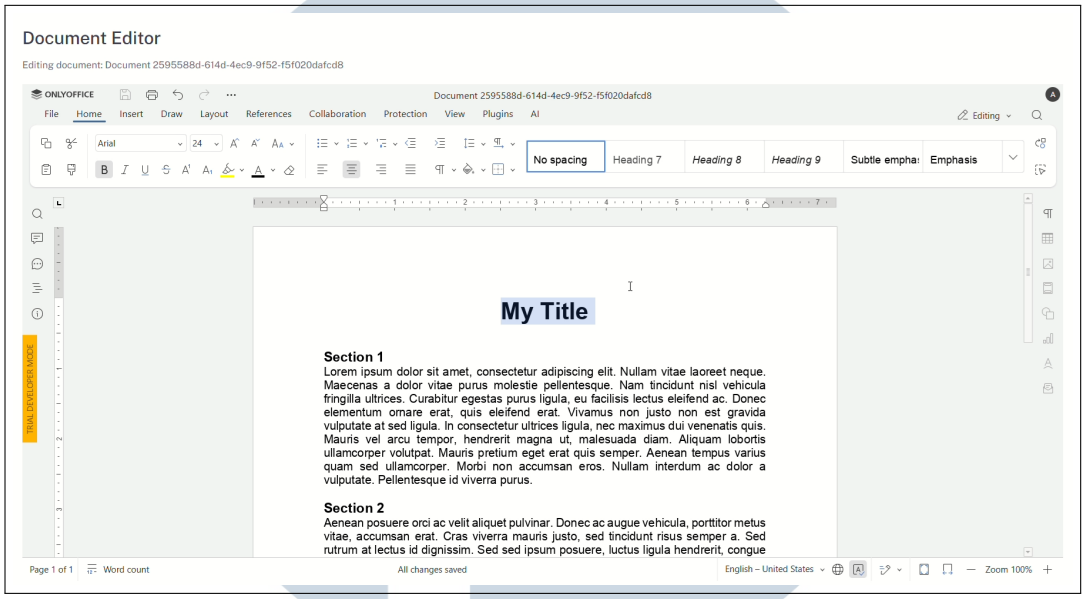
1. *Client* membuat konfigurasi editor untuk dokumen tertentu; apabila belum disediakan ID dokumen, maka dibuat yang baru berdasarkan template yang ditentukan, jika sudah maka dibuat yang baru di *storage*.
2. UI memuat editor berdasarkan konfigurasi dalam bentuk *embedded frame* yang terhubung langsung dengan server, editor mengambil dokumen melalui URL terautentikasi.
3. Ketika user melakukan penyimpanan, server editor mengirimkan *callback* ke aplikasi untuk mengunggah dokumen yang telah diperbarui.

Alur pengambilan dan penyimpanan dokumen juga dapat dilihat di halaman komponen pada Kode 3.8 berikut.

```
const DocumentEditorPage = ({ params }: DocumentEditorPageProps)
    => {
  const {'doc-id': docId} = params;
  const filename = `${docId}.docx`;

  const { data: files } = await getDocStorage().search(filename);

  // If document doesn't exist, create one from template
  if (!files || files.length === 0) {
    await createDocumentFromTemplate(documentsStorage, filename);
  }

  // Get signed URL for the document, 1 hour expiry
  const { data: urlData } = await getDocStorage().createSignedUrl(
   filename, 3600);

  // Build document config
  const config = buildDocConfig({ docId, urlData, ... });

  return (
    <>
      <Title>Document Editor</Title>
      <DocumentEditor config={config} />
    </>
  );
}
```

Kode 3.8: Alur pemuatan dan penyimpanan komponen `DocumentEditorPage`

Tampilan dari `DocumentEditorPage` tersebut pun dapat dilihat di Gambar 3.6, yang menunjukkan judul halaman *Document Editor* dan editor di bawahnya yang sudah memuat suatu dokumen dari *document storage*.



Gambar 3.6. *Document editor* dengan *toolbar* lengkap dan area konten

Hasil pengujian menunjukkan bahwa integrasi dasar berhasil, termasuk pembuatan dokumen awal dari *template*, pemuatan ulang dokumen tersimpan, serta pengiriman callback penyimpanan otomatis. Namun, terdapat anomali pada muatan awal dokumen di mana editor menampilkan pesan kegagalan memuat meskipun dokumen berhasil ditampilkan, yang memerlukan penelusuran sumber masalah lebih lanjut.

Setelah evaluasi, solusi dinilai tidak sejalan dengan kebutuhan ELN, karena:

1. Penyimpanan konten berbasis dokumen lengkap (`.docx`) tidak sesuai dengan kebutuhan yang hanya memerlukan konten teks tersimpan.
2. *Document engine* bersifat monolitik dan tidak modular, sehingga modifikasi logika penyimpanan sesuai kebutuhan memerlukan penanganan *custom* yang kompleks.
3. Kebutuhan menjalankan server sendiri memperkenalkan kompleksitas konfigurasi tambahan dan tidak sebanding manfaat fitur tambahan selain pengeditan teks.

Dengan demikian, POC dihentikan dan tidak dilanjutkan ke tahap implementasi penuh. Catatan teknis dan konfigurasi lingkungan tetap disimpan sebagai referensi apabila kebutuhan pengeditan dokumen berformat lengkap muncul kembali di masa mendatang.

3.3.2 Kontribusi pada AI Chatbot Internal

Proyek AI Chatbot Internal merupakan sebuah chatbot yang digunakan secara internal untuk mendukung efisiensi kerja dan manajemen pengetahuan di perusahaan. Proyek berada pada tahap pengembangan dengan integrasi bertahap ke lingkungan *staging*, sementara backlog ditetapkan sesuai kebutuhan user yang terus berkembang. Kontribusi dalam proyek ini mencakup perbaikan bug UI pada *chat page*, integrasi fitur manual prompt yang menghubungkan frontend dengan API dan backend yang telah tersedia, serta pengembangan halaman Admin Page yang meliputi dari perancangan UI serta implementasi Prompt Form Editor.

A Perbaikan Bug UI (Week 2–3)

Perbaikan bug UI difokuskan pada peningkatan konsistensi tampilan dan kenyamanan user pada *chat page*. Pekerjaan mencakup identifikasi serta penanganan perilaku UI yang tidak sesuai dengan ekspektasi user, khususnya terkait mekanisme *scrollbar* dan *scroll-down button* pada *chat page*.

A.1 Kontribusi Berdasarkan Timeline

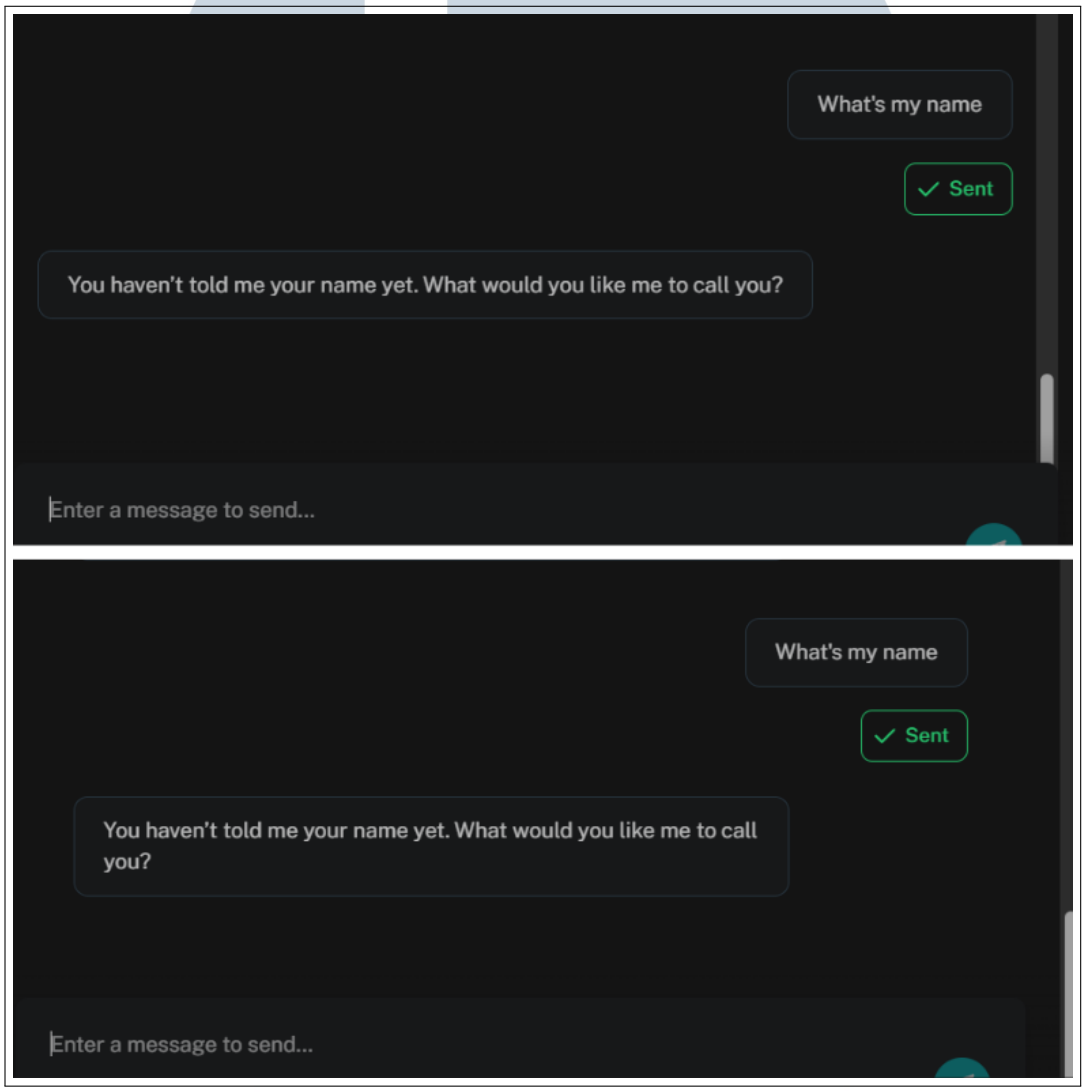
Berikut adalah kontribusi pada perbaikan bug UI berdasarkan timeline:

A.1.1 Week 2: Perbaikan Scrollbar Bug pada Chat Page

Masalah pada tahap ini berkaitan dengan tidak munculnya *scrollbar* internal pada area chat sehingga user tidak dapat menilai posisi *scroll* maupun menavigasi chat dengan tepat. Implementasi awal menyebabkan *scrollbar* muncul pada tingkat *window* aplikasi, bukan pada *container* chat. Hal ini terjadi karena elemen chat tidak memiliki batasan dimensi dan mekanisme *overflow* yang eksplisit, sehingga browser menerapkan perilaku *scroll* pada level halaman secara keseluruhan. Perbaikan dilakukan dengan menambahkan batas ketinggian dan

pengaturan `overflow-y` pada *container* chat sehingga elemen tersebut menjadi satu-satunya area yang dapat di-*scroll*.

Perbandingan tampilan UI sebelum perbaikan (atas) dan sesudah perbaikan (bawah) ditunjukkan pada Gambar 3.7, di mana *scrollbar* pada kondisi sebelum perbaikan muncul pada level *window* sedangkan setelah perbaikan muncul pada *container* chat.



Gambar 3.7. Perbandingan UI *scrollbar* - atas: sebelum (*scrollbar* di *window*), bawah: setelah (*scrollbar* di chat *container*)

Hasil perbaikan menghasilkan *scrollbar* yang tidak tersembunyi di belakang *chatbox* dan diposisikan secara sesuai di sebelah *container* chat.

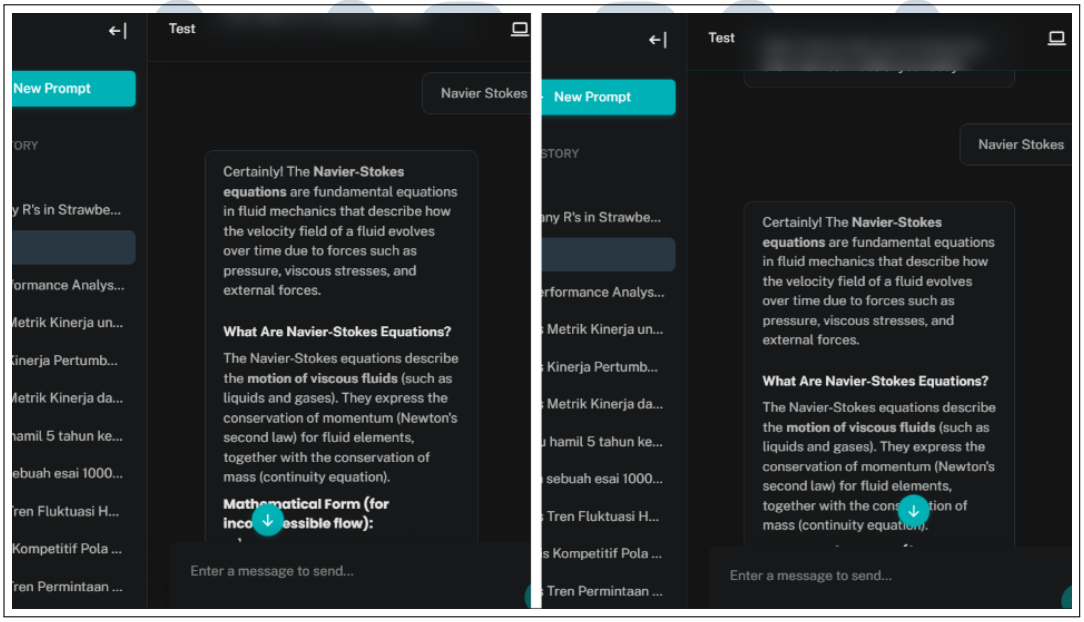
A.1.2 Week 3: Perbaikan Scroll-Down Button Bug pada Chat Page

Masalah berikutnya terkait tombol *scroll-down arrow* yang digunakan untuk membawa user kembali ke bagian bawah chat. Sebelum perbaikan, tombol ini bergantung pada perhitungan posisi absolut layar, sehingga posisinya berubah-ubah bergantung pada lebar halaman, keberadaan *sidebar*, dan tinggi konten. Akibatnya tombol tidak sejajar tengah dengan *container* chat, tetapi dengan *window* aplikasi.

Perbaikan dilakukan dengan:

1. Menempatkan tombol dalam *container* yang mengikuti lebar elemen chat utama, bukan posisi absolut *window*
2. Mengubah orientasi posisi menjadi berbasis *flex layout* agar tombol selalu berada di tengah *container* secara horizontal
3. Menambahkan jarak vertikal tetap dari kotak input untuk menjaga keterbacaan dan aksesibilitas

Perbandingan tampilan tombol *scroll* sebelum perbaikan (kiri) dan sesudah perbaikan (kanan) ditunjukkan pada Gambar 3.8 berikut, di mana sebelum perbaikan tombol berada di tengah relatif terhadap lebar *window* aplikasi, sedangkan setelah perbaikan berada di tengah relatif terhadap lebar *container* chat.



Gambar 3.8. Perbandingan tombol *scroll* - kiri: sebelum (posisi tidak konsisten), kanan: sesudah (posisi *center* konsisten)

Hasil perbaikan memastikan bahwa tombol selalu berada di tengah *container* chat seiring perubahan ukuran layar dan memiliki jarak konsisten dan cukup dari *chatbox*.

A.2 Outcome

Seluruh perbaikan bug UI yang dilakukan telah diintegrasikan ke lingkungan *staging* dan tervalidasi melalui pengujian visual dan fungsional. Penyempurnaan ini meningkatkan konsistensi UI dan memastikan pengalaman user yang lebih stabil pada halaman chat.

B Integrasi Fitur Manual Prompt (Week 3)

Fitur manual prompt dikembangkan sebagai alternatif mekanisme interaksi dengan AI Chatbot internal selain prompt form terstruktur yang telah tersedia sebelumnya. Jika form prompt mengharuskan user mengisi sejumlah *field* terdefinisi, manual prompt memungkinkan user mengetikkan pesan bebas layaknya sistem chat umum. Pada tahap ini API backend untuk pembuatan *session* sudah tersedia, namun belum terdapat implementasi UI maupun integrasi logika pada sisi frontend. Kontribusi utama meliputi pembangunan halaman input manual prompt, penyusunan *server actions* untuk membuat *session* chat baru melalui API, serta penyesuaian komponen UI agar mendukung skenario pembuatan *session* tanpa `conversationId` awal.

Proses integrasi dapat diringkas sebagai berikut:

1. Pembuatan halaman manual prompt sebagai *entry point* pembuatan *session* chat baru
2. Penyesuaian komponen `Chatbox` agar dapat beroperasi tanpa `conversationId`
3. Implementasi *server action* untuk membuat chat baru dan meneruskan pesan pertama ke API
4. Navigasi otomatis ke halaman chat setelah *session* berhasil dibuat

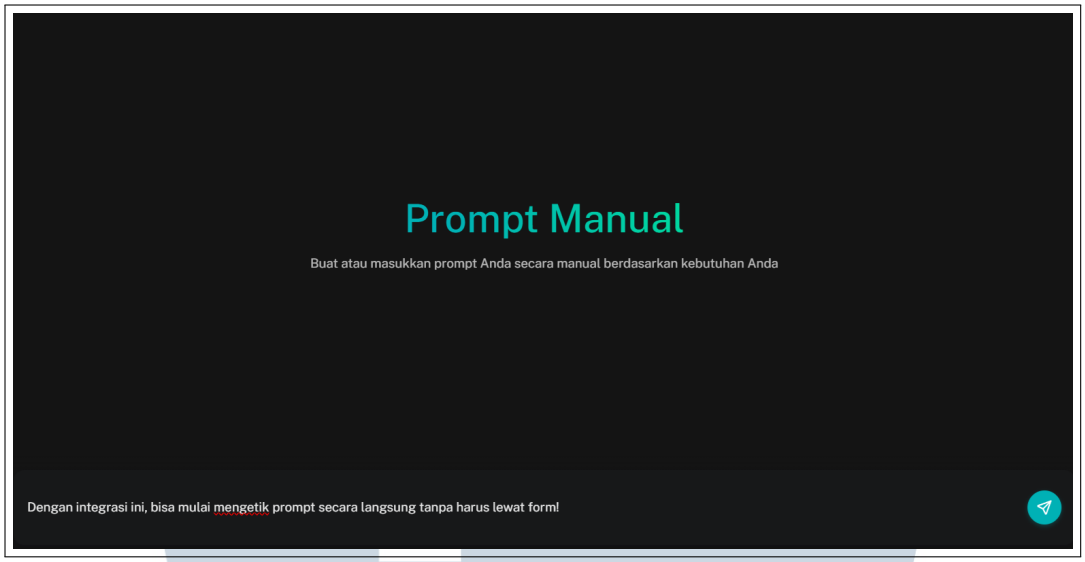
Implementasi inti dari fitur ini ditunjukkan pada Kode 3.9.

```
1 // Client-side: ManualPrompt component
2 const ManualPrompt = () => {
3   const handleMessageSent = async (message: string) => {
```

```
    const result = await createManualConversation(message);

    if (result.success) {
      router.push(`/chat/${result.conversationId}`);
    }
  };

  return <Chatbox onMessageSent={handleMessageSent} />;
};

// Server action: Create conversation and process with API
async function createManualConversation(message: string) {
  // Create conversation record
  const conversation = await db.insertConversation({ message, ...
    });

  // Process through AI API
  await processMessage(conversation.id, { message, userId,
    conversationId: conversation.id });

  return { success: true, conversationId: conversation.id };
}
```

Kode 3.9: Implementasi integrasi manual prompt

Tampilan UI sesudah integrasi ditunjukkan pada Gambar 3.9, yang menunjukkan halaman simpel dengan judul *Prompt Manual*, sebuah *caption* sederhana, serta `Chatbox` yang dapat diketik langsung layaknya sistem chat umum.



Gambar 3.9. Tampilan halaman chat manual prompt yang sudah berfungsi

Perubahan teknis yang dilakukan mencakup:

1. *Refactoring* logika pemrosesan pesan menjadi satu *utility* yang digunakan baik oleh manual prompt maupun *session* chat biasa
2. Penyesuaian struktur `Chatbox` sehingga parameter `conversationId` bersifat opsional dan dapat mengembalikan isi pesan pertama melalui *callback*
3. Implementasi mekanisme `loading state` agar pengiriman prompt tidak dieksekusi ganda
4. Penanganan *fallback* error, sehingga apabila terjadi kegagalan pemrosesan oleh API, *session* tetap dibuat dan pesan sistem dicatat sebagai penanda gagal

Hasil integrasi memastikan bahwa user dapat memulai sebuah chat dengan metode manual prompt dengan hasil struktur chat, pencatatan metadata, penyimpanan pesan, serta pembuatan *session* yang selaras dengan standar sistem chat dan metode yang sudah ada (form prompt).

B.1 Outcome

Fitur Manual Prompt telah berhasil diintegrasikan dengan frontend dan backend secara konsisten, serta telah melalui proses evaluasi internal sebelum digabungkan ke lingkungan *staging*. Integrasi ini memastikan bahwa user dapat mengirimkan prompt langsung melalui UI chat dengan struktur respons yang selaras dengan sistem yang sudah ada.

C Pengembangan Admin Page (Week 6–16)

Fitur Admin Page dikembangkan dari awal sebagai bagian dari perluasan fungsi konfigurasi internal pada sistem chatbot. Pengembangan mencakup perancangan UI, penataan konteks dan *layout* admin, serta penyusunan struktur fungsionalitas utama yang akan digunakan untuk mendukung pengelolaan konfigurasi internal. Pada tahap ini, halaman admin dirancang untuk memiliki dua fitur inti, yaitu halaman Prompt Form Editor serta User Management. Adapun komponen User Management belum diimplementasikan pada saat penulisan laporan ini sehingga fokus pengembangan berada pada perancangan dan integrasi Prompt Form Editor.

C.1 Kontribusi Berdasarkan Timeline

Berikut adalah kontribusi pada Admin Page berdasarkan timeline:

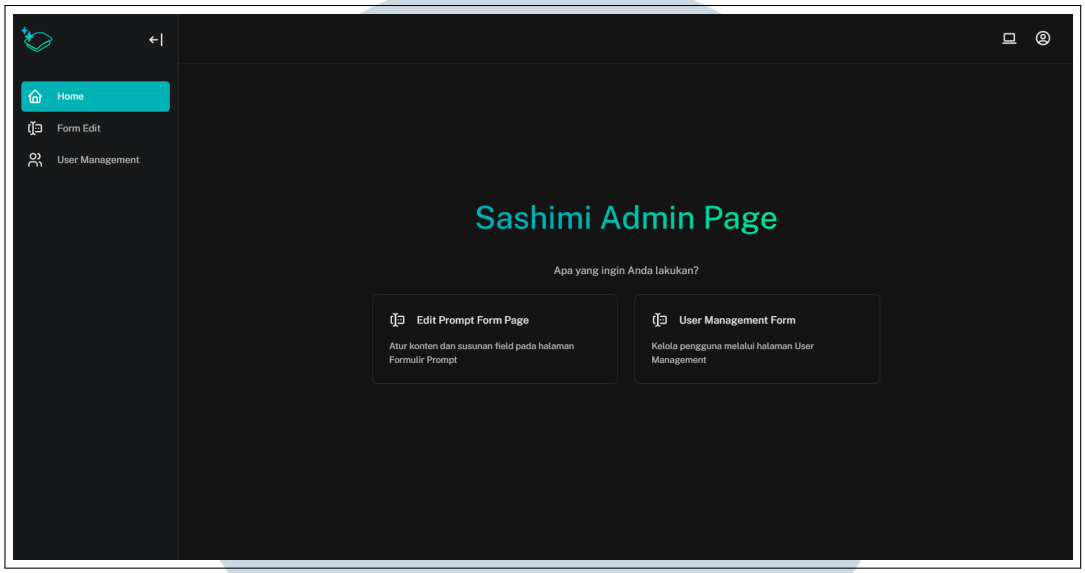
C.1.1 Week 6: UI Awal Admin Page

Tahap ini berfokus pada penyusunan struktur dasar Admin Page sebagai landasan untuk pengembangan fitur-fitur konfigurasi internal. Admin Page dirancang sebagai halaman terpisah dengan *layout* khusus, sehingga diperlukan pemisahan konteks UI, penyusunan ulang navigasi, serta adaptasi komponen agar konsisten dengan gaya visual aplikasi yang sudah ada. Pada tahap ini belum terdapat logika bisnis maupun integrasi data; implementasi dipusatkan pada kerangka tampilan dan arsitektur navigasi.

Struktur dasar dikembangkan melalui pembuatan admin *layout* yang membungkus seluruh halaman di bawah rute `/admin`. *Layout* ini memuat *sidebar* khusus admin, area konten utama, serta mekanisme responsif yang mengatur perilaku *sidebar* pada tampilan *desktop* dan *mobile*. *Sidebar* disiapkan dengan navigasi menuju dua fitur inti, yaitu Prompt Form Editor dan User Management, dengan halaman kedua masih berupa *placeholder*. Mekanisme navigasi diatur melalui struktur rute bertingkat sehingga setiap halaman admin dapat memanfaatkan *layout* bersama.

Halaman beranda admin disiapkan sebagai titik masuk utama, berisi kartu navigasi menuju fitur-fitur yang tersedia. Struktur awal ini memastikan bahwa modul Prompt Form Editor dapat diintegrasikan secara bertahap tanpa mengubah

fondasi *layout*. Tampilan awal halaman admin dengan *sidebar*, header dinamis, dan kartu navigasi diperlihatkan pada Gambar 3.10.



Gambar 3.10. Admin Page dengan *sidebar* kiri, *header*, dan kartu navigasi ke Prompt Form Editor dan User Management

C.1.2 Week 7–16: Kontribusi pada Prompt Form Editor

Kontribusi pada rentang waktu ini sepenuhnya berkaitan dengan pengembangan Prompt Form Editor, mencakup perancangan UI, penyesuaian dan pembentukan skema database, integrasi editor dengan database, penyusunan mekanisme *save*, integrasi editor dengan Prompt Form yang tersedia, hingga penyusunan mekanisme validasi. Penjelasan rinci mengenai pengembangan Prompt Form Editor akan disampaikan pada subbagian berikutnya.

C.2 Outcome

Kontribusi yang telah direalisasikan pada Admin Page, terutama pada Prompt Form Editor, telah melalui evaluasi internal dan sudah terintegrasi ke lingkungan *staging*. Struktur halaman dan mekanisme penyusunan form dinamis telah berfungsi sebagaimana dirancang. Namun, fitur User Management belum memulai pengembangan pada fase ini dan direncanakan untuk tahap berikutnya.

D Prompt Form Editor (Week 7–16)

Prompt Form Editor merupakan salah satu dari dua fitur utama yang dikembangkan pada Admin Page dan menjadi fokus utama kontribusi dalam rentang pengembangan ini. Sebelumnya, sistem chatbot hanya memiliki satu jenis Prompt Form statis, yaitu formulir terstruktur berisi sejumlah *field* tetap yang harus diisi user sebelum mengirimkan permintaan ke backend. Prompt Form digunakan sebagai alternatif dari mekanisme manual prompt dan bertujuan memastikan bahwa permintaan user memiliki struktur, kelengkapan, dan konsistensi sesuai metode prompt tertentu. Namun, karena seluruh struktur Prompt Form masih bersifat statis dan didefinisikan langsung di dalam kode, proses perubahan maupun penambahan *field* harus dilakukan melalui modifikasi manual di sisi pengembangan.

Prompt Form Editor dikembangkan untuk menjadikan konfigurasi Prompt Form bersifat dinamis dengan memindahkan seluruh definisi *field*, struktur, dan pengaturan validasi ke dalam database. Melalui fitur ini, admin yang berwenang dapat menambah, mengubah, menghapus, serta mengatur urutan *field* secara langsung melalui UI editor, dan perubahan tersebut akan tercermin pada Prompt Form yang digunakan user tanpa perlu modifikasi kode. Proses pengembangan fitur ini memerlukan analisis menyeluruh terhadap struktur Prompt Form yang kompleks, perancangan skema data dan mekanisme penyimpanan, hingga integrasi UI dan penyelarasan perilaku antara sistem lama dan sistem baru agar transisi terjadi secara mulus.

D.1 Kontribusi Berdasarkan Timeline

Berikut adalah kontribusi pada fitur Prompt Form Editor untuk Admin Page berdasarkan timeline:

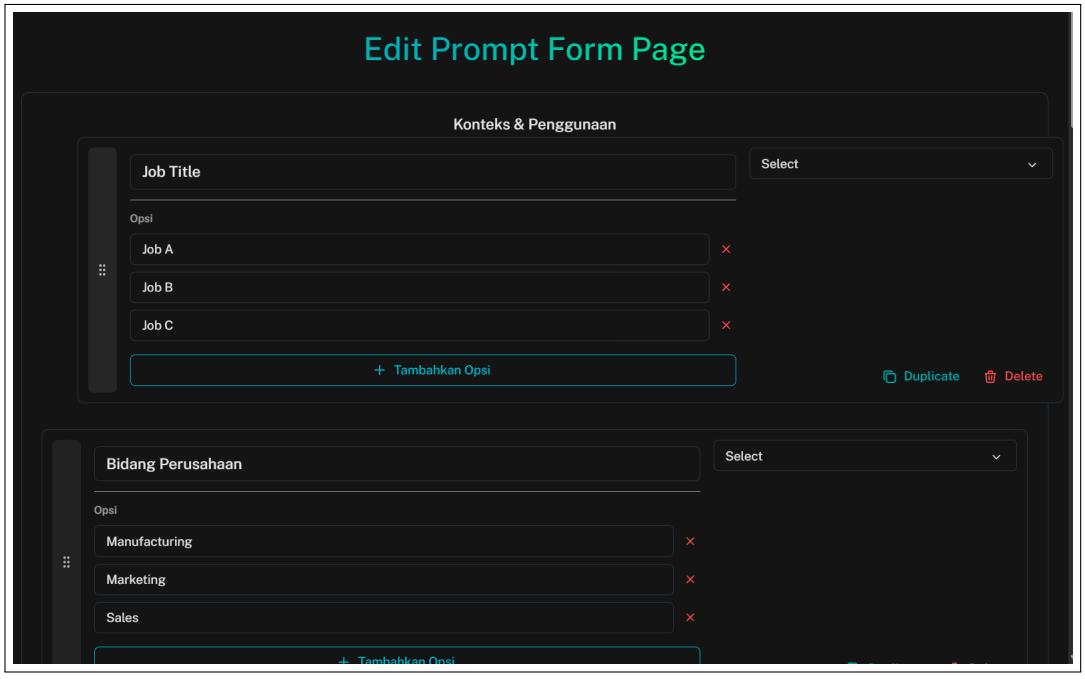
D.1.1 Week 7: Mock-Up Draggable dan Interaktivitas Dasar

Tahap awal pengembangan Prompt Form Editor difokuskan pada pembuatan struktur visual dasar editor dan implementasi mekanisme *drag-and-sort* untuk menyusun urutan *field*. Pada fase ini, seluruh perubahan masih bersifat lokal, belum terhubung dengan mekanisme penyimpanan maupun struktur data akhir yang digunakan pada sistem produksi. *Field* disusun menggunakan data *dummy*, namun struktur komponen, tipe data, serta interaksi utama telah merepresentasikan perilaku yang diharapkan.

Fungsionalitas utama yang berhasil diimplementasikan pada tahap ini meliputi:

1. *Drag-and-drop* untuk menyusun urutan *field* dalam satu *section*
2. *Inline field* editing untuk nama *field*, tipe *field*, dan parameter dasar lainnya
3. Struktur komponen modular untuk memisahkan `FieldBlock` (komponen *field* individu), `SectionBlock` (komponen *section* yang berisi kumpulan *field*), dan *wrapper* `Sortable` (komponen *wrapper* yang mengelola dan memungkinkan isinya di-*drag-and-drop*)
4. Sistem tipe awal menggunakan `discriminated union` untuk menjaga konsistensi struktur data

Tampilan umum editor dengan *field* yang dapat di-*drag* ditunjukkan pada Gambar 3.11 berikut dengan `FieldBlock` yang berbentuk *draggable block*, dengan *field-field* yang memungkinkan pengubahan *field name*, *field type*, opsi, dsb. Dapat dilihat juga `FieldBlock` pertama *field* yang sedang dalam proses *drag* untuk mendemonstrasikan fungsi *drag-and-drop*, sehingga terlihat sedikit tergeser ke kanan.



Gambar 3.11. Prompt Form Editor dengan beberapa *field* dan satu *field* sedang di-drag

Sistem *drag-and-drop* dibangun menggunakan *third-party library* yang menyediakan mekanisme *reordering* dengan dukungan aksesibilitas penuh. Implementasi ditunjukkan pada Kode 3.10, menampilkan komponen `SectionBlock` yang mengelola *state* lokal daftar *field* dan menangani *event* perpindahan posisi. Komponen ini membungkus setiap *field* dalam `SortableWrapper` yang secara internal meneruskan *drag listeners* hanya ke komponen `DragHandle`, memastikan bahwa elemen input dan tombol lain tetap dapat berinteraksi secara normal tanpa memicu operasi *drag*.

```
// SectionBlock.tsx - manages field reordering
const SectionBlock = ({ fields }: Props) => {
  const [localFields, setLocalFields] = useState(fields);
  const { listeners, ...rest } = useSortableContext();

  const handleDrag = (event: DragEvent) => {
    const { active, over } = event;
    if (!over || active.id === over.id) return;

    const oldIndex = localFields.findIndex(f => f.id === active.id
    );
    const newIndex = localFields.findIndex(f => f.id === over.id);
    setLocalFields(arrayMove(localFields, oldIndex, newIndex));
  };

  return (
    <SortableContext onDrag={handleDrag}>
      {localFields.map(field => (
        <SortableWrapper key={field.id} id={field.id}>
          <FieldBlock fieldInfo={field} dragListeners={listeners}
    />
        </SortableWrapper>
      ))}
    </SortableContext>
  );
};
```

Kode 3.10: Implementasi mekanisme *drag-and-drop* field editor

Setiap *field* individual dikelola oleh komponen `FieldBlock` yang berfungsi sebagai editor interaktif dengan kemampuan pengeditan properti secara *inline*. Komponen ini mengelola *state* internal untuk nilai-nilai *field* dan menyediakan UI visual yang terbagi menjadi tiga area: `DragHandle` untuk operasi *reordering*, *editable content area* untuk pengeditan properti, dan *action buttons* untuk operasi

duplicate dan *delete*. Di dalam *editable content area*, `DynamicFieldEditor` bertanggung jawab me-*render* kontrol tambahan secara dinamis berdasarkan tipe *field* yang dipilih, misal untuk tipe `select`, komponen menampilkan editor daftar opsi dengan kemampuan menambah dan menghapus item. Mekanisme rendering kondisional ini memungkinkan editor beradaptasi terhadap perubahan tipe *field* tanpa memerlukan pembaruan manual pada struktur UI. Struktur komponen `FieldBlock` ditunjukkan pada Kode 3.11.

```
// FieldBlock.tsx - interactive field editor
const FieldBlock = ({ fieldData, dragListeners }: FieldBlockProps)
    => {
  return (
    <div>
      {/* Drag handle for reordering */}
      <DragHandle {...dragListeners} />

      {/* Editable content area */}
      <div>
        <NameField />
        <TypeSelectField />
        <DynamicFieldEditor fieldType={fieldData.type} /> {/*
   Dynamic based on field type */}
      </div>

      {/* Action buttons */}
      <div className='field-actions'>
        <DuplicateButton />
        <DeleteButton />
      </div>
    </div>
  );
};
```

Kode 3.11: Implementasi komponen `FieldBlock` untuk editing *field* individual

Fondasi ini memungkinkan pengembangan tahap berikutnya untuk fokus pada persistensi data, sinkronisasi dengan backend, serta implementasi fungsional penuh dari aksi editor seperti penambahan, penghapusan, dan duplikasi *field*.

D.1.2 Week 8–9: Skema Data, Types, RPCs, DDL, dan DML

Tahap ini berfokus pada penyusunan skema data dinamis yang menjadi dasar arsitektur Prompt Form Editor. Struktur Prompt Form statis yang sebelumnya

tertanam langsung pada kode dianalisis ulang untuk dipetakan menjadi model tiga lapis, yaitu *section*, daftar *field*, dan opsi tambahan bagi *field* yang membutuhkannya. Dengan desain ini, konfigurasi form tidak lagi bergantung pada struktur *hard-coded*, melainkan dapat diambil, diubah, dan disimpan melalui mekanisme data terpusat.

Perancangan skema database dilakukan dengan mempertimbangkan fleksibilitas terhadap tujuh tipe *field* yang digunakan dalam sistem, termasuk dua tipe yang memiliki karakteristik khusus. Setiap *field* dapat menyimpan metadata tambahan sesuai kebutuhannya, misalnya opsi terstruktur atau batasan nilai. Pendekatan ini memastikan bahwa representasi data di backend dan representasi data di frontend tetap konsisten.

Struktur tabel disusun untuk mengakomodasi kebutuhan tersebut yang dapat dilihat sebagai skema DDL pada Kode 3.12.

```
CREATE TABLE form_section (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT NOT NULL,
  position INTEGER NOT NULL,
  is_active BOOLEAN NOT NULL DEFAULT TRUE,
);

CREATE TABLE form_field (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  section_id UUID NOT NULL REFERENCES form_section(id) ON DELETE
    CASCADE,
  field_type TEXT NOT NULL,
  name TEXT NOT NULL,
  placeholder TEXT,
  position INTEGER NOT NULL,
  metadata JSONB DEFAULT '{}',  -- konfigurasi per tipe field
);

CREATE TABLE field_option (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  field_id UUID NOT NULL REFERENCES form_field(id) ON DELETE
    CASCADE,
  value TEXT NOT NULL,
  parent_option_id UUID REFERENCES field_option(id),
  position INTEGER NOT NULL,
);
```

Kode 3.12: Struktur tabel inti Prompt Form

Selanjutnya disusun tipe data pada sisi frontend yang merepresentasikan konfigurasi form dalam bentuk objek statis. Setiap tipe *field* memiliki struktur metadata tertentu, dan seluruhnya disatukan dalam tipe union yang digunakan oleh editor untuk melakukan rendering dinamis. Struktur tipe data ditunjukkan pada Kode 3.13.

```
interface BaseField {
  id: string
  name: string
  placeholder?: string
  position: number
}

interface TextField extends BaseField {
  type: 'text'
}

// Other field types extending from BaseField...

type FormField =
  | TextField
  | SelectField
  | CheckboxField
  | DateRangeField
  | NumberField
  | CompositeSelectField
  | NestedSelectField
```

Kode 3.13: Struktur tipe data dinamis untuk field Prompt Form

Agar frontend dapat mengambil data form secara terstruktur, disusun sebuah RPC awal yang mengembalikan hierarki *section*, *field*, dan opsi sebagai satu objek JSON. Tujuannya adalah menyederhanakan proses pemanggilan data pada sisi frontend dan mengurangi kebutuhan *query* berulang. RPC ini ditunjukkan pada Kode 3.14.

```
CREATE FUNCTION get_prompt_form_structure()
RETURNS JSON
AS $$
DECLARE
  sections JSON;
BEGIN
  SELECT json_agg(section_data ORDER BY position)
  INTO sections
```

```
  FROM (
    SELECT
      s.*,
      (SELECT json_agg(field_data ORDER BY position)
       FROM (
         SELECT
           f.*,
           (SELECT json_agg(o ORDER BY position)
            FROM field_option o
            WHERE o.field_id = f.id) AS options
         FROM form_field f
         WHERE f.section_id = s.id
       ) field_data) AS fields
    FROM form_section s
  ) section_data;

  RETURN sections;
END;
$$ LANGUAGE plpgsql;
```

Kode 3.14: RPC pengambilan struktur form

Tahap berikutnya adalah penyusunan DML awal untuk menghasilkan *seed* data. *Seed* ini merekonstruksi Prompt Form statis yang sebelumnya tertulis sebagai konfigurasi *hard-coded* pada kode frontend. Data tersebut mencakup seluruh *section*, seluruh *field* dari masing-masing *section*, serta daftar opsi yang jumlahnya paling banyak berasal dari tipe select dan checkbox. Proses ini memastikan bahwa editor dapat menampilkan form lengkap tanpa perlu kembali pada implementasi statis lama.

D.1.3 Week 10–12: Integrasi Database dengan Frontend Editor

Tahap ini berfokus pada penghubungan editor Prompt Form dengan data nyata yang tersimpan dalam database. Dengan skema, tipe data, dan RPC yang telah tersedia pada tahap sebelumnya, struktur form tidak lagi bersumber dari *dummy* data, tetapi dibangun secara dinamis berdasarkan hasil pembacaan tabel `form_section`, `form_field`, dan `field_option` yang relevan. Integrasi ini membutuhkan penyesuaian pada hierarki data, normalisasi tipe *field*, koreksi struktur *nested field* khusus, serta mekanisme pembaruan *state* agar perubahan yang dilakukan pada UI terselaraskan dengan representasi data backend.

Pada tingkat pengambilan data, editor kini melakukan pemanggilan RPC

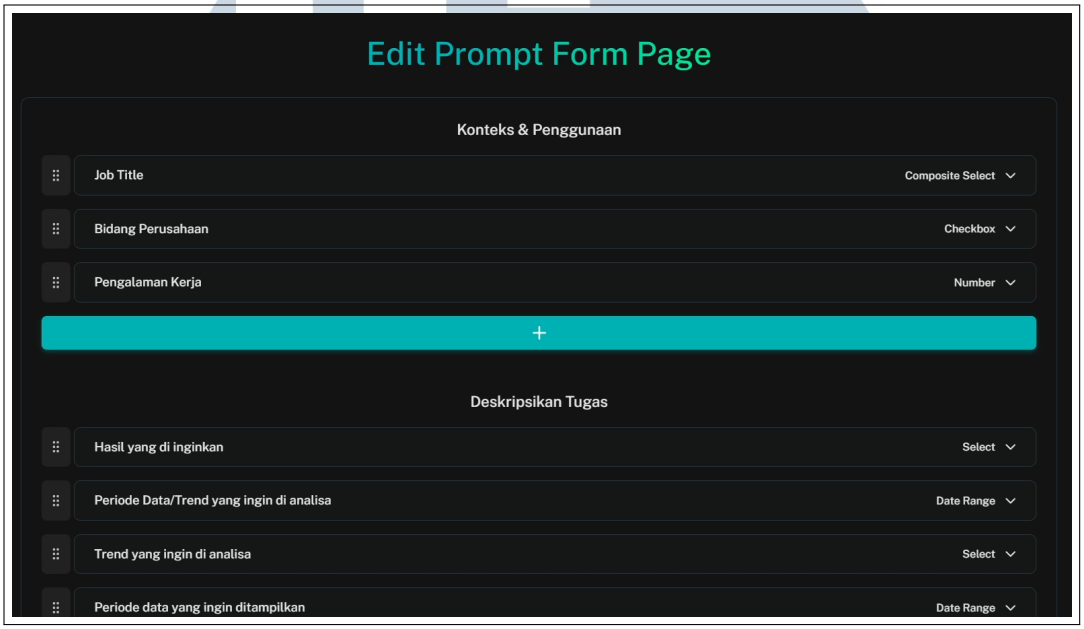
untuk memperoleh struktur form secara hierarkis dalam satu kali *request* sehingga mengurangi kompleksitas penggabungan data di frontend. Pada sisi representasi data, setiap *field* dinormalisasi menjadi bentuk internal yang konsisten sehingga dapat diproses oleh komponen-komponen editor tanpa perlu bergantung pada format asli tabel. Mekanisme pengubahan *field* kemudian direvisi untuk menangani seluruh tipe yang didukung, termasuk dua tipe *field* unik yang memiliki struktur *nested*. Alur *state* form dari database ke UI ditunjukkan pada Kode 3.15, yang menampilkan pola *server action* untuk pengambilan data serta normalisasi *field* dengan *discriminated union*.

```
// Server action: fetch and normalize form data
async function loadEditorState(): Promise<FormSection[]> {
  const rawSections = await rpc('get_form_structure');

  return rawSections.map(section => ({
    id: section.id,
    name: section.name,
    position: section.position,
    fields: section.fields.map(normalizeField)
  }));
}

// Normalize field based on discriminated union type
function normalizeField(raw: RawField): FormField {
  const base = {
    id: raw.id,
    name: raw.name,
    type: raw.type,
    position: raw.position
  };

  switch (raw.type) {
    case 'text':
      return { ...base, type: 'text', placeholder: raw.placeholder
    };
    // Other cases for different field types
  }
}
```

Kode 3.15: Integrasi data form dengan normalisasi tipe *field*

Integrasi ini menghasilkan editor yang mampu membaca, menampilkan, dan menyusun ulang struktur form secara dinamis dari database. Implementasi

type-safe dengan *discriminated union* memastikan konsistensi tipe data. Tampilan editor setelah integrasi ini ditunjukkan pada Gambar 3.12, di mana *field-field* yang ditampilkan kini berasal dari database, dengan masing-masing *field* berbentuk *collapsible block* yang dapat diperluas untuk menampilkan properti serta dapat diatur posisinya, dikelompokkan sesuai dengan `form_section`-nya.



Gambar 3.12. Prompt Form Editor setelah integrasi database dengan *field* dari database

D.1.4 Week 13–14: Mekanisme Saving dan Integrasi Editor dengan Prompt Form

Tahap ini berfokus pada implementasi mekanisme penyimpanan dengan deteksi perubahan otomatis. Pendekatan *autosave* tidak diterapkan karena frekuensi perubahan yang tinggi (misal saat mengetik nama *field*) akan menyebabkan beban kueri berlebihan, sementara editor tidak menargetkan kolaborasi *real-time*. Sebagai gantinya, diterapkan pola *edit-lock* dengan tombol *save* manual yang dilengkapi deteksi perubahan otomatis melalui perbandingan *deep equality* antara *state* saat ini dengan *state* tersimpan.

Mekanisme ini menyimpan representasi *pristine state* saat *initial load*, kemudian membandingkannya secara kontinu dengan *working state* menggunakan `useMemo`. Ketika terdeteksi perbedaan, UI menampilkan indikator *unsaved changes* dan mengaktifkan tombol *save*. Penyimpanan dilakukan secara transaksional dengan validasi sebelumnya, memastikan konsistensi data. Implementasi deteksi

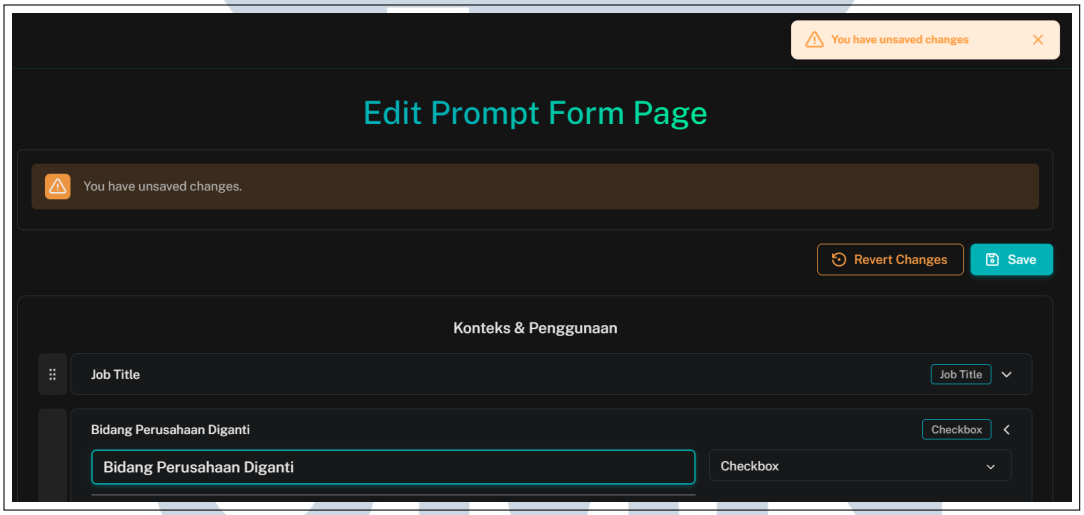
perubahan ditunjukkan pada Kode 3.16.

```
// Change detection with deep equality comparison
const FormEdit = ({ initialFormBlocks }: Props) => {
  const [formBlocks, setFormBlocks] = useState(initialFormBlocks);
  const [savedFormBlocks, setSavedFormBlocks] = useState(
   initialFormBlocks);

  // Detect unsaved changes via JSON comparison
  const hasUnsavedChanges = useMemo(
    () => JSON.stringify(formBlocks) !== JSON.stringify(
   savedFormBlocks),
    [formBlocks, savedFormBlocks]
  );

  const handleSave = async () => {
    // 1. Validate form structure
    const validation = validateForm(formBlocks);
    if (!validation.isValid) {
      showErrors(validation.errors);
      return;
    }

    // 2. Save entire state transactionally
    const result = await saveFormData(formBlocks);

    if (result.success) {
      // 3. Refetch to sync with database
      const refetched = await getAllFormSectionsWithFields();
      setSavedFormBlocks(refetched.data);
      setFormBlocks(refetched.data);
    }
  };

  const handleRevert = () => {
    setFormBlocks(savedFormBlocks);  // Reset to pristine state
  };

  return (
    // Form Editor UI
  );
};
```

Kode 3.16: Deteksi perubahan dan alur penyimpanan

Mekanisme *deep equality* melalui `JSON.stringify` pada Kode 3.16 memungkinkan deteksi perubahan otomatis tanpa perlu melacak setiap properti secara manual. Ketika user mengetik kembali nilai asli (*manual revert*), perbandingan JSON akan mendeteksi kesamaan dan menonaktifkan indikator perubahan. Penyimpanan dilakukan secara transaksional melalui RPC `save_form_data_transaction` yang menangani *upsert section*, *field*, dan opsi dalam satu transaksi database, memastikan konsistensi data dan memungkinkan *rollback* otomatis jika terjadi kesalahan.

Pada UI editor, ketika user melakukan perubahan pada *field*, sistem mendeteksi dan menampilkan indikator bahwa terdapat perubahan yang belum disimpan. Tampilan editor dengan indikator *unsaved changes* ditunjukkan pada Gambar 3.13. User dapat memilih untuk menyimpan perubahan atau membatalkannya. Ketika tombol *revert* ditekan, perubahan yang belum disimpan dikembalikan ke *state* awal.



Gambar 3.13. Editor dengan indikator *unsaved changes* aktif

Integrasi dengan Prompt Form memerlukan transformasi struktur editor menjadi format yang dapat di-*render* oleh komponen form. Editor menyimpan data dalam struktur *flat* (*section* dengan *array field*, opsi sebagai *array* terpisah), sedangkan Prompt Form memerlukan struktur *nested* untuk beberapa tipe unik. Transformasi ini dilakukan melalui `buildPromptForm()` yang memetakan setiap *field* ke komponen UI yang sesuai berdasarkan `field_type`, seperti ditunjukkan pada Kode 3.17.

```
1 // Transform database structure to Prompt Form props
```

```
function buildPromptForm(formSections: FormSection[]):
    ReactElement[] {
  return formSections.map(section => (
    <FormSection key={section.id} title={section.name}>
      {section.fields.map(field => {
        const baseProps = {
          key: field.id,
          name: field.name,
          placeholder: field.placeholder,
          required: field.validation_type === 'required'
        };

        // Map field type to appropriate component
        switch (field.field_type) {
          case 'text':
            return <TextField {...baseProps} />;

          // Other cases for different field types
        }
      })}
    </FormSection>
  ));
}
```

Kode 3.17: *Adapter* struktur editor ke Prompt Form

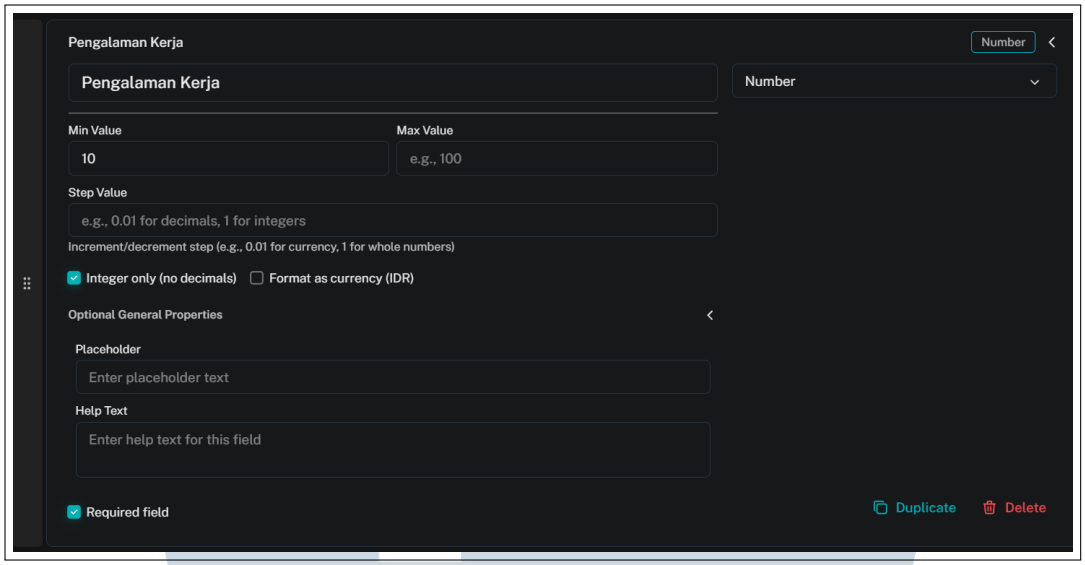
Dengan ini, Prompt Form telah dimodifikasi untuk selalu memuat data dari database. Penyelarasan visual memastikan versi dinamis tetap identik dengan versi statis dalam hal UI dan perilaku komponen. Dengan integrasi ini, Prompt Form beroperasi sepenuhnya menggunakan konfigurasi dari editor, memungkinkan admin mengubah struktur form tanpa modifikasi kode. Tampilan Prompt Form dinamis setelah integrasi ini ditunjukkan pada Gambar 3.14, dengan *field-field* di setiap `form_section`, dipetakan sesuai dengan *state* yang dikonfigurasi di editor.

Gambar 3.14. Prompt Form dinamis yang terintegrasi dengan konfigurasi dari editor

D.1.5 Week 15–16: Validation Schema & Fix Bug Chat Generation dari Prompt Form Integrasi

Tahap ini berfokus pada implementasi skema validasi dinamis yang tersimpan sebagai JSONB di database. Setiap *field* menyimpan `validation_schema` yang berisi aturan validasi spesifik per tipe: `minLength`/`maxLength` untuk teks, `min`/`max` untuk angka, `earliest`/`latest` untuk tanggal, dan `minItems`/`maxItems` untuk opsi. Struktur ini memungkinkan konfigurasi validasi dari editor tanpa perubahan kode, dengan aturan yang konsisten antara editor dan Prompt Form.

Konfigurasi validasi ditampilkan sebagai UI tambahan di `FieldBlock` editor, memungkinkan admin mengatur batasan dan status wajib isi untuk setiap *field*. Tampilan *field* dengan pengaturan validasi ditunjukkan pada Gambar 3.15, dengan pengaturan umum seperti nama *field*, pengaturan spesifik berdasarkan tipe seperti nilai *min* dan *max*, serta properti umum baru seperti *placeholder* dan *help text*.



Gambar 3.15. `FieldBlock` dengan *section settings* validasi (*required*, min, max, dll)

Mekanisme validasi diimplementasikan melalui *builder* yang mengonversi `validation_schema` JSONB menjadi *runtime validator* menggunakan *validator library*. Konversi ini ditunjukkan pada Kode 3.18, yang membaca aturan dari database dan membentuk validator sesuai tipe *field*.

```
// Convert ValidationSchema JSONB to runtime validator
const buildValidator = (schema: ValidationSchema): Validator => {
  const { type, rules, messages } = schema;
  let validator: Validator;

  // Build type-specific validators
  switch (type) {
    case 'string':
      validator = createStringValidator();
      if (rules?.pattern) {
        validator.addValidation(rules.pattern, messages?.pattern);
      }
      // Other rules to consider...
      break;

    // Other cases for different types of fields
  }

  return validator;
}
```

Kode 3.18: Konversi `validation_schema` ke *runtime validator*

Validator yang dihasilkan diterapkan pada Prompt Form, memungkinkan validasi *real-time* sebelum submit. Pesan kesalahan yang ditampilkan diambil dari `messages` dalam `validation_schema`, memastikan konsistensi antara konfigurasi editor dan *user feedback*.

Tahap ini juga mencakup perbaikan bug pada proses pembuatan chat baru. API chat mengharuskan seluruh *payload* berupa *string*, namun *field* bertipe angka menghasilkan nilai numerik setelah validasi.

Dengan implementasi skema validasi JSONB, konversi ke *runtime validator*, dan normalisasi *payload*, integrasi Prompt Form dinamis berhasil diselesaikan secara menyeluruh.

D.2 Outcome

Fitur Prompt Form Editor telah dinilai berfungsi dengan baik dan telah digabungkan ke lingkungan *staging* setelah melalui proses evaluasi internal. Seluruh integrasi inti, termasuk penyusunan struktur form dinamis, mekanisme penyimpanan, dan pemetaan data ke Prompt Form baru telah berjalan sesuai kebutuhan. Fitur ini telah siap digunakan oleh admin untuk mengelola struktur form secara langsung tanpa perlu modifikasi kode.

3.4 Kendala dan Solusi yang Ditemukan

Selama proses pengembangan sistem, sejumlah tantangan teknis ditemukan pada implementasi fitur-fitur terkait ELN dan AI Chatbot internal. Kendala tersebut mencakup permasalahan pada desain data, integrasi UI, mekanisme penyimpanan, hingga konsistensi perilaku API. Berikut merupakan rangkuman kendala utama beserta solusi yang diterapkan untuk mengatasinya.

3.4.1 Implementasi Server-Side Pagination

Tantangan muncul ketika menerapkan mekanisme *pagination* pada modul User & Roles serta Admin & Tenant Management. API dari penyedia identitas tidak menyediakan dukungan *pagination* yang fleksibel, terutama ketika permintaan data disertai parameter pencarian ataupun filtering kolom secara simultan. Penggunaan *offset-based pagination* juga terbukti tidak efisien untuk dataset berukuran besar.

Solusi: Dilakukan evaluasi terhadap berbagai pendekatan *fetching*, dan metode *progressive batched fetching* dipilih karena lebih efisien serta konsisten dalam menangani volume data besar. Pengujian dilakukan dengan berbagai variasi parameter pencarian dan filtering untuk memastikan perilaku API tetap stabil pada seluruh skenario.

3.4.2 Struktur Dinamis Prompt Form

Kompleksitas tinggi muncul ketika merancang Prompt Form Editor yang harus mendukung struktur formulir dinamis dengan berbagai tipe *field*, termasuk opsi bertingkat dan relasi antaropsi. Beberapa *field* memiliki struktur yang lebih kompleks dibandingkan select umum, sehingga generalisasi tipe data menjadi tidak stabil dan menghasilkan inkonsistensi representasi antar*field*. Hal ini menyulitkan penyusunan skema data serta integrasi frontend dengan struktur dinamis tersebut.

Solusi: Skema data dirancang ulang menggunakan konfigurasi berbasis `JSONB` agar setiap *field* dapat menyimpan struktur dan metadata secara fleksibel. Pendekatan ini memudahkan penyimpanan opsi bertingkat serta aturan tambahan tiap *field* tanpa membebani struktur database inti. Setelah konsultasi dan evaluasi, ditetapkan bahwa dua *field* dengan struktur kompleks akan diperlakukan sebagai tipe unik tersendiri, bukan diturunkan dari tipe select dasar. Pendekatan ini mengurangi kompleksitas generalisasi serta meningkatkan konsistensi data dalam editor.

3.4.3 Perancangan Mekanisme Penyimpanan pada Prompt Form Editor

Tantangan signifikan muncul ketika merancang mekanisme penyimpanan perubahan pada Prompt Form Editor. Pendekatan *autosave* tidak diimplementasikan karena berpotensi menghasilkan beban *query* tinggi ke database, meskipun telah menggunakan *debounce*. Selain itu, editor tidak memiliki kebutuhan kolaborasi *real-time* yang biasanya menjadi alasan utama penggunaan *autosave*. Pendekatan manual save sederhana juga tidak ideal karena berisiko menyebabkan penyimpanan ulang seluruh struktur formulir meskipun perubahan yang dilakukan bersifat minimal, terutama karena data tersebar pada banyak tabel dan baris.

Solusi: Dirancang mekanisme tracking perubahan yang mampu mendeteksi

selisih antara *state* awal dan *state* baru. Sistem ini memungkinkan user menunda penyimpanan jika diperlukan, mendukung opsi *revert*, dan hanya mengirimkan perubahan aktual dalam satu operasi simpan ketika tombol *save* ditekan. Pendekatan ini mengurangi *overhead* API, menghindari *upsert* tidak perlu, serta menghasilkan alur penyimpanan yang lebih efisien dan aman. Mekanisme ini diuji secara iteratif mengingat banyaknya skenario perubahan pada struktur dinamis editor, namun berhasil distabilkan dan disetujui setelah serangkaian evaluasi.

3.4.4 Bug pada Proses Generasi Chat dari Prompt Form Dinamis

Setelah integrasi skema validasi pada Prompt Form dinamis, sebuah bug muncul ketika sistem mencoba menghasilkan sesi percakapan baru dari formulir tersebut. Proses gagal pada sisi backend dan hanya menampilkan pesan kesalahan generik, sehingga sulit menelusuri penyebabnya. Pengujian menunjukkan bahwa formulir statis lama tetap berfungsi dengan baik, mengindikasikan bahwa masalah muncul dari perbedaan struktur payload yang dihasilkan oleh Prompt Form dinamis.

Solusi: Setelah membandingkan *payload* antara versi statis dan dinamis melalui serangkaian percobaan, ditemukan bahwa salah satu nilai pada *payload* dihasilkan sebagai tipe numerik, sedangkan backend mengharapkan seluruh nilai dikirim dalam bentuk string. Perbedaan kecil ini menyebabkan proses pembuatan chat gagal pada sisi API. Setelah seluruh nilai distandardisasi menjadi string sebelum pengiriman, proses generasi chat kembali berjalan normal. Pengalaman ini juga memberikan pelajaran betapa pentingnya error *logging* yang deskriptif untuk masa mendatang.