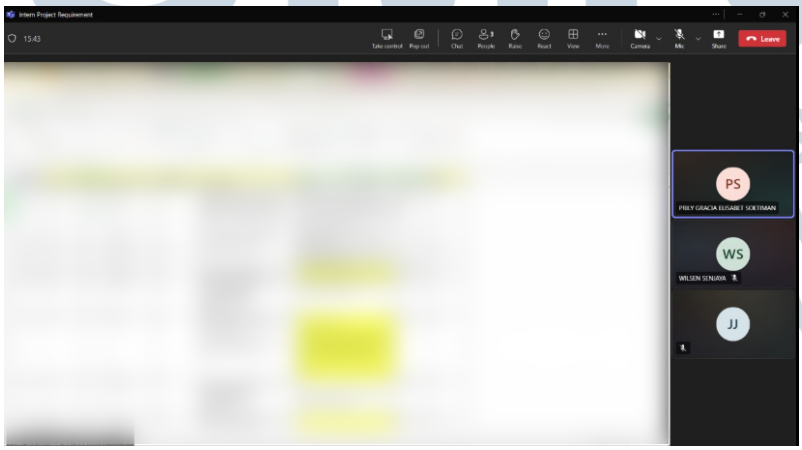
BAB 3
Pelaksanaan Magang

3.1 Kedudukan dan Organisasi

Selama pelaksanaan kerja magang di Divisi Digital Banking Bank SMBC, mahasiswa magang ditempatkan sebagai *Full Stack Developer Intern* yang bertanggung jawab atas pengembangan *Complaint Management System* internal. Struktur organisasi dan koordinasi kerja magang dapat dijelaskan sebagai berikut:

1. **Supervisor Digital Banking:** Bertanggung jawab memberikan arahan teknis, supervisi proyek, dan evaluasi berkala terhadap progres pengembangan sistem. Supervisor juga menjadi penghubung antara mahasiswa magang dengan manajemen divisi dan unit terkait.
2. **Full Stack Developer Intern:** Bertugas melakukan perancangan sistem, pengembangan antarmuka pengguna (*front end*), pengembangan layanan dan basis data (*back end*), integrasi komponen sistem, serta pengujian fungsional. Posisi ini menuntut kemampuan bekerja secara mandiri dengan tanggung jawab penuh atas siklus pengembangan aplikasi.

Koordinasi dilakukan secara rutin melalui *weekly meeting* dengan supervisor untuk melakukan *progress review*, penyelarasan prioritas pengembangan, dan evaluasi teknis. Komunikasi *ad-hoc* memanfaatkan kanal daring (chat/video call) sesuai kebutuhan untuk diskusi teknis mendalam atau penyelesaian *blocker* yang bersifat mendesak.



Gambar 3.1. Sesi koordinasi daring mingguan dengan supervisor.

Kedudukan sebagai *Full Stack Developer* yang bekerja secara mandiri memberikan kesempatan untuk terlibat dalam berbagai tahapan pengembangan sistem, mulai dari riset antarmuka pengguna, implementasi fitur, hingga pengujian dan dokumentasi. Fokus utama pada tahap awal adalah riset UI untuk memastikan sistem mudah digunakan dan familiar bagi pengguna, mengingat pentingnya aspek *usability* dalam sistem manajemen komplain internal. Alur kerja yang dijalankan mencakup: riset dan perancangan UI/UX, implementasi bertahap (frontend–backend), integrasi antar komponen, uji fungsional dan *debugging*, serta penyusunan dokumentasi.

3.2 Tugas yang Dilakukan

Tugas utama yang dilaksanakan selama masa magang adalah merancang dan membangun sistem manajemen komplain berbasis web yang digunakan untuk mendukung operasional penanganan pengaduan secara terpusat. Aplikasi dikembangkan secara *end-to-end* dalam peran sebagai *full-stack developer*, dengan menggunakan Next.js dan React pada sisi *frontend* serta layanan Firebase (Authentication, Firestore/Realtime Database, dan Storage) pada sisi *backend*. Sistem dirancang untuk mendukung tiga peran utama, yaitu User, Staff, dan Admin, lengkap dengan mekanisme autentikasi dan otorisasi hak akses.

Secara garis besar, proses pengembangan dimulai dari analisis kebutuhan dan penyusunan rancangan sistem berupa diagram *use case*, flowchart alur komplain, serta struktur data untuk tiket, pesan percakapan, dan *audit log*. Setelah rancangan disepakati, penulis mengimplementasikan struktur data dan layanan backend, menyusun skema koleksi pada Firestore, mengonfigurasi autentikasi, serta membuat *API route* untuk pengelolaan tiket, chat, perubahan status, dan penyajian data statistik.

Tahap berikutnya adalah pengembangan antarmuka pengguna yang meliputi halaman login, formulir pembuatan tiket, dashboard admin, ruang chat per tiket, *badge* notifikasi, serta modul statistik, disertai fitur pendukung seperti unggah gambar, pencatatan *audit log*, ekspor ke Excel, dan kontrol status tiket dari *New* $\rightarrow$ *In Progress* $\rightarrow$ *Closed*. Seluruh kegiatan tersebut dilaksanakan secara bertahap setiap minggunya, sebagaimana dirangkum pada Tabel 3.1.

Tabel 3.1. Uraian Pelaksanaan Magang Selama 16 Minggu.

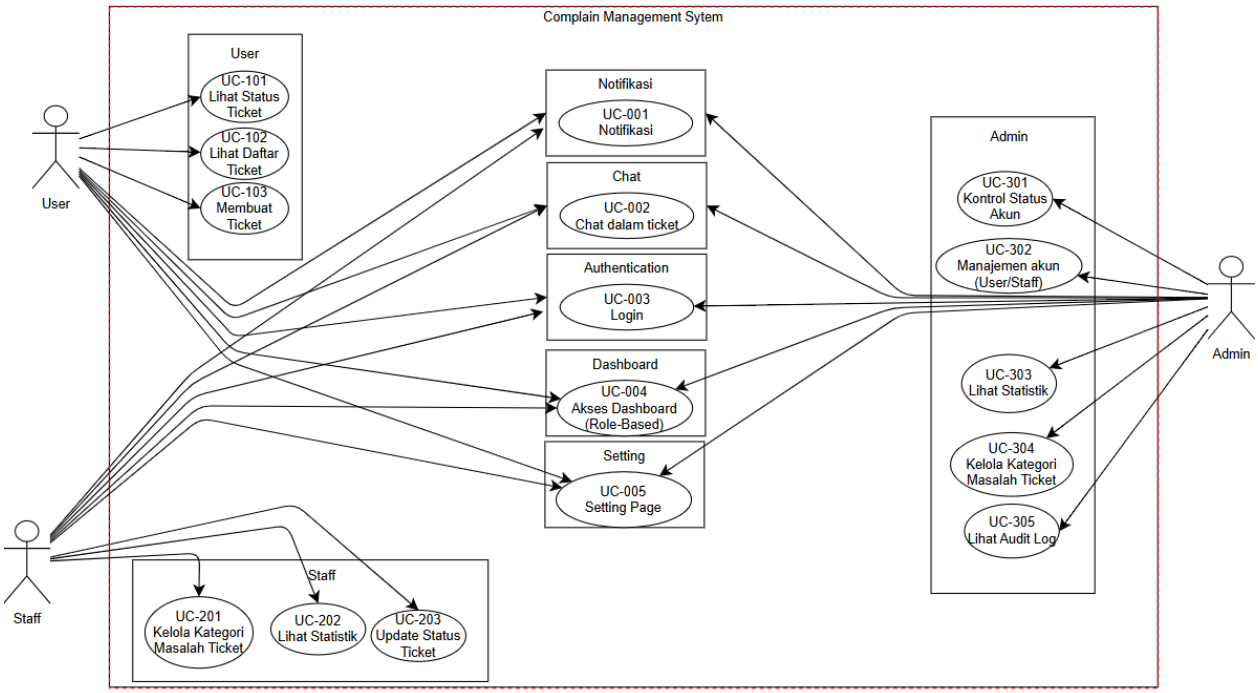
Minggu Ke-	**Pekerjaan yang Dilakukan**
1	Pengenalan lingkungan kerja, memahami kebutuhan sistem manajemen komplain, mempelajari dasar Next.js App Router, struktur proyek, integrasi awal Firebase (Authentication, Firestore, Storage), serta menyiapkan repositori dan konfigurasi awal aplikasi.
2	Menyusun rancangan sistem: diagram *use case*, alur proses komplain (flowchart), identifikasi aktor (User, Staff, Admin), serta perancangan struktur data tiket, chat, dan *audit log* pada Firestore. Menyusun draft arsitektur *frontend-backend*.
3	Implementasi autentikasi Firebase: pembuatan halaman login, middleware autentikasi, validasi peran (role-based access), dan pengamanan routing. Menguji *session* serta kondisi akses bagi tiga peran pengguna.
4	Pengembangan halaman pembuatan tiket: form input, unggah lampiran dasar, validasi data, serta penyimpanan data tiket ke Firestore. Menyusun struktur koleksi `tickets` dan metadata status.
5	Pengembangan Dashboard Admin: menampilkan daftar tiket, menyusun tabel komplain, mengimplementasikan fitur pencarian, *filter* status, dan penyortiran berdasarkan tanggal. Menambahkan pagination berbasis query Firebase.
6	Membangun ruang chat per tiket: struktur koleksi pesan, input chat, mekanisme *real-time listener* Firebase, tampilan percakapan dinamis, serta pembeda pesan pengguna/admin.
7	Menambahkan fitur unggah gambar pada tiket dan chat menggunakan Firebase Storage, membuat progres bar unggah, serta menampilkan pratinjau file. Mengoptimalkan ukuran file dan keamanan URL download.
8	Implementasi *badge notifikasi*: mendeteksi pesan baru, menandai tiket dengan aktivitas terbaru, serta sinkronisasi indikator notifikasi antara Dashboard, Header, dan tampilan chat.

Minggu Ke-	Pekerjaan yang Dilakukan
9	Pengembangan *audit log*: mencatat riwayat perubahan status tiket, aktivitas pengguna, dan tindakan admin. Merancang struktur koleksi `audit_logs`, serta tampilan tabel riwayat di Admin Panel.
10	Pengembangan modul statistik: grafik jumlah komplain per periode, kategori terbanyak, status komplain, dan ringkasan aktivitas. Menggunakan query agregasi Firestore dan integrasi grafik di Dashboard Admin.
11	Implementasi fitur kontrol status tiket: perubahan dari *New* $\rightarrow$ *In Progress* $\rightarrow$ *Closed*, validasi peran yang boleh mengubah status, serta notifikasi otomatis pada perubahan status.
12	Pengembangan fitur ekspor data ke Excel: menyiapkan struktur data ekspor, membangun generator Excel, serta menyediakan *download handler* di Dashboard. Pengujian kelengkapan data ekspor.
13	*Refactor* kode frontend: pemecahan komponen menjadi lebih modular (form field, kartu tiket, tabel, modal), perbaikan struktur folder, optimalisasi *hooks*, dan pengurangan duplikasi logika.
14	Pengujian responsivitas di berbagai perangkat (mobile, tablet, desktop), perbaikan *layout*, alur navigasi, dan penyelarasan UI/UX. Penyesuaian warna, spasi, fokus, serta aksesibilitas form.
15	Pengujian menyeluruh seluruh modul: login, tiket, chat, notifikasi, status, unggah file, dan statistik. Melakukan perbaikan bug, validasi batasan akses, dan penyesuaian performa Firebase (pengurangan read/write berlebih).
16	Finalisasi sistem: pembersihan kode, penyusunan dokumentasi teknis (alur sistem, struktur data, API internal, skenario uji), serta persiapan demo/serah terima kepada pembimbing dan tim internal.

3.3 Uraian Pelaksanaan Magang

3.3.1 Use Case Diagram

Use case diagram pada Gambar 3.2 menggambarkan hubungan antara aktor (User, Staff, dan Admin) dengan fungsi-fungsi utama yang tersedia dalam Complaint Management System. Setiap aktor memiliki hak akses terhadap modul yang berbeda, namun tetap terhubung melalui mekanisme autentikasi dan dashboard berbasis peran.



Gambar 3.2. Use Case Diagram Complaint Management System

3.3.2 Daftar Modul Berdasarkan Peran Pengguna

Berdasarkan use case diagram pada Gambar 3.2, modul-modul yang dapat diakses oleh masing-masing peran dalam sistem dapat dirangkum pada Tabel 3.2.

Tabel 3.2. Daftar Modul Berdasarkan Peran Pengguna

Peran	Modul yang Dapat Diakses
User (Pengguna)	• UC-101: Lihat Status Ticket • UC-102: Lihat Daftar Ticket • UC-103: Membuat Ticket
Staff (Petugas)	• UC-201: Kelola Kategori Masalah Ticket • UC-202: Lihat Statistik • UC-203: Update Status Ticket
Admin (Administrator)	• UC-301: Kontrol Status Akun • UC-302: Manajemen Akun (User/Staff) • UC-303: Lihat Statistik • UC-304: Kelola Kategori Masalah Ticket • UC-305: Lihat Audit Log
Modul Umum	• UC-001: Notifikasi • UC-002: Chat dalam Ticket • UC-003: Login (Authentication) • UC-004: Akses Dashboard (Role-Based) • UC-005: Setting Page

3.3.2.1 Modul Authentication (Login)

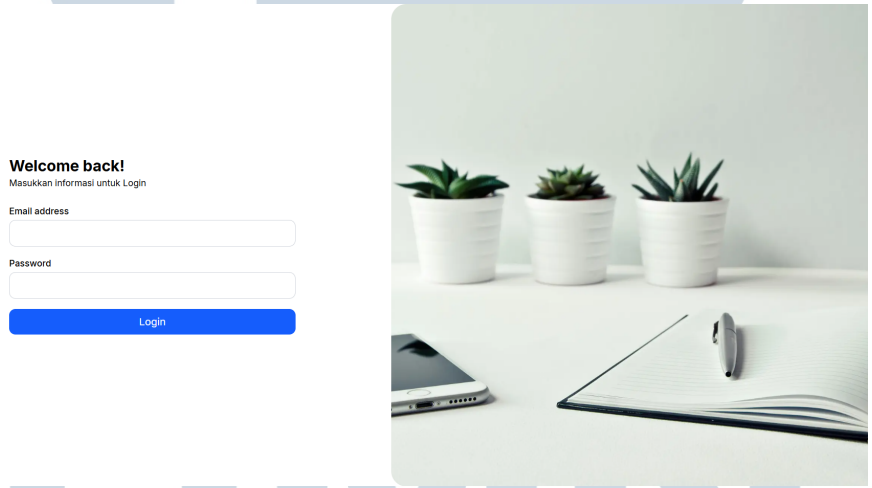
Modul authentication (login) berfungsi sebagai gerbang utama untuk mengakses Complaint Management System. Melalui modul ini, setiap pengguna (User, Staff, maupun Admin) diwajibkan melakukan proses login terlebih dahulu sebelum dapat menggunakan fitur lain pada sistem. Proses login memastikan bahwa hanya akun yang sah dan masih aktif yang dapat mengakses aplikasi, sekaligus menjadi dasar penentuan hak akses (*role-based access control*) untuk menampilkan dashboard dan menu sesuai peran masing-masing.

Secara umum, alur login dimulai ketika pengguna membuka halaman login dan memasukkan kredensial berupa alamat surel dan kata sandi. Sistem kemudian meneruskan kredensial tersebut ke layanan autentikasi untuk diverifikasi. Jika data yang dimasukkan valid, sistem akan membuat sesi login dan menyimpan informasi

identitas serta peran pengguna, lalu mengarahkan pengguna ke dashboard sesuai perannya. Apabila proses autentikasi gagal, misalnya karena kata sandi salah atau akun dinonaktifkan, sistem akan menampilkan pesan galat dan pengguna diminta untuk memperbaiki input atau menghubungi administrator.

Implementasi Frontend

Dari sisi *frontend*, halaman login dirancang dengan pendekatan yang sederhana dan fokus agar pengguna dapat langsung memahami tujuan halaman tanpa distraksi. Tampilan dibagi menjadi dua bagian utama: panel kiri berisi formulir login, sedangkan panel kanan menampilkan ilustrasi visual. Pembagian ini memberikan keseimbangan antara fungsi dan estetika, sekaligus memanfaatkan ruang layar lebar pada perangkat *desktop*.



Gambar 3.3. Tampilan halaman login Complaint Management System

Pada panel formulir, komponen utama yang ditampilkan adalah judul sambutan *"Welcome back!"*, keterangan singkat berbahasa Indonesia, serta dua buah isian yaitu *Email address* dan *Password*. Penggunaan tipografi huruf tebal untuk judul, diikuti teks penjelas yang lebih ringan, ditujukan untuk mengarahkan fokus pengguna secara bertahap dari konteks umum ke instruksi yang lebih spesifik. Komponen tombol *Login* diberi warna biru yang kontras dengan latar belakang putih untuk menegaskan bahwa elemen tersebut merupakan *primary action* yang harus diklik oleh pengguna.

Dari sisi tata letak, formulir disusun secara vertikal sehingga alur pengisian berjalan dari atas ke bawah dengan urutan yang logis: judul, deskripsi singkat, isian

email, isian kata sandi, kemudian tombol login. Margin, spasi antar komponen, dan radius sudut pada *input field* serta tombol diatur agar tampilan terasa modern dan konsisten dengan gaya desain antarmuka web kontemporer. Desain ini juga dibuat responsif menggunakan utilitas CSS sehingga pada ukuran layar yang lebih kecil, panel gambar dapat disembunyikan atau diposisikan ulang agar formulir tetap mudah diakses dan dibaca oleh pengguna.

Implementasi Backend

Pada sisi *backend*, proses login diimplementasikan melalui endpoint `POST /api/auth/login`. Endpoint ini menerima body berformat JSON yang wajib memuat `idToken` (Firebase ID Token) yang diperoleh dari sisi *frontend*. Selanjutnya, backend memverifikasi token menggunakan *Firebase Admin SDK* untuk memastikan bahwa token valid, belum kedaluwarsa, dan tidak berada pada kondisi tidak dapat digunakan (misalnya telah dicabut).

Apabila verifikasi berhasil, backend mengekstrak UID dari token untuk mengidentifikasi pengguna. Backend kemudian memastikan data pengguna tersedia pada basis data, misalnya dengan membuat dokumen pengguna `users/{uid}` apabila belum ditemukan. Setelah itu, backend menghasilkan *session cookie* melalui mekanisme autentikasi berbasis sesi (`createSessionCookie`), lalu mengirimkannya melalui cookie `HttpOnly` bernama `authToken`. Selain cookie utama tersebut, backend dapat menambahkan cookie pendukung seperti `sessionExp` dan `lastActive` untuk mendukung pengelolaan masa berlaku sesi dan mekanisme *idle timeout* pada middleware. Sebagai penutup, backend mengembalikan respons JSON yang menyatakan bahwa proses login berhasil.

Jika `idToken` tidak dikirim atau bernilai kosong, backend mengembalikan respons *Bad Request* (400). Jika verifikasi token gagal, backend mengembalikan respons *Unauthorized* (401) beserta pesan galat. Rangkuman status dan makna respons API login ditampilkan pada Tabel 3.3. Contoh format request dan respons yang digunakan pada masing-masing skenario ditampilkan pada Tabel 3.4.

Tabel 3.3. Ringkasan Status dan Makna Respons API Login

Kondisi	**Status HTTP**	**Makna dan Dampak**
Login berhasil	200	Token valid, backend membuat sesi dan mengirim cookie autentikasi agar sesi tetap aktif pada request berikutnya.
`idToken` tidak dikirim / kosong	400	Request tidak memenuhi field wajib sehingga proses autentikasi dihentikan sebelum verifikasi token dilakukan.
Token tidak valid / kedaluwarsa / verifikasi gagal	401	Token gagal diverifikasi sehingga backend menolak pembuatan sesi dan tidak mengirim cookie autentikasi.

Tabel 3.4. Contoh Request dan Respons API Login

Skenario	**Contoh Request dan Respons**

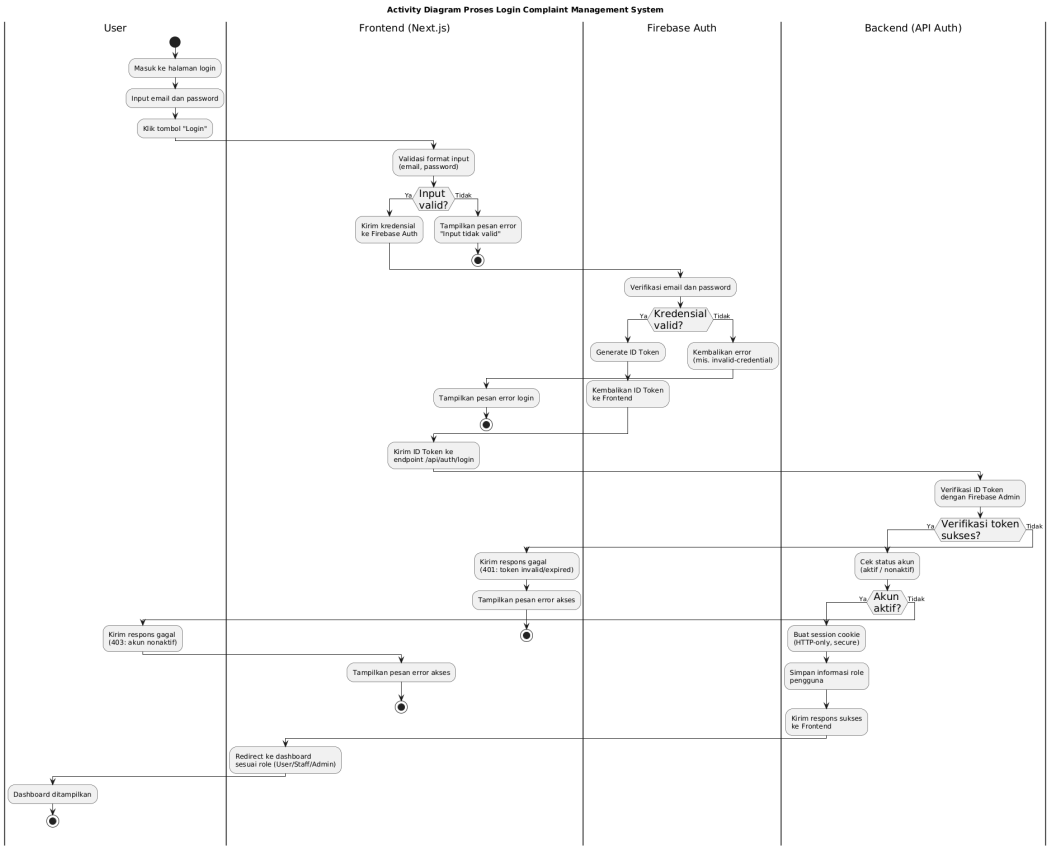
Login berhasil (200)	`Request:`
	`POST /api/auth/login`
	`Content-Type: application/json`
	`Body: {"idToken":"<ID_TOKEN>"}`
	
	`Respons:`
	`HTTP/1.1 200 OK`
	`Set-Cookie: authToken=<...>; HttpOnly; ...`
	`Body: {"message":"Login berhasil"}`

`idToken` tidak dikirim/kosong (400)	`Request:`
	`POST /api/auth/login`
	`Content-Type: application/json`
	`Body (opsi 1): {}`
	`Body (opsi 2): {"idToken":""}`
	
	`Respons:`
	`HTTP/1.1 400 Bad Request`
	`Body: {"message":"idToken diperlukan","error":"BAD_REQUEST"}`

Token tidak valid/kedaluwarsa (401)	`Request:`
	`POST /api/auth/login`
	`Content-Type: application/json`
	`Body: {"idToken":"<INVALID_TOKEN>"}`
	
	`Respons:`
	`HTTP/1.1 401 Unauthorized`
	`Body: {"message":"Gagal login","error":"UNAUTHORIZED","detail":"Token tidak valid atau kedaluwarsa"}`

Activity Diagram

Untuk memberikan gambaran lebih jelas mengenai proses autentikasi, Gambar 3.4 menunjukkan alur aktivitas login mulai dari pengguna melakukan input hingga sistem menentukan apakah proses autentikasi berhasil atau gagal.



Gambar 3.4. Activity diagram proses login Complaint Management System

3.3.2.2 Modul Akses Dashboard (Role-Based)

Modul dashboard berfungsi sebagai pusat aktivitas setelah pengguna berhasil login. Sistem menampilkan menu dan modul yang berbeda berdasarkan peran pengguna, yaitu *User*, *Staff*, dan *Admin*. Pembatasan akses diterapkan pada level fitur, sehingga pengguna hanya melihat fungsi yang relevan dengan kewenangannya.

Konsep Akses dan Hak Peran

Dashboard pada sistem menggunakan satu *shell* halaman yang sama untuk seluruh pengguna. Setelah proses login berhasil, sistem menentukan peran pengguna (*User*, *Staff*, atau *Admin*) dan menggunakan informasi tersebut sebagai dasar pengaturan navigasi serta modul yang dapat diakses. Pendekatan ini menjaga struktur antarmuka tetap konsisten bagi seluruh pengguna, namun kontrol akses tetap diterapkan sehingga setiap peran hanya melihat fitur yang relevan dengan tanggung jawabnya (*role-based access control*). Pemetaan fitur berdasarkan peran ditampilkan pada Tabel 3.5.

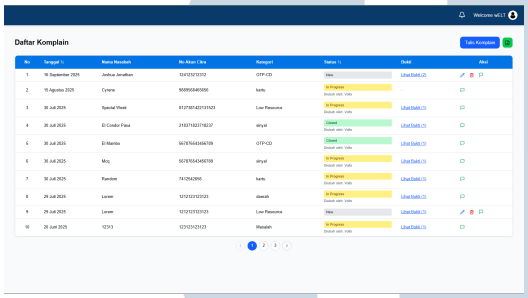
Tabel 3.5. Pemetaan fitur dashboard berdasarkan peran pengguna

Peran	**Jenis Akses**	**Fitur yang Ditampilkan pada Dashboard**
User	Terbatas	UC-101 (Lihat Status Ticket), UC-102 (Lihat Daftar Ticket), UC-103 (Membuat Ticket), UC-001 (Notifikasi), UC-002 (Chat dalam Ticket).
Staff	Operasional	UC-201 (Kelola/Update Status Ticket), UC-001 (Notifikasi), UC-002 (Chat dalam Ticket).
Admin	Penuh	UC-101/UC-102 (Monitoring Ticket), UC-201 (Kontrol Penanganan), Manajemen Akun (Khusus Admin), Audit Log (Khusus Admin), Manajemen Kategori Komplain, Statistik Komplain, serta modul umum UC-001 (Notifikasi) dan UC-002 (Chat dalam Ticket).

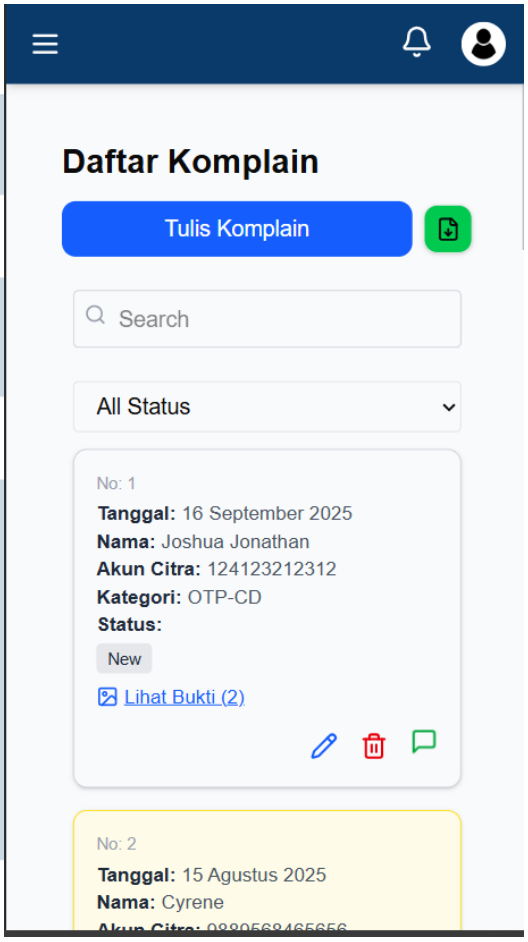
Implementasi Frontend dan Responsif

Pada sisi *frontend*, perbedaan akses antar peran diterapkan melalui *conditional rendering* pada komponen navigasi dan modul dashboard. Sistem hanya menampilkan menu dan modul yang sesuai dengan peran pengguna sehingga antarmuka tetap fokus dan mengurangi risiko akses fitur yang tidak berwenang dari sisi tampilan. Walaupun demikian, kontrol akses utama tetap ditegakkan pada sisi server agar tidak dapat dilewati hanya dengan manipulasi antarmuka.

Tampilan dashboard juga dibuat responsif agar nyaman digunakan pada berbagai ukuran layar. Pada perangkat *desktop*, informasi ditampilkan dalam format tabel untuk memudahkan pemindaian data dan melihat banyak kolom sekaligus. Pada perangkat *mobile*, data disajikan dalam format kartu agar lebih mudah dibaca dan dioperasikan melalui sentuhan, tanpa mengurangi inti informasi yang ditampilkan.



Gambar 3.5. Tampilan dashboard pada perangkat desktop.



Gambar 3.6. Tampilan dashboard pada perangkat mobile.

Validasi Akses pada Backend

Pada sisi *backend*, validasi dilakukan untuk memastikan dashboard tidak dapat diakses tanpa sesi login. Middleware memeriksa keberadaan dan validitas *session cookie* (`authToken`) sebelum permintaan ke rute dashboard diproses lebih lanjut. Jika sesi tidak valid, sistem mengembalikan respons *redirect* ke halaman login. Jika sesi valid, akses dashboard diizinkan dan sistem melanjutkan pemrosesan halaman. Dengan strategi ini, akses dashboard tetap aman meskipun pengguna mencoba mengakses rute privat secara langsung melalui URL.

Call dan Respons Akses Dashboard

Akses dashboard tidak menggunakan endpoint API khusus, melainkan berupa permintaan HTTP ke halaman `GET /dashboard`. Middleware bertindak sebagai *guard* yang menentukan respons berdasarkan kondisi sesi, sebagaimana dirangkum pada Tabel 3.6.

Tabel 3.6. Respons akses dashboard berdasarkan validasi sesi

Kondisi	**Respons HTTP**	**Hasil**
`authToken` valid	200 OK	Halaman dashboard ditampilkan, lalu modul dan menu dirender sesuai peran pengguna.
`authToken` tidak ada / tidak valid	3xx Redirect	Dialihkan ke halaman login karena sesi dianggap tidak aktif.
`authToken` ada tetapi sesi kedaluwarsa/idle timeout	3xx Redirect	Cookie sesi dihapus, kemudian dialihkan ke halaman login.

Dashboard User

Dashboard *User* berfokus pada pelaporan komplain dan pemantauan tiket yang dibuat. Pengguna dapat membuat tiket baru, melihat daftar tiket, memantau status penanganan, serta melakukan komunikasi melalui chat di dalam tiket. Notifikasi ditampilkan untuk membantu pengguna mengetahui pembaruan penting tanpa harus memeriksa tiket satu per satu.

Dashboard Staff

Dashboard *Staff* difokuskan pada aktivitas operasional penanganan tiket. Staff menerima tiket yang masuk, melakukan komunikasi lanjutan melalui chat, serta memperbarui status penanganan sesuai progres. Dengan pemetaan modul yang lebih terbatas dibanding admin, tampilan staff dibuat lebih fokus pada pekerjaan inti agar proses penanganan berjalan efisien.

Dashboard Admin

Dashboard *Admin* mencakup akses operasional sekaligus administratif. Selain memantau dan mengendalikan proses penanganan tiket, admin dapat mengelola akun, meninjau audit log untuk kebutuhan akuntabilitas, mengelola kategori komplain untuk menjaga konsistensi data, serta menggunakan statistik komplain untuk evaluasi kinerja penanganan. Modul notifikasi dan chat tetap tersedia untuk mendukung pemantauan komunikasi dalam sistem.

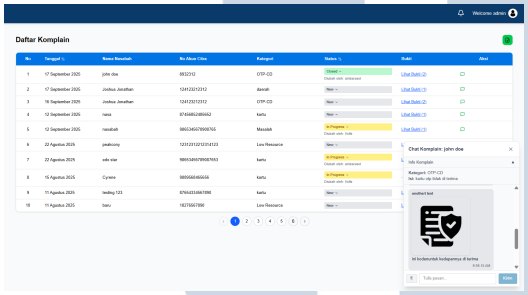
3.3.2.3 Fitur Chat dalam Tiket Komplain

Fitur chat merupakan sarana komunikasi langsung antara pengguna dan petugas (*Staff/Admin*) di dalam satu tiket komplain. Melalui chat, pengguna dapat memberikan klarifikasi, melengkapi informasi, maupun menanyakan progres penanganan tanpa perlu berpindah kanal komunikasi lain. Setiap tiket memiliki ruang chat tersendiri agar percakapan terdokumentasi dan konteks pembahasan tidak bercampur dengan tiket lain.

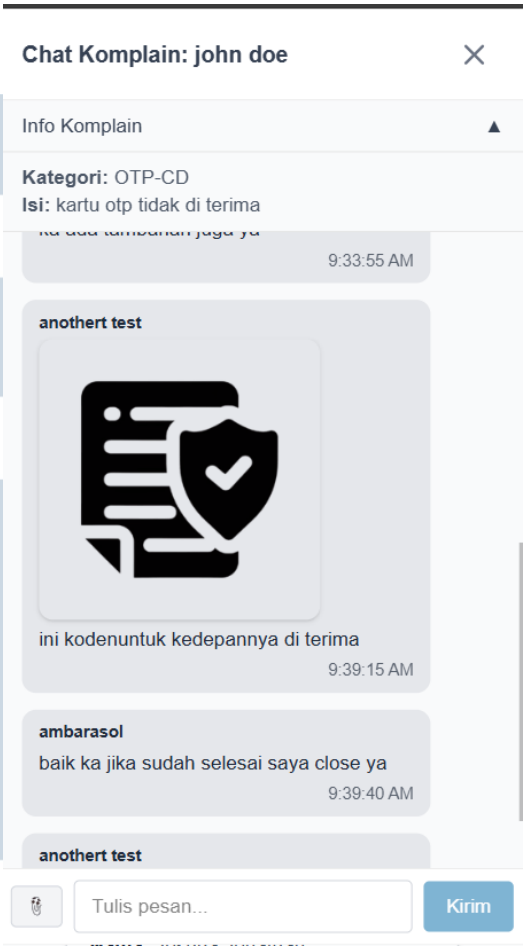
Pada implementasinya, chat dibuka dari daftar tiket pada dashboard. Sistem juga menampilkan indikator notifikasi (*badge*) pada ikon chat ketika terdapat pesan baru yang belum dibaca. Indikator tersebut akan hilang ketika pengguna membuka chat, baik dari daftar tiket maupun dari notifikasi.

Tampilan Chat (Desktop dan Mobile)

Antarmuka chat ditampilkan sebagai *popup* agar pengguna tetap berada pada konteks dashboard. Pada perangkat desktop, chat muncul sebagai panel *popup* dengan ukuran tetap. Pada perangkat mobile, chat ditampilkan memenuhi layar agar area percakapan dan input tetap nyaman digunakan.



Gambar 3.7. Tampilan chat pada perangkat desktop (popup/panel).



Gambar 3.8. Tampilan chat pada perangkat mobile (full screen).

Implementasi Frontend

Pada sisi frontend, chat dirancang sebagai komponen *popup* dengan header yang menampilkan judul percakapan dan tombol tutup. Area pesan menggunakan format *bubble chat* sehingga pesan pengguna dibedakan dari pesan petugas melalui posisi dan gaya tampilan. Setiap pesan memuat identitas pengirim dan waktu pengiriman agar kronologi percakapan jelas.

Chat mendukung pengiriman gambar. Pengguna dapat memilih file melalui ikon lampiran, melihat pratinjau sebelum mengirim, serta membatalkan lampiran jika diperlukan. Jika gambar pada pesan ditekan, sistem menampilkan ukuran lebih besar melalui *modal viewer*. Selain itu, panel *Info Komplain* disediakan dalam bentuk *collapsible* untuk menampilkan ringkasan tiket tanpa menutupi area chat secara permanen.

Implementasi Backend

Fitur chat tidak melalui endpoint API internal seperti modul login, melainkan memanfaatkan Firebase Realtime Database sebagai *Backend-as-a-Service* untuk komunikasi real-time. Pesan disimpan pada jalur per tiket (misalnya `komplainMessages/<idTiket>`). Pengambilan data dilakukan melalui *listener* berbasis event, sehingga pesan baru dapat tampil seketika tanpa mekanisme *refresh* manual.

Pengiriman pesan dilakukan menggunakan mekanisme *direct push* dari frontend ke RTDB. Operasi *push* secara otomatis menghasilkan *messageId* (key) unik untuk setiap pesan. Untuk kebutuhan notifikasi pesan baru, sistem menyimpan nilai *lastSeen* per pengguna dan per tiket pada jalur `usersLastSeenMessages/<uid>/<idTiket>` dalam bentuk *timestamp*. Nilai ini diperbarui ketika chat dibuka dari daftar tiket maupun dari notifikasi agar indikator *badge* langsung hilang.

Struktur Data

Struktur data pesan dan *lastSeen* pada Realtime Database dirangkum sebagai berikut.

```
komplainMessages/<ticketId>/<messageId> = {
  "senderId": "uid",
  "senderName": "Nama Pengirim",
  "text": "Isi pesan",
  "imageUrl": "https://... (opsional)",
  "timestamp": 1730000000000
}

usersLastSeenMessages/<uid>/<ticketId> = 1730000000000
```

Kode 3.1: Struktur data chat dan lastSeen pada Firebase Realtime Database.

Call dan Respons (RTDB)

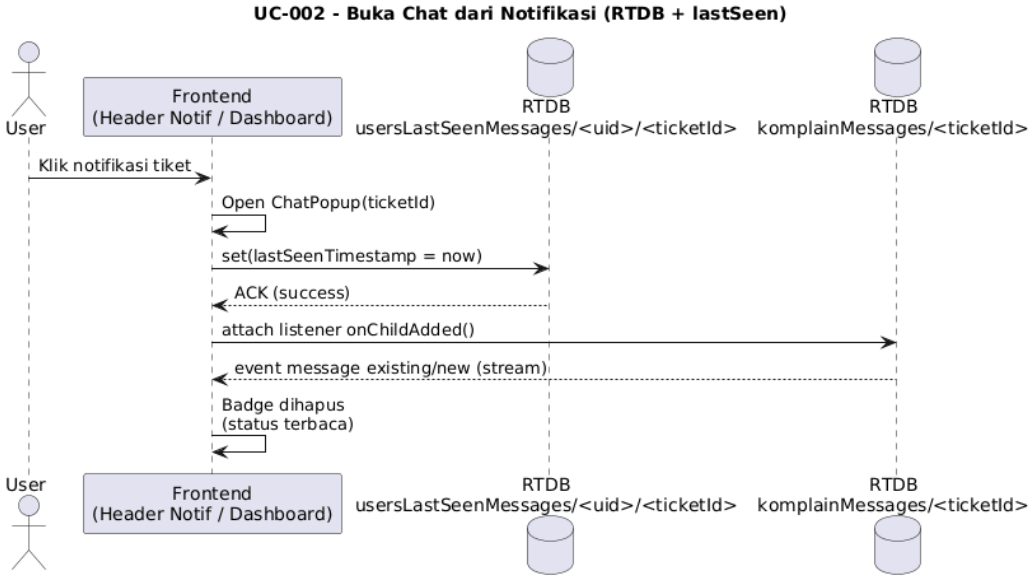
Karena chat terhubung langsung ke Firebase Realtime Database, bentuk respons tidak berupa JSON seperti endpoint API, melainkan berupa ACK dari operasi database (berhasil atau gagal) serta event real-time dari listener ketika terdapat pesan baru. Ringkasan call dan respons ditampilkan pada Tabel 3.7.

Tabel 3.7. Call dan respons pada fitur chat berbasis RTDB (direct push dan listener)

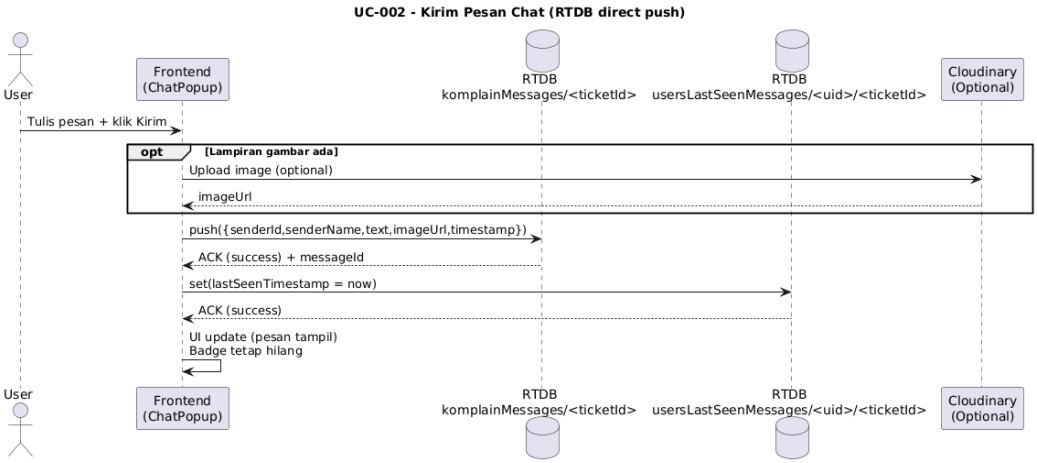
Aktivitas	**Call (Operasi)**	**Respons (ACK / Event)**
Membuka chat	*Attach listener* ke `komplainMessages/<idTiket>`	Event listener memuat pesan yang tersedia, kemudian mengirim event pesan baru ketika ada node baru.
Mengirim pesan teks (direct push)	`push` ke `komplainMessages/<idTiket>` dengan payload pesan	ACK sukses: pesan tersimpan dan menghasilkan *messageId*. ACK gagal: operasi ditolak (mis. *permission denied* atau gangguan jaringan).
Mengirim pesan dengan gambar	Upload ke Cloudinary lalu `push` message berisi `imageUrl`	ACK sukses: pesan tersimpan dengan URL gambar. ACK gagal: gagal upload atau gagal menulis ke RTDB.
Menandai sudah dibaca (*lastSeen*)	`set/update usersLastSeenMessages/<uid>/<idTiket>` (timestamp)	ACK sukses: nilai tersimpan dan indikator *badge* dihapus. ACK gagal: gagal tulis karena aturan akses atau jaringan.
Menerima pesan baru secara real-time	Event listener (mis. *onChildAdded*) pada jalur pesan	Event diterima dan UI menambahkan bubble pesan tanpa *refresh*.

Sequence Diagram

Alur proses utama pada fitur chat dirangkum dalam sequence diagram untuk memperjelas interaksi antar komponen sistem. Gambar 3.9 menggambarkan proses pembukaan chat dari notifikasi sekaligus pembaruan nilai *lastSeen*. Gambar 3.10 menggambarkan proses pengiriman pesan menggunakan mekanisme *direct push* ke Realtime Database.



Gambar 3.9. Sequence diagram pembukaan chat dari notifikasi dan pembaruan *lastSeen*.



Gambar 3.10. Sequence diagram pengiriman pesan chat menggunakan direct push ke RTDB.

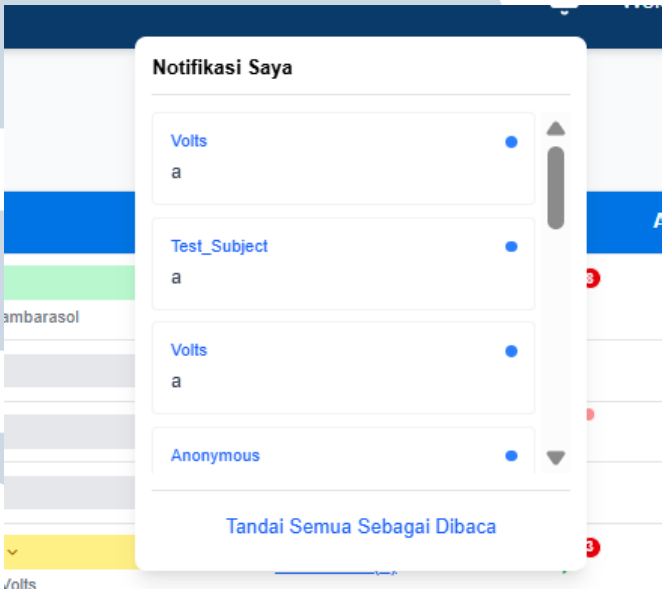
3.3.2.4 Notifikasi Pesan Chat Tiket Komplain

Fitur notifikasi digunakan untuk memberi sinyal ketika terdapat pesan chat baru pada tiket komplain. Indikator ini membantu pengguna memprioritaskan tiket yang membutuhkan respons tanpa harus membuka tiket satu per satu.

Antarmuka Notifikasi

Notifikasi ditampilkan melalui ikon lonceng pada *header*. Ketika terdapat pesan baru yang belum terbaca, ikon lonceng menampilkan badge sebagai indikator (Gambar 3.11). Saat ikon ditekan, sistem menampilkan dropdown "Notifikasi Saya" yang berisi daftar tiket dengan pesan baru, lengkap dengan cuplikan pesan terakhir dan penanda status belum dibaca (Gambar 3.12).

Gambar 3.11. Ikon lonceng dengan badge sebagai indikator pesan baru.



Gambar 3.12. Dropdown "Notifikasi Saya" yang menampilkan daftar pesan baru dan kontrol dibaca.

Implementasi Backend (Realtime Database)

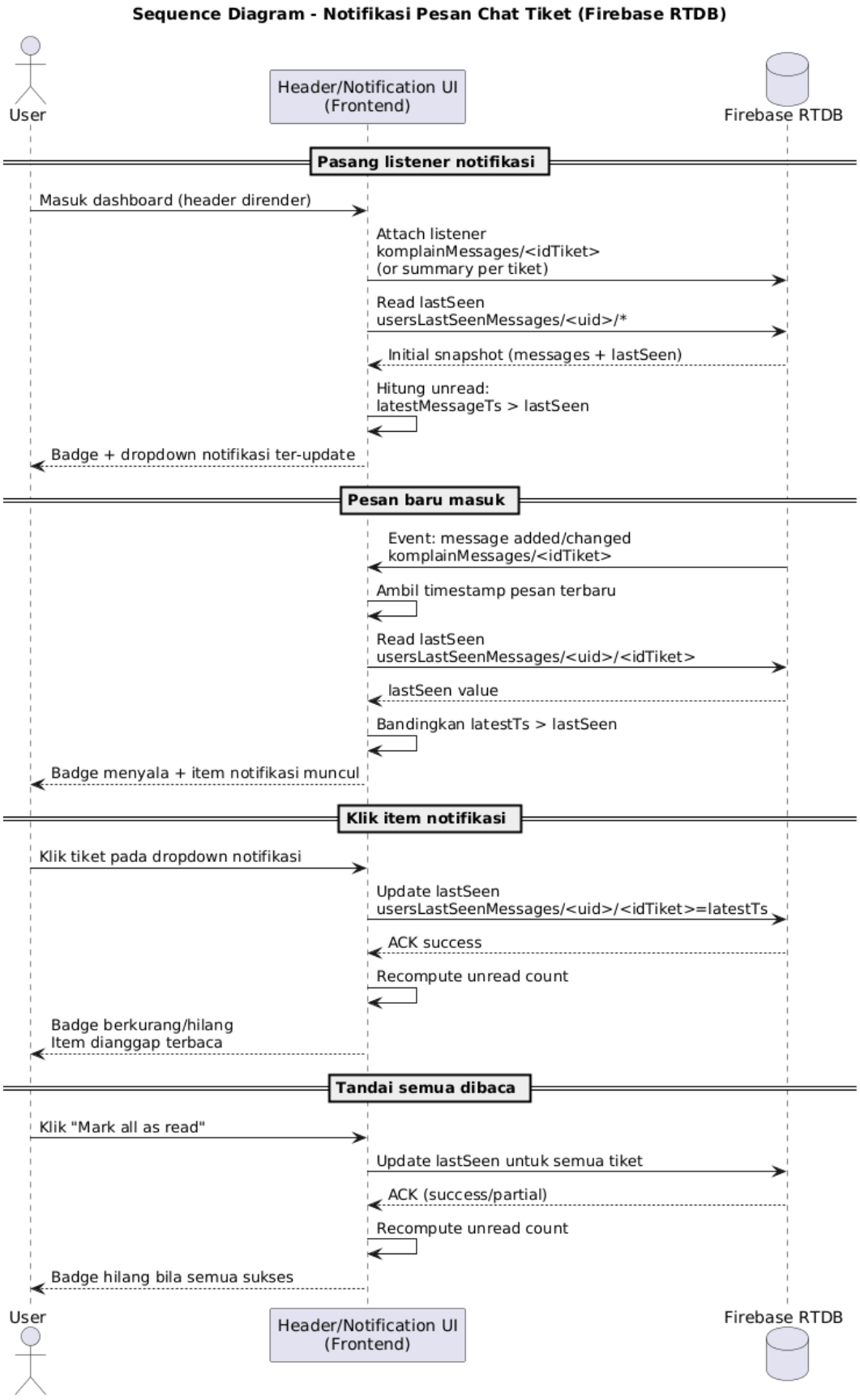
Notifikasi dibangun menggunakan Firebase Realtime Database. Secara konseptual, tiket dianggap memiliki pesan baru apabila *timestamp* pesan terbaru pada `komplainMessages/<idTiket>` lebih besar daripada nilai *lastSeen* pada `usersLastSeenMessages/<uid>/<idTiket>`. Ringkasan call dan respons yang terjadi pada notifikasi ditampilkan pada Tabel 3.8.

Tabel 3.8. Call dan respons fitur notifikasi chat berbasis RTDB

Aktivitas	**Call (Operasi)**	**Respons (ACK / Event)**
Memuat daftar notifikasi	*Attach listener* pada jalur pesan dan/atau ringkasan pesan per tiket	Event listener mengirim data real-time sehingga daftar notifikasi dan badge dapat diperbarui tanpa *refresh*.
Klik item notifikasi	Update *lastSeen* pada `usersLastSeenMessages/<uid>/<idTiket>`	ACK sukses: nilai *lastSeen* tersimpan dan item notifikasi dianggap terbaca. ACK gagal: pembaruan ditolak (aturan akses/jaringan) sehingga status belum berubah.
Tandai semua dibaca	Update *lastSeen* untuk seluruh tiket terkait notifikasi	ACK sukses: seluruh notifikasi dianggap terbaca dan badge lonceng hilang. ACK gagal: sebagian pembaruan gagal, sehingga beberapa item masih dianggap belum dibaca.

Sequence Diagram

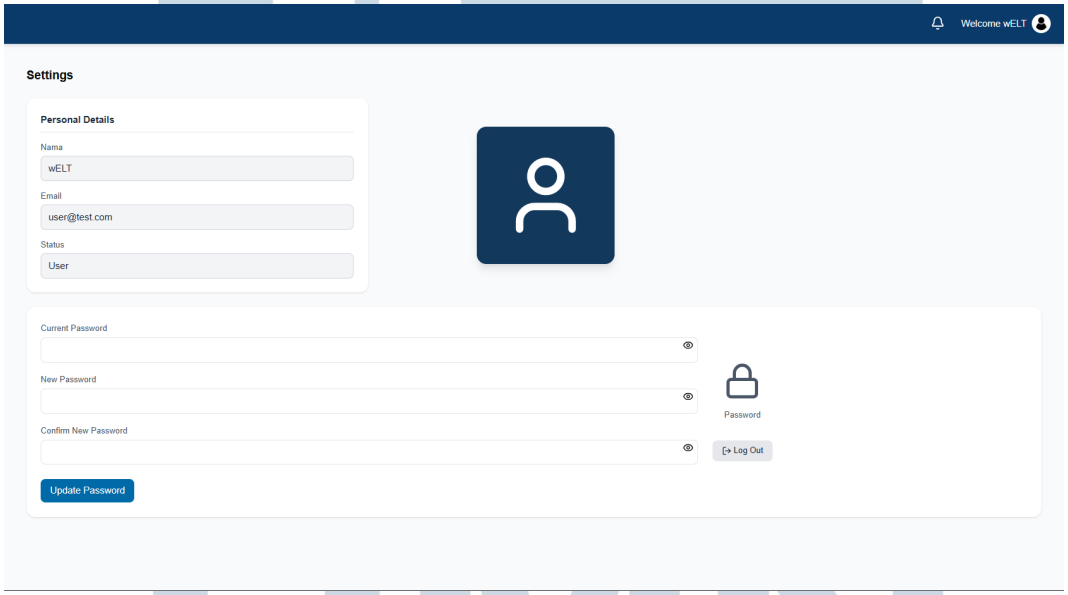
Untuk memperjelas mekanisme notifikasi berbasis event real-time, Gambar 3.13 menunjukkan alur ketika sistem memasang listener, menerima event pesan baru, menghitung kondisi belum dibaca berdasarkan *lastSeen*, lalu memperbarui badge dan daftar notifikasi. Diagram ini juga menunjukkan proses ketika pengguna membuka item notifikasi atau menandai semua notifikasi sebagai dibaca.



Gambar 3.13. Sequence diagram fitur notifikasi pesan chat tiket komplain berbasis Firebase RTDB

3.3.2.5 Halaman Pengaturan (Settings)

Halaman Pengaturan (*Settings*) merupakan modul umum yang dapat diakses oleh seluruh peran pengguna (*User*, *Staff*, dan *Admin*) setelah berhasil login. Modul ini berfungsi sebagai pusat pengelolaan informasi akun secara mandiri, yaitu menampilkan ringkasan identitas pengguna yang sedang aktif serta menyediakan fasilitas keamanan berupa perubahan kata sandi dan aksi *logout*. Dengan menyediakan fitur ini pada satu halaman terpusat, sistem membantu pengguna memahami status akun yang digunakan sekaligus mempermudah pemeliharaan keamanan akun tanpa perlu melibatkan pihak lain.



Gambar 3.14. Tampilan halaman Pengaturan (Settings) pada sistem.

Implementasi Frontend

Dari sisi *frontend*, halaman pengaturan dirancang dengan struktur yang sederhana dan informatif. Bagian *Personal Details* menampilkan data identitas utama seperti nama, email, dan status/peran akun dalam bentuk *read-only* sehingga pengguna dapat melakukan verifikasi informasi akun tanpa risiko perubahan tidak sengaja. Untuk memperkuat konteks visual, halaman menampilkan ikon yang menyesuaikan peran, misalnya ikon *shield* untuk akun *admin* dan ikon pengguna untuk peran lainnya .

Bagian keamanan akun disediakan melalui form perubahan kata sandi yang terdiri dari *Current Password*, *New Password*, dan *Confirm New Password*. Setiap

field kata sandi memiliki kontrol *show/hide* agar pengguna dapat memeriksa input bila diperlukan tanpa mengorbankan kenyamanan penggunaan. Setelah perubahan berhasil, sistem menampilkan pesan umpan balik dan mengosongkan field agar tidak menyisakan data sensitif pada tampilan. Selain itu, tersedia tombol *Log Out* untuk mengakhiri sesi penggunaan secara langsung dari halaman yang sama.

Implementasi Backend

Pada sisi *backend*, modul Settings diakses melalui API internal agar format layanan konsisten dengan modul login. Setiap request membawa cookie sesi `authToken` yang diverifikasi untuk memastikan pengguna sudah terautentikasi. Jika sesi valid, backend mengambil profil dari koleksi `users` berdasarkan *UID* dan mengembalikannya dalam format JSON. Untuk perubahan kata sandi, backend menerapkan *reauthentication* sebelum pembaruan password dilakukan, sehingga aksi sensitif hanya diproses ketika kredensial pengguna tervalidasi. Aksi *logout* mengakhiri sesi dengan menghapus cookie sesi dan mengembalikan respons JSON sebagai umpan balik kepada frontend.

Call dan Respons API

Ringkasan call dan respons untuk modul Settings ditampilkan pada Tabel 3.9 dan contoh request/response JSON ditampilkan pada Tabel 3.10.

Tabel 3.9. Ringkasan endpoint dan status response modul Settings

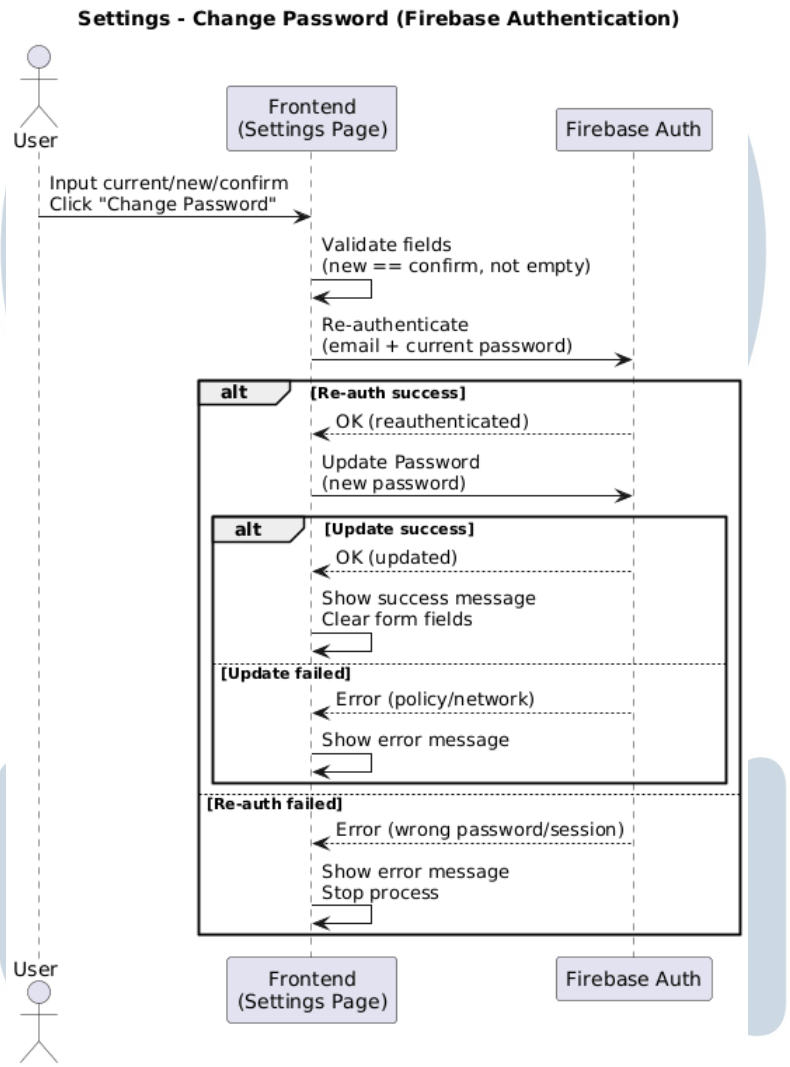
Fitur	**Endpoint**	**Status**	**Makna**
Muat profil	`GET /api/users/me`	200	Profil berhasil dimuat sesuai sesi login.
Muat profil	`GET /api/users/me`	401	Sesi tidak valid / tidak ada `authToken`.
Ubah password	`POST /api/auth/change-password`	200	Password berhasil diperbarui.
Ubah password	`POST /api/auth/change-password`	400	Validasi input gagal (field wajib/konfirmasi).
Ubah password	`POST /api/auth/change-password`	401	Sesi tidak valid / belum login.
Ubah password	`POST /api/auth/change-password`	403	Re-auth gagal (password saat ini salah).
Logout	`POST /api/auth/logout`	200	Logout berhasil dan sesi diakhiri.
Logout	`POST /api/auth/logout`	401	Sesi tidak valid / tidak ada `authToken`.

Tabel 3.10. Contoh request dan response JSON modul Settings

Skenario	**Contoh Request dan Respons**
Muat profil (200)	Request: GET /api/users/me Response: HTTP/1.1 200 OK Body: {"message":"Berhasil memuat profil","data":{"uid": "...","name":"...","email":"...","role":"..."}}
Ubah password (200)	Request: POST /api/auth/change-password Content-Type: application/json Body: {"currentPassword":"***","newPassword":"***","confirmPassword":"***"} Response: HTTP/1.1 200 OK Body: {"message":"Password berhasil diperbarui"}
Ubah password (400)	Request: POST /api/auth/change-password Response: HTTP/1.1 400 Bad Request Body: {"message":"Validasi input gagal","error":"VALIDATION_ERROR"}
Ubah password (403)	Request: POST /api/auth/change-password Response: HTTP/1.1 403 Forbidden Body: {"message":"Password saat ini tidak valid","error":"REAUTH_FAILED"}
Logout (200)	Request: POST /api/auth/logout Response: HTTP/1.1 200 OK Body: {"message":"Logout berhasil"}

Sequence Diagram Perubahan Password

Alur perubahan password dirangkum pada Gambar 3.15 untuk memperjelas tahapan validasi sesi, re-autentikasi, dan pembaruan kata sandi.



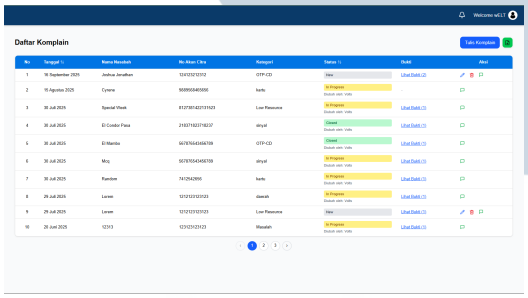
Gambar 3.15. Sequence diagram proses perubahan password pada modul Settings.

3.3.2.6 Lihat Daftar Tiket

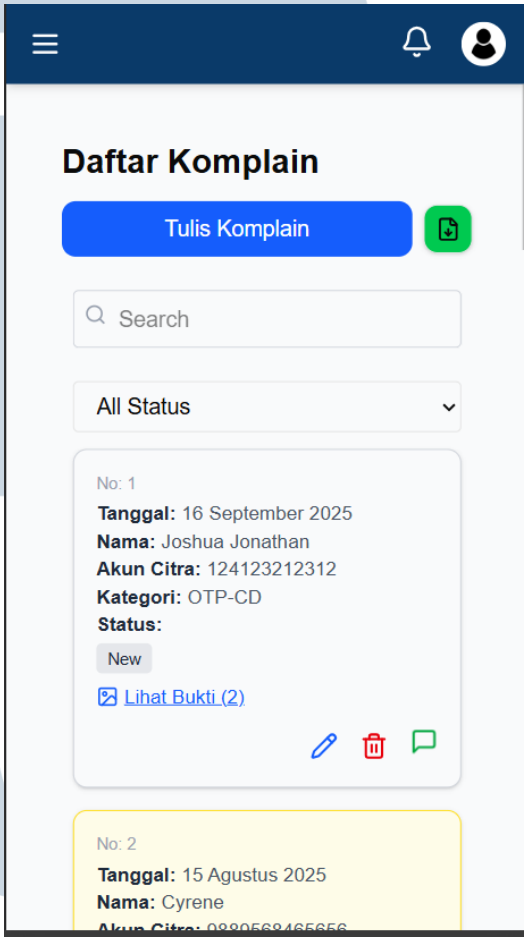
Modul *Lihat Daftar Tiket* berfungsi untuk menampilkan daftar tiket komplain milik pengguna yang sedang login. Melalui modul ini, pengguna dapat memantau status penanganan, membuka detail tiket, melihat bukti yang telah diunggah, serta melanjutkan komunikasi melalui fitur chat pada tiket terkait.

Selain itu, sistem memberikan kontrol terbatas pada tiket tertentu sesuai kebijakan (misalnya hanya pada status awal).

Untuk menjaga kenyamanan penggunaan di berbagai ukuran layar, antarmuka disusun secara responsif. Pada perangkat *desktop*, daftar tiket ditampilkan dalam bentuk tabel agar pengguna mudah melakukan pemindaian data, sedangkan pada perangkat *mobile*, daftar tiket ditampilkan dalam bentuk kartu agar lebih nyaman dibaca dan dioperasikan melalui sentuhan.



Gambar 3.16. Tampilan daftar tiket pada dashboard (desktop).



Gambar 3.17. Tampilan daftar tiket pada dashboard (mobile).

Implementasi Frontend

Pada sisi *frontend*, daftar tiket diterima sebagai *props* lalu dikelola kembali dalam *state* lokal agar UI dapat segera diperbarui ketika terdapat tiket baru atau perubahan status. Komponen menyediakan fungsi `addNewTicket` melalui *ref* sehingga ketika pengguna mengirim komplain baru, tiket dapat langsung disisipkan

ke urutan teratas dan halaman otomatis kembali ke halaman pertama.

Akses data pada tampilan user dibatasi melalui mekanisme *visibility filtering*, yaitu tiket yang ditampilkan hanyalah tiket yang memiliki `userId` sama dengan *UID* pengguna yang sedang login. Identitas pengguna didapatkan dari *Firebase Authentication*, sedangkan peran pengguna diambil dari dokumen `users/{uid}` dan dapat disimpan sementara pada *localStorage* untuk mengurangi proses pembacaan berulang.

Pada tampilan *desktop*, tiket ditampilkan dalam bentuk tabel yang mendukung *pagination* sehingga daftar tetap ringkas saat jumlah tiket bertambah. Tabel juga menyediakan kemampuan *sorting* untuk kolom tanggal dan status agar pengguna lebih mudah meninjau tiket berdasarkan urutan yang dibutuhkan. Pada tampilan *mobile*, daftar tiket ditampilkan dalam bentuk kartu, serta dilengkapi fitur pencarian dan filter (tanggal dan status) agar tiket dapat ditemukan lebih cepat pada layar kecil.

Setiap tiket menyediakan beberapa aksi utama yang relevan untuk user. Pertama, pengguna dapat membuka bukti komplain (jika ada) melalui tombol *Lihat Bukti* yang akan menampilkan gambar pada *modal viewer*. Kedua, pengguna dapat membuka chat tiket melalui tombol chat, dan sistem menampilkan indikator *badge* ketika terdapat pesan baru agar pengguna dapat merespons tanpa harus memeriksa satu per satu tiket. Untuk mencegah *double trigger*, pembukaan chat menggunakan mekanisme *deduplication guard* berbasis waktu sehingga klik cepat berulang tidak menyebabkan chat terbuka lebih dari sekali.

Selain itu, pengguna hanya diizinkan melakukan perubahan pada tiket dengan status awal. Pada implementasi ini, tombol *edit* dan *hapus* hanya muncul ketika status tiket masih `New`, sehingga perubahan tidak dilakukan ketika tiket sudah diproses. Jika pengguna memilih hapus, sistem menampilkan *dialog konfirmasi* terlebih dahulu untuk menghindari penghapusan tidak sengaja.

Implementasi Backend

Pada sisi backend, data tiket komplain disimpan di Cloud Firestore pada koleksi `komplain`. Untuk kebutuhan menampilkan daftar tiket, sistem menyediakan endpoint `GET /api/complaints` yang memverifikasi sesi melalui cookie `authToken`. Setelah sesi valid, backend memuat data tiket dan menerapkan pembatasan akses, yaitu untuk peran *User* hanya tiket milik pengguna (berdasarkan UID) yang dikembalikan. Parameter *pagination*, pencarian, dan filter juga diproses

pada backend agar hasil yang dikirim ke frontend tetap ringkas dan sesuai kebutuhan tampilan tabel/kartu.

Call dan Respons API

Rangkuman status dan makna respons endpoint `GET /api/complaints` ditampilkan pada Tabel 3.11. Contoh request dan respons JSON ditampilkan pada Tabel 3.12.

Tabel 3.11. Ringkasan status response endpoint `GET /api/complaints`

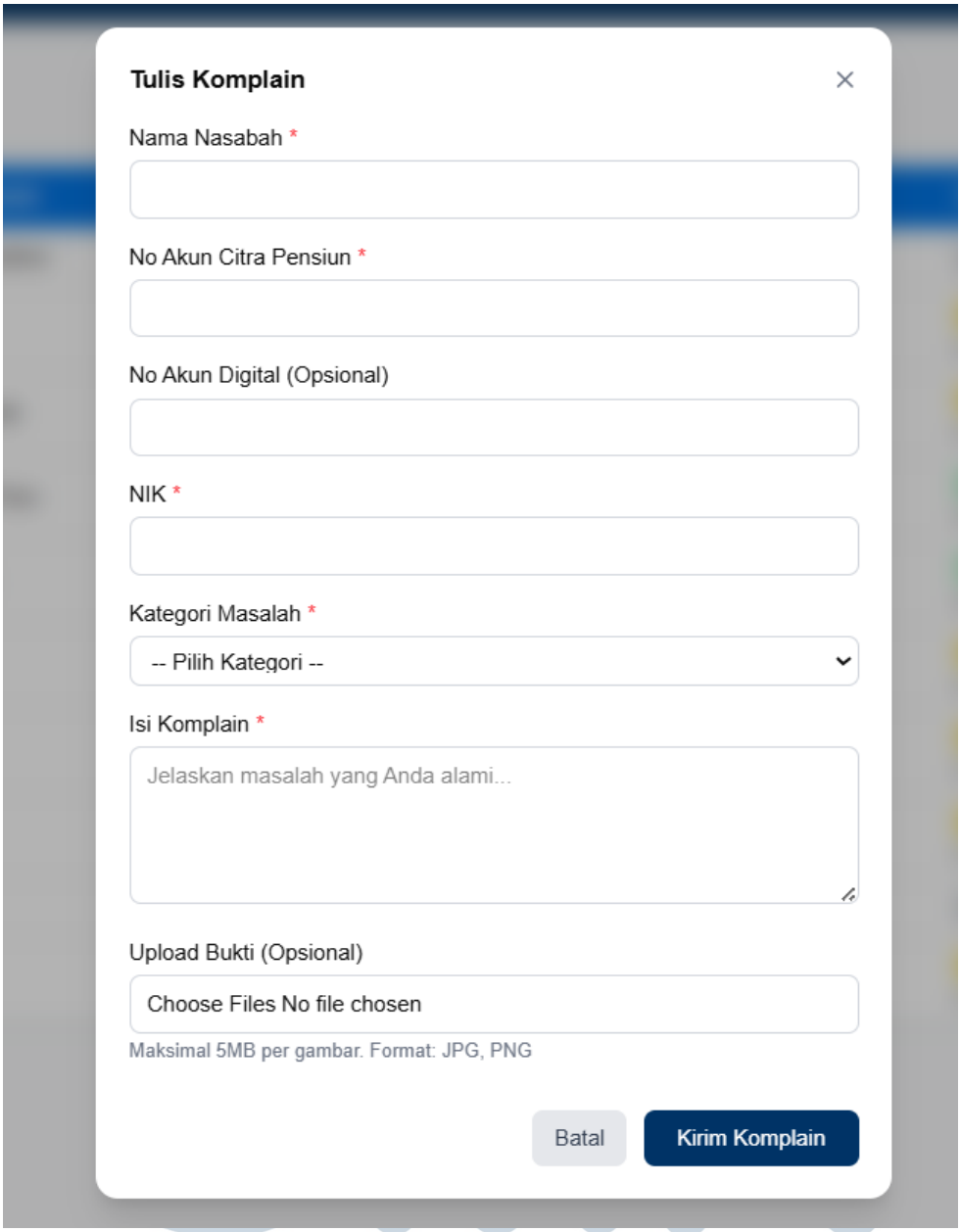
Kondisi	**Status HTTP**	**Makna**
Berhasil memuat daftar tiket	200	Data tiket dikembalikan sesuai hak akses dan parameter filter/pagination.
Parameter tidak valid	400	Request ditolak karena format *query parameter* tidak sesuai (misalnya `page` bukan angka).
Tidak terautentikasi	401	Cookie sesi tidak ada atau tidak valid sehingga akses ditolak.
Tidak berhak mengakses data	403	Pengguna terautentikasi tetapi tidak memiliki kewenangan pada konteks permintaan tertentu.
Kesalahan internal	500	Terjadi kegagalan proses server atau layanan data.

Tabel 3.12. Contoh request dan respons JSON endpoint `GET /api/complaints`

Skenario	**Contoh Request dan Respons**
Berhasil (200)	Request: GET /api/complaints?scope=mine&page=1&limit=10&status=New Respons: HTTP/1.1 200 OK Body: {"message":"Berhasil memuat tiket","data":{"items":[{"id":"...","title":"...","status":"New","createdAt":"..."}], "page":1,"limit":10}}
Tidak terautentikasi (401)	Request: GET /api/complaints?scope=mine Respons: HTTP/1.1 401 Unauthorized Body: {"message":"Unauthorized","error":"UNAUTHORIZED"}
Parameter tidak valid (400)	Request: GET /api/complaints?scope=mine&limit=abc Respons: HTTP/1.1 400 Bad Request Body: {"message":"Parameter tidak valid","error":"BAD_REQUEST"}

3.3.2.7 Modul Pembuatan Tiket

Modul pembuatan tiket merupakan fitur khusus pada *Dashboard User* yang digunakan untuk membuat tiket baru maupun memperbarui tiket yang sudah ada setelah pengguna berhasil login. Untuk menjaga fokus pengguna dan mengurangi perpindahan konteks, form pembuatan tiket ditampilkan dalam bentuk *modal dialog* (Gambar 3.18) dengan *overlay*, sehingga pengguna tetap berada pada halaman dashboard saat melakukan pengisian atau pembaruan data tiket.



Gambar 3.18. Tampilan form pembuatan tiket pada Dashboard User dalam bentuk modal.

Field yang disediakan pada form mencakup Nama Nasabah, Nomor Akun Citra Pensiun, Nomor Akun Digital, NIK, Kategori Masalah, dan Deskripsi Tiket. Selain itu, pengguna dapat melampirkan bukti pendukung dalam bentuk dokumen atau foto. Untuk memastikan kelengkapan data, beberapa field ditetapkan sebagai field wajib (misalnya nama, Nomor Akun Citra Pensiun, NIK, kategori, dan deskripsi tiket), sehingga tiket yang dikirim memiliki informasi minimum yang diperlukan untuk proses penanganan.

Implementasi Frontend

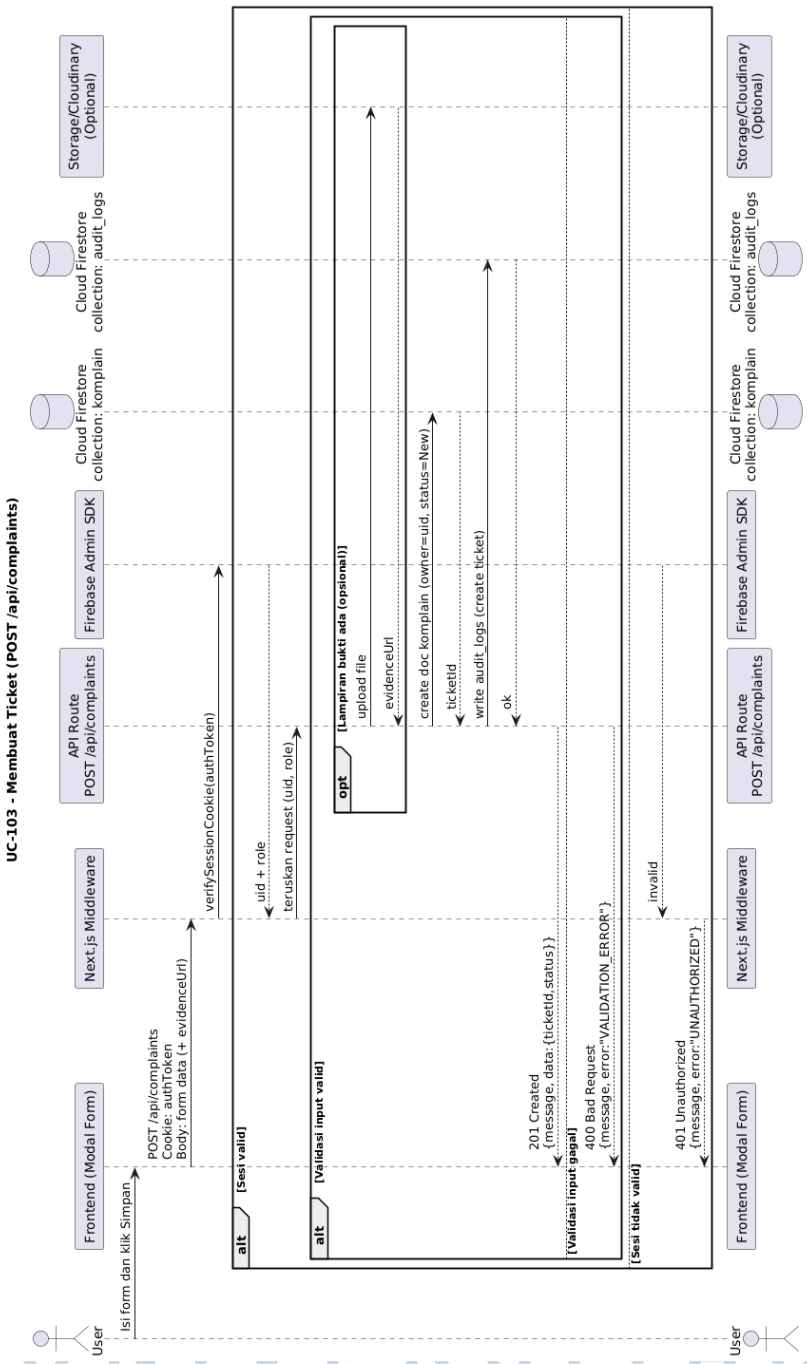
Pada sisi *frontend*, modul pembuatan tiket dibangun menggunakan React dan ditampilkan sebagai modal dengan *overlay* agar pengguna tetap berada pada konteks dashboard. Pengelolaan input form menggunakan mekanisme *form handling* sehingga proses validasi dapat dilakukan secara konsisten sebelum data dikirim atau diperbarui. Ketika pengguna belum mengisi field wajib atau format input tidak sesuai, sistem menampilkan pesan kesalahan secara langsung pada komponen input terkait untuk mengarahkan pengguna memperbaiki data.

Dropdown kategori dimuat secara dinamis dari basis data agar daftar kategori dapat dikelola tanpa *hardcode* pada antarmuka. Selain itu, kategori yang bersifat nonaktif tidak ditampilkan pada dropdown, sehingga pengguna hanya dapat memilih kategori yang masih berlaku dalam sistem. Pendekatan ini menjaga konsistensi antara pengaturan kategori pada sistem dan opsi yang muncul pada form tiket.

Antarmuka juga menerapkan kontrol interaksi berdasarkan status tiket dan hak akses. Sebagai contoh, apabila pengguna merupakan pemilik tiket dan tiket sudah berada pada status *In Progress*, maka beberapa field dan tombol simpan dinonaktifkan. Pembatasan ini bertujuan mencegah perubahan data inti ketika proses penanganan sudah berjalan, sehingga informasi yang sedang diproses oleh petugas tetap stabil.

Implementasi Backend

Alur pembuatan tiket dirangkum pada Gambar 3.19. Ketika pengguna menekan tombol simpan, frontend mengirim request `POST /api/complaints` beserta cookie sesi `authToken`. Middleware memverifikasi sesi menggunakan Firebase Admin SDK. Jika sesi valid, request diteruskan ke API route untuk dilakukan validasi input. Apabila pengguna melampirkan bukti, sistem mengunggah file ke Storage/Cloudinary dan menyimpan URL lampiran. Selanjutnya backend membuat dokumen tiket pada koleksi `komplain` dengan atribut *owner* dan status awal `New`. Setelah tiket berhasil dibuat, sistem dapat mencatat aktivitas ke koleksi `audit_logs`, lalu mengembalikan respons `201 Created`. Jika validasi input gagal, backend mengembalikan `400 Bad Request`, sedangkan sesi tidak valid menghasilkan `401 Unauthorized`.s.



Gambar 3.19. Sequence diagram proses pembuatan tiket komplain.

Call dan Respons API

Untuk mendukung form, backend menyediakan layanan: (1) memuat kategori aktif untuk dropdown, (2) membuat tiket baru, dan (3) memperbarui tiket pada kondisi tertentu. Ringkasan status respons untuk setiap endpoint disajikan

pada Tabel 3.15–3.17, sedangkan contoh request dan respons JSON disajikan pada Tabel 3.16–3.18.

Tabel 3.13. Ringkasan status response endpoint `GET /api/categories` (kategori aktif)

Kondisi	**Status HTTP**	**Makna**
Berhasil memuat kategori	200	Backend mengembalikan daftar kategori aktif untuk dropdown.
Tidak terautentikasi	401	Cookie sesi tidak ada/tidak valid sehingga akses ditolak.
Kesalahan internal	500	Terjadi kegagalan layanan data.

Tabel 3.14. Contoh request dan respons JSON endpoint `GET /api/categories`

Skenario	**Contoh Request dan Respons**

Berhasil (200)	`Request:`
	`GET /api/categories?active=true`
	
	`Respons:`
	`HTTP/1.1 200 OK`
	`Body:`
	`{"message":"Berhasil memuat kategori",`
	`"data":{"items":[{"id":"...","name":"..."}]}}`

Tidak terautentikasi (401)	`Request:`
	`GET /api/categories?active=true`
	
	`Respons:`
	`HTTP/1.1 401 Unauthorized`
	`Body:`
	`{"message":"Unauthorized","error":"UNAUTHORIZED"}`

Tabel 3.15. Ringkasan status response endpoint `POST /api/complaints` (buat tiket)

Kondisi	**Status HTTP**	**Makna**
Berhasil membuat tiket	201	Tiket berhasil dibuat dan backend mengembalikan *ticketId*.
Validasi input gagal	400	Field wajib tidak lengkap / format tidak sesuai.
Tidak terautentikasi	401	Cookie sesi tidak ada/tidak valid sehingga akses ditolak.
Tidak berhak (role tidak sesuai)	403	Peran pengguna tidak diizinkan membuat tiket.
Kesalahan internal	500	Terjadi kegagalan proses simpan atau layanan terkait.

Tabel 3.16. Contoh request dan respons JSON endpoint `POST /api/complaints`

Skenario	**Contoh Request dan Respons**
Berhasil (201)	Request: POST /api/complaints Content-Type: application/json Body: {"namaNasabah":"...","noAkunCitra":"...","noAkunDigital":" ...","nik":"...","categoryId":"...","description":"..."," evidenceUrl":"..."} Respons: HTTP/1.1 201 Created Body: {"message":"Tiket berhasil dibuat ","data":{" ticketId":"<DOC_ID>","status":"New"}}
Validasi gagal (400)	Request: POST /api/complaints Body: {"namaNasabah":"","nik":"","categoryId":null,"description":""} Respons: HTTP/1.1 400 Bad Request Body: {"message":"Field wajib belum lengkap","error":"VALIDATION_ERROR"}

Tabel 3.17. Ringkasan status response endpoint `PATCH /api/complaints/<id>` (update tiket)

Kondisi	**Status HTTP**	**Makna**
Berhasil memperbarui tiket	200	Tiket diperbarui dan backend mengembalikan data ringkas hasil update.
Validasi input gagal	400	Field tidak sesuai aturan (misalnya format NIK tidak valid).
Tidak terautentikasi	401	Cookie sesi tidak ada/tidak valid sehingga akses ditolak.
Bukan pemilik tiket	403	Pengguna bukan *owner* tiket yang diminta.
Tiket tidak dapat diubah (status tidak sesuai)	403	Tiket sudah diproses (misalnya *In Progress*) sehingga perubahan ditolak.
Tiket tidak ditemukan	404	*ticketId* tidak valid / dokumen tidak tersedia.
Kesalahan internal	500	Terjadi kegagalan proses simpan atau layanan terkait.

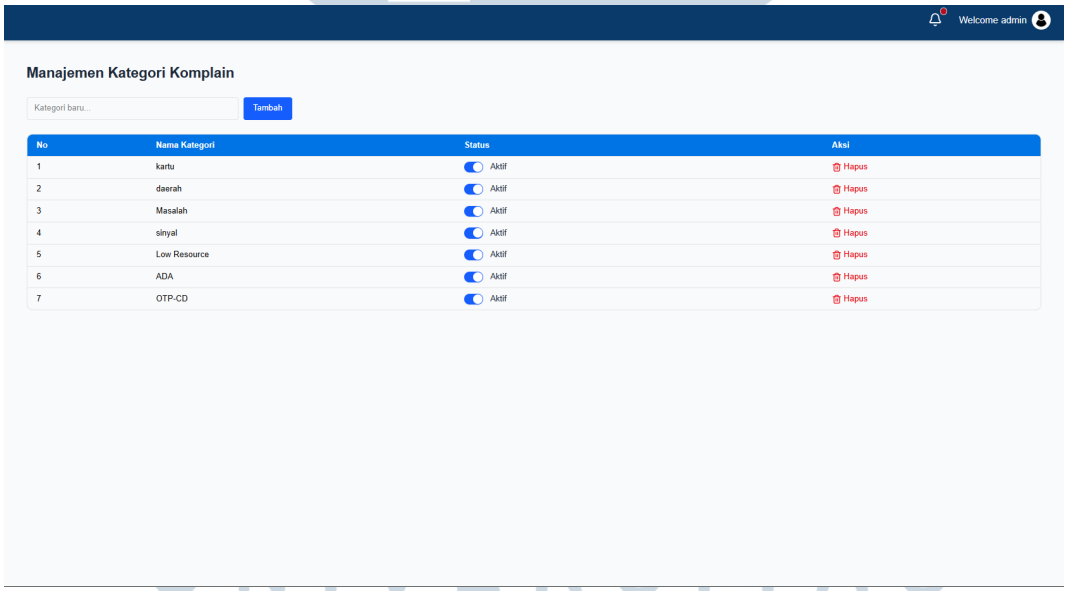
Tabel 3.18. Contoh request dan respons JSON endpoint `PATCH /api/complaints/<id>`

Skenario	**Contoh Request dan Respons**
Berhasil (200)	Request: PATCH /api/complaints/<DOC_ID> Body: {"description":"Update deskripsi...","evidenceUrl":"..."} Respons: HTTP/1.1 200 OK Body: {"message":"Tiket berhasil diperbarui"," data":{"ticketId":"<DOC_ID>","updatedAt":"..."}}
Ditolak karena status (403)	Request: PATCH /api/complaints/<DOC_ID> Respons: HTTP/1.1 403 Forbidden Body: {"message":"Tiket tidak dapat diubah pada status saat ini","error":"STATUS_LOCKED"}

3.3.2.8 Manajemen Kategori Komplain

Fitur Manajemen Kategori Komplain digunakan untuk mengelola daftar kategori masalah yang menjadi acuan pengelompokan tiket komplain. Kategori ini dipakai secara langsung pada saat pengguna membuat komplain, sehingga konsistensi kategori berpengaruh terhadap proses triase, pelaporan statistik, serta kemudahan pencarian data. Oleh karena itu, akses fitur ini dibatasi hanya untuk peran *Staff* dan *Admin*, sedangkan *User* tidak diberikan akses agar integritas data kategori tetap terjaga.

Secara umum, alur pengelolaan kategori dimulai ketika *Staff/Admin* membuka halaman kategori dan sistem memuat daftar kategori yang tersedia. Pengelola dapat menambahkan kategori baru melalui input yang disediakan, mengubah status kategori menjadi aktif atau nonaktif, serta menghapus kategori apabila diperlukan. Ketika suatu kategori dinonaktifkan, kategori tersebut tidak lagi ditampilkan sebagai opsi pada formulir pembuatan komplain, sehingga pengguna hanya dapat memilih kategori yang masih berlaku.



Gambar 3.20. Tampilan fitur manajemen kategori komplain.

Implementasi Frontend

Dari sisi *frontend*, halaman kategori dirancang sederhana dan langsung pada tujuan, yaitu menambahkan kategori dan mengelola daftar kategori yang sudah ada. Input penambahan kategori ditempatkan pada bagian atas agar pengelola dapat

menambah kategori baru tanpa berpindah halaman. Untuk menjaga validitas data, sistem melakukan validasi dasar dan menolak penambahan apabila nama kategori kosong.

Daftar kategori ditampilkan dalam bentuk tabel pada perangkat *desktop* agar informasi lebih mudah dibaca secara bersamaan. Pada perangkat *mobile*, data dapat disajikan lebih ringkas agar tetap nyaman dioperasikan melalui sentuhan. Setiap kategori memiliki aksi inti berupa pengaturan status aktif/nonaktif melalui kontrol *toggle* dan penghapusan kategori melalui tombol hapus yang dilindungi dialog konfirmasi.

Pendekatan status aktif/nonaktif digunakan sebagai kontrol operasional agar kategori yang tidak lagi relevan dapat dinonaktifkan tanpa menghilangkan jejak data. Dengan demikian, kategori yang dinonaktifkan tidak digunakan lagi pada proses input komplain, namun data historis tetap konsisten.

Implementasi Backend

Pada sisi *backend*, data kategori disimpan pada koleksi `kategori_komplain` di Cloud Firestore. Saat halaman dibuka, backend mengambil seluruh dokumen kategori dan mengembalikannya ke *frontend* untuk ditampilkan. Setiap data kategori minimal memuat nama kategori, status aktif/nonaktif, serta informasi waktu pembuatan untuk menjaga keterlacakan perubahan data.

Saat kategori ditambahkan, backend membuat dokumen baru dengan status awal aktif. Perubahan status kategori diproses melalui operasi pembaruan (*update*) pada field *isActive* sehingga perubahan dapat langsung mempengaruhi seluruh bagian sistem yang menggunakan kategori, khususnya pilihan kategori pada form pembuatan komplain. Penghapusan kategori diproses melalui operasi *delete* setelah pengelola melakukan konfirmasi pada antarmuka.

Dari sisi keamanan, backend menerapkan pembatasan akses berbasis peran. Akses halaman dan menu dibatasi pada lapisan aplikasi, lalu diperkuat kembali pada lapisan basis data melalui *Security Rules* agar hanya *Staff* dan *Admin* yang dapat melakukan operasi tulis (*create*, *update*, *delete*) pada koleksi kategori. Jika terjadi kegagalan proses, backend mengembalikan respons galat yang jelas agar *frontend* dapat menampilkan umpan balik yang informatif.

Rangkuman status dan makna respons API kategori ditampilkan pada Tabel 3.19. Contoh request dan respons ditampilkan pada Tabel 3.20.

Tabel 3.19. Ringkasan Status dan Makna Respons API Manajemen Kategori Komplain

Kondisi	**Status HTTP**	**Makna dan Dampak**
Request berhasil	200	Backend berhasil memuat/menambah/mengubah/menghapus kategori dan *frontend* dapat memperbarui tampilan.
Input tidak valid	400	Nama kategori kosong atau format request tidak sesuai sehingga proses dibatalkan sebelum menulis ke basis data.
Belum login / sesi tidak valid	401	Permintaan ditolak karena pengguna belum terautentikasi.
Role tidak sesuai (bukan Staff/Admin)	403	Permintaan ditolak karena pengguna tidak memiliki hak akses pengelolaan kategori.
Kategori tidak ditemukan	404	Target kategori yang dimaksud tidak ditemukan sehingga operasi tidak dapat dilanjutkan.
Gagal eksekusi di server	500	Terjadi kegagalan proses di server atau basis data sehingga aksi tidak terselesaikan.

Tabel 3.20. Contoh Request dan Respons API Manajemen Kategori Komplain

Skenario	**Contoh Request dan Respons**

Memuat kategori (200)	`Request:`
	`GET /api/kategori-komplain`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"data":[{"id":"...","name":"Layanan","isActive":true},...]}`

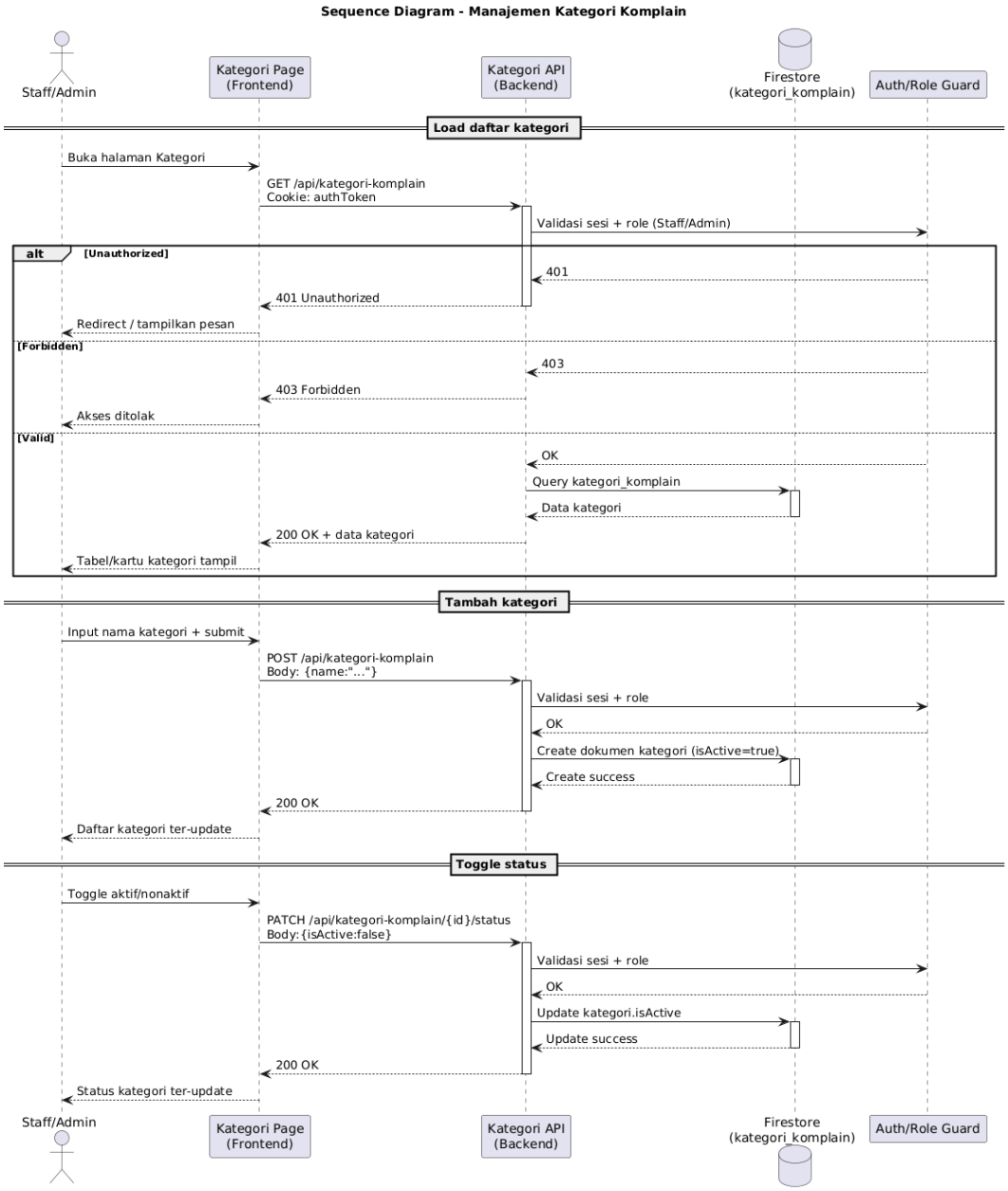
Tambah kategori (200)	`Request:`
	`POST /api/kategori-komplain`
	`Body: {"name":"Fasilitas"}`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"message":"Kategori created"}`

Toggle aktif/nonaktif (200)	`Request:`
	`PATCH /api/kategori-komplain/{id}/status`
	`Body: {"isActive":false}`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"message":"Status updated"}`

Hapus kategori (200)	`Request:`
	`DELETE /api/kategori-komplain/{id}`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"message":"Kategori deleted"}`

Sequence Diagram

Untuk memperjelas interaksi antara *Staff/Admin*, *frontend*, dan *backend*, Gambar 3.21 menunjukkan alur ketika sistem memuat daftar kategori serta menjalankan aksi inti seperti menambah kategori dan mengubah status aktif/nonaktif.



Gambar 3.21. Sequence diagram proses Manajemen Kategori Komplain

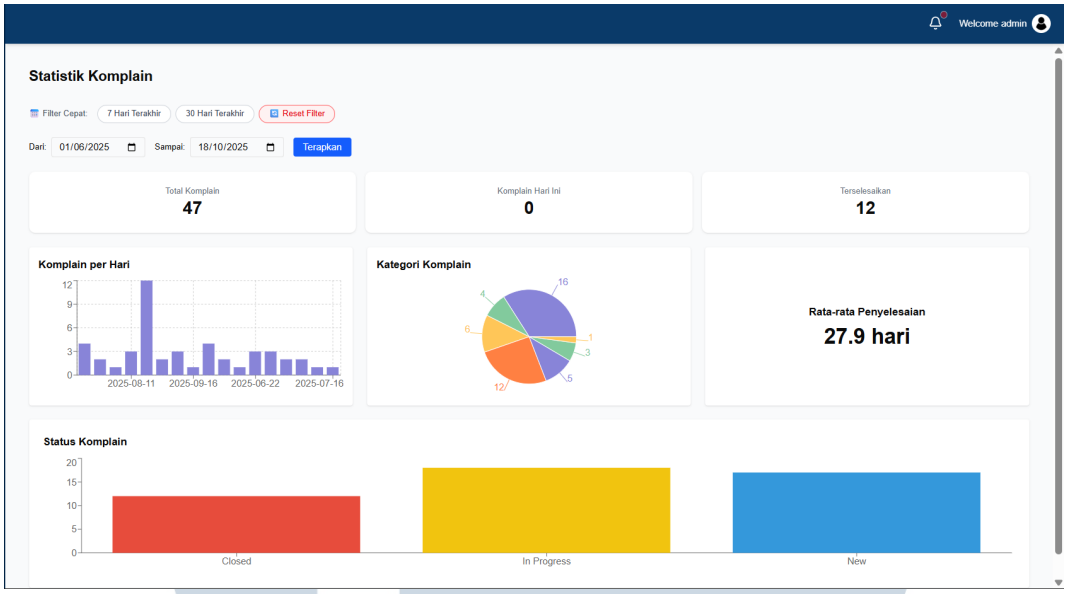
3.3.2.9 Modul Statistik Komplain

Modul Statistik Komplain berfungsi untuk membantu *Staff* dan *Admin* memantau kondisi penanganan komplain secara ringkas dan terukur melalui ringkasan metrik dan visualisasi data. Modul ini dirancang agar pengelola dapat melakukan pemantauan harian serta evaluasi periodik, misalnya dengan melihat total komplain pada periode tertentu, jumlah komplain yang masuk pada hari berjalan, jumlah komplain yang terselesaikan, distribusi komplain berdasarkan kategori, distribusi komplain berdasarkan status, serta rata-rata durasi penyelesaian.

Secara umum, alur statistik dimulai ketika pengguna membuka halaman statistik dan memilih rentang tanggal yang ingin dianalisis. Pengguna dapat memakai *filter cepat* (misalnya 7 hari terakhir atau 30 hari terakhir) maupun menentukan tanggal secara manual. Sistem kemudian meminta data statistik ke backend berdasarkan parameter tanggal tersebut. Backend memvalidasi autentikasi dan otorisasi, mengambil data komplain sesuai rentang, menghitung agregasi yang dibutuhkan, lalu mengembalikan hasil dalam format JSON. Selanjutnya, *frontend* merender hasil tersebut menjadi kartu ringkasan dan grafik. Apabila terjadi kegagalan pengambilan data atau rentang tanggal tidak valid, sistem menampilkan pesan galat yang informatif agar pengguna dapat melakukan perbaikan.

Implementasi Frontend

Dari sisi *frontend*, Modul Statistik Komplain diimplementasikan sebagai satu halaman khusus pada dashboard, yaitu halaman *Statistik Komplain*. Halaman ini dirancang sebagai dashboard analitik yang menggabungkan kontrol filter, ringkasan metrik, dan grafik dalam satu tampilan agar pengguna dapat memahami kondisi secara cepat tanpa perlu berpindah halaman.



Gambar 3.22. Tampilan modul Statistik Komplain dengan filter tanggal, ringkasan, dan grafik.

Pada bagian atas halaman, tersedia kontrol filter yang terdiri dari *filter cepat* (7 hari terakhir dan 30 hari terakhir), filter rentang tanggal manual (Dari–Sampai), serta tombol *reset* untuk mengembalikan filter ke kondisi awal. Ketika filter diubah, *frontend* melakukan pemanggilan endpoint statistik dengan menyertakan parameter tanggal awal dan tanggal akhir, kemudian memperbarui tampilan berdasarkan respons yang diterima.

Hasil statistik ditampilkan dalam dua bentuk utama. Bagian pertama adalah kartu ringkasan (*summary cards*) yang memprioritaskan metrik inti seperti Total Komplain, Komplain Hari Ini, Terselesaikan, serta rata-rata durasi penyelesaian. Bagian kedua adalah visualisasi grafik yang menampilkan tren komplain per hari, distribusi kategori, dan distribusi status. Grafik dilengkapi *tooltip* agar pengguna dapat melihat nilai detail tanpa membuat tampilan terlalu padat.

Untuk menjaga pengalaman pengguna, halaman menampilkan indikator *loading* ketika data sedang dimuat dan menampilkan pesan error yang informatif ketika pemanggilan API gagal. Tampilan juga disusun responsif sehingga pada layar lebar komponen dapat ditata dalam grid, sedangkan pada layar sempit komponen disusun vertikal agar tetap mudah dibaca dan dioperasikan.

Implementasi Backend

Pada sisi *backend*, data statistik disediakan melalui endpoint `GET /api/statistik`. Endpoint ini hanya dapat diakses oleh peran *Admin* dan *Staff* sehingga data analitik yang bersifat operasional tidak dapat dilihat oleh pengguna biasa. Backend memvalidasi autentikasi berdasarkan cookie sesi `authToken`, kemudian memeriksa peran pengguna sebelum memproses permintaan.

Request statistik menerima parameter `start` dan `end` dalam format tanggal *YYYY-MM-DD*. Backend memvalidasi parameter agar formatnya benar dan memastikan rentang tanggal konsisten. Setelah validasi berhasil, backend mengambil data komplain sesuai rentang tanggal dan menghitung agregasi untuk kebutuhan tampilan, yaitu total komplain, komplain pada hari berjalan, jumlah terselesaikan, tren komplain per tanggal, distribusi kategori, distribusi status, serta rata-rata durasi penyelesaian untuk komplain yang benar-benar selesai. Hasil perhitungan kemudian dikirim dalam format JSON agar dapat langsung digunakan oleh komponen visualisasi pada *frontend*.

Jika pengguna belum login, backend mengembalikan *Unauthorized* (401). Jika pengguna login tetapi role tidak sesuai, backend mengembalikan *Forbidden* (403). Jika parameter tanggal tidak valid, backend mengembalikan *Bad Request* (400). Jika terjadi kegagalan query atau perhitungan agregasi, backend mengembalikan *Internal Server Error* (500). Ringkasan status dan makna respons API statistik ditampilkan pada Tabel 3.21. Contoh request dan respons yang digunakan pada masing-masing skenario ditampilkan pada Tabel 3.22.

Tabel 3.21. Ringkasan Status dan Makna Respons API Statistik Komplain

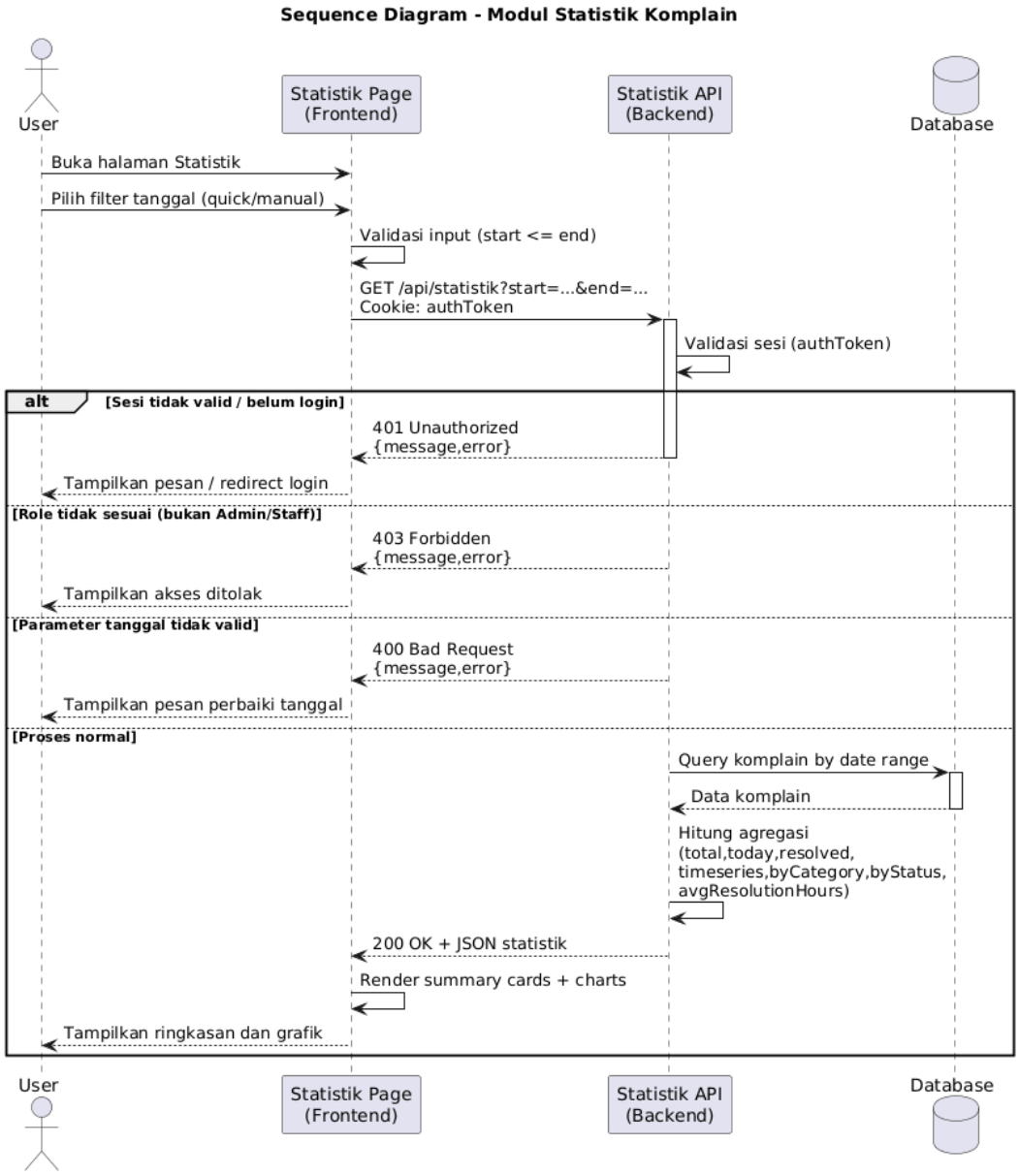
Kondisi	**Status HTTP**	**Makna dan Dampak**
Request statistik berhasil	200	Backend mengembalikan ringkasan dan dataset grafik sesuai rentang tanggal agar dapat langsung dirender oleh *frontend*.
Parameter `start`/`end` tidak valid	400	Format tanggal salah atau rentang tidak konsisten sehingga backend menghentikan proses sebelum agregasi dilakukan.
Belum login / sesi tidak valid	401	Backend menolak akses karena pengguna belum terautentikasi.
Role tidak sesuai (*bukan Admin/Staff*)	403	Backend menolak akses meskipun pengguna login karena tidak memiliki hak akses analitik.
Gagal query / gagal agregasi	500	Terjadi kegagalan proses di server sehingga *frontend* tidak dapat menampilkan statistik.

Tabel 3.22. Contoh Request dan Respons API Statistik Komplain

Skenario	**Contoh Request dan Respons**
Request berhasil (200)	Request: GET /api/statistik?start=2025-10-01 &end=2025-10-30 Response: HTTP/1.1 200 OK Body: {"meta":{"start":"2025-10-01","end":"2025-10-30"}, "summary":{"total":120,"today":6,"resolved":70, "avgResolutionHours":15.4}, "timeseries":[{"date":"2025-10-01","total":3}, {"date":"2025-10-02","total":5}], "byCategory":[{"category":"Layanan","total":40}, {"category":"Fasilitas","total":30}], "byStatus":[{"status":"New","total":20}, {"status":"In Progress","total":30}, {"status":"Done","total":70}]}
Tanggal tidak valid / rentang tidak konsisten (400)	Request: GET /api/statistik?start=2025-10-30 &end=2025-10-01 Response: HTTP/1.1 400 Bad Request Body: {"message":"Invalid date range","error":"BAD_REQUEST"}
Belum login / sesi tidak valid (401)	Request: GET /api/statistik?start=2025-10-01 &end=2025-10-30 Response: HTTP/1.1 401 Unauthorized Body: {"message":"Unauthorized","error":"UNAUTHORIZED"}
Role tidak sesuai (403)	Request: GET /api/statistik?start=2025-10-01 &end=2025-10-30 Response: HTTP/1.1 403 Forbidden Body: {"message":"Forbidden","error":"FORBIDDEN"}

Sequence Diagram

Untuk memperjelas interaksi antara pengguna, *frontend*, dan *backend*, Gambar 3.23 menunjukkan alur ketika pengguna menerapkan filter, sistem melakukan request statistik, backend menghitung agregasi, lalu *frontend* merender ringkasan dan grafik berdasarkan respons.



Gambar 3.23. Sequence diagram proses Statistik Komplain

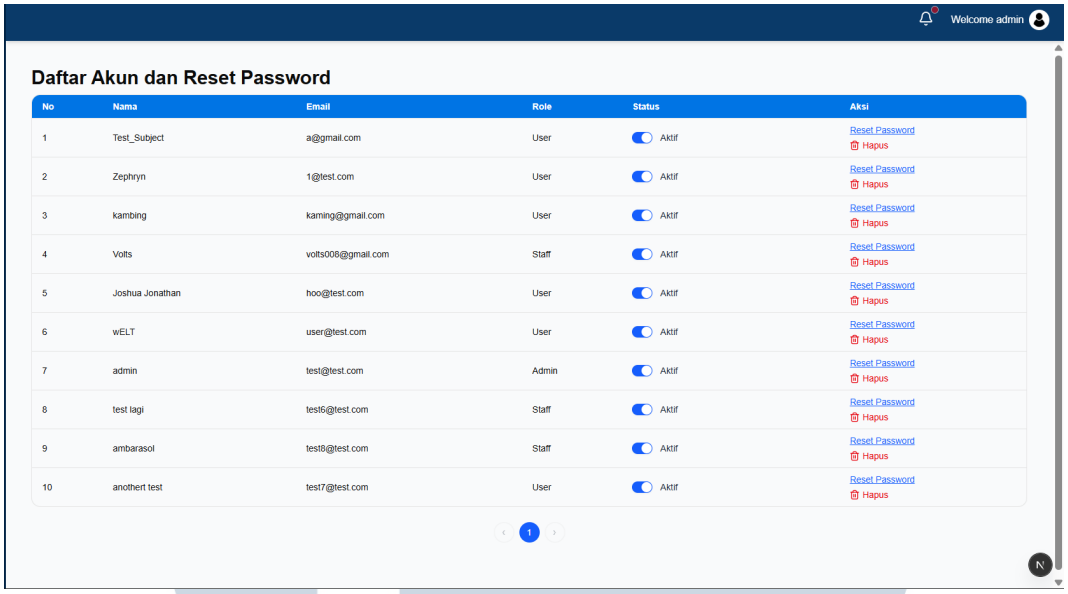
3.3.2.10 Manajemen Akun (Khusus Admin)

Modul Manajemen Akun merupakan fitur administratif yang hanya dapat diakses oleh *Admin* untuk mengelola akun pengguna dalam sistem. Modul ini disediakan untuk kebutuhan operasional dan keamanan, seperti meninjau daftar akun, mengubah status aktif/nonaktif, melakukan *reset password* ketika pengguna mengalami kendala akses, serta menghapus akun yang tidak lagi digunakan. Selain itu, admin juga dapat membuat akun baru (misalnya untuk *Staff*) melalui halaman khusus, sehingga proses onboarding dapat dilakukan secara terkontrol tanpa membuka akses pembuatan akun bagi peran lain.

Secara umum, alur manajemen akun dimulai ketika admin membuka halaman daftar akun. Sistem memuat daftar pengguna secara terotorisasi, lalu admin dapat menjalankan aksi administratif pada akun tertentu, seperti mengubah status aktif/nonaktif, melakukan *reset password*, atau menghapus akun. Jika admin memilih membuat akun baru, admin diarahkan ke halaman pembuatan akun, mengisi informasi akun, lalu sistem menyimpan akun baru sehingga langsung tersedia pada daftar akun. Apabila sesi tidak valid atau role tidak sesuai, sistem menolak akses untuk memastikan fitur sensitif ini hanya dapat digunakan oleh admin.

Implementasi Frontend

Dari sisi *frontend*, modul Manajemen Akun disusun menjadi dua halaman utama, yaitu halaman daftar akun dan halaman pembuatan akun baru. Pemisahan ini dilakukan agar aktivitas pemantauan dan aksi cepat (misalnya toggle status, reset password, hapus akun) tetap berada pada satu halaman, sementara proses pembuatan akun memiliki alur input yang lebih terfokus.

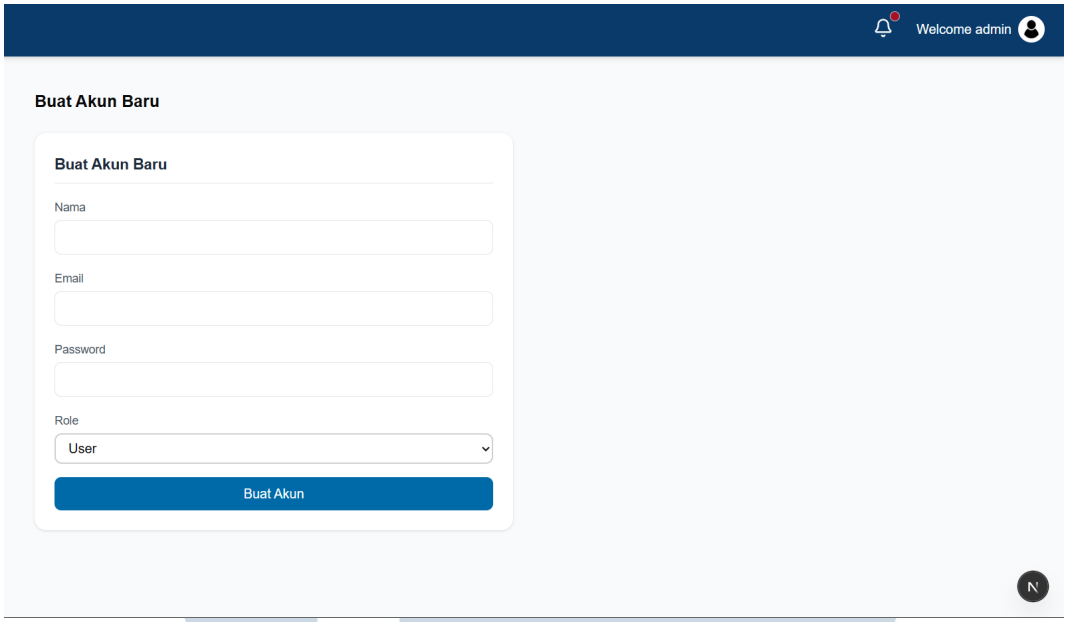


Gambar 3.24. Halaman daftar akun, pengaturan status, serta aksi reset password dan hapus akun.

Pada halaman daftar akun, data pengguna ditampilkan dalam bentuk tabel agar admin dapat melihat informasi inti secara cepat, seperti nama, email, role, status, serta aksi yang tersedia. Untuk menjaga kenyamanan penggunaan pada berbagai perangkat, tampilan dibuat responsif dengan menyesuaikan penyajian data pada perangkat *mobile* agar tetap terbaca dan mudah dioperasikan.

Aksi administrasi dirancang eksplisit dan aman. Perubahan status akun dilakukan melalui kontrol *toggle* (aktif/nonaktif) sehingga admin dapat menonaktifkan akun tanpa menghapus data. Aksi *reset password* disediakan melalui *modal* agar admin dapat menetapkan kata sandi baru tanpa berpindah halaman; pada *modal* ini dapat disediakan bantuan seperti *generate password* serta indikator kekuatan password untuk mendorong penerapan kata sandi yang lebih kuat. Aksi hapus akun dilindungi dengan *dialog konfirmasi* untuk mencegah penghapusan tidak sengaja.

Untuk menjaga keterbacaan data ketika jumlah pengguna bertambah, halaman daftar akun dilengkapi pagination sehingga admin dapat menavigasi data per halaman secara konsisten tanpa membebani tampilan dengan daftar yang terlalu panjang.



Gambar 3.25. Halaman pembuatan akun baru oleh admin.

Pada halaman pembuatan akun, admin mengisi informasi yang diperlukan untuk akun baru, kemudian sistem memvalidasi input sebelum menyimpan akun. Setelah berhasil dibuat, akun baru akan muncul pada daftar akun sehingga admin dapat langsung melakukan pengelolaan lanjutan jika diperlukan.

Implementasi Backend

Pada sisi *backend*, modul Manajemen Akun menerapkan kontrol akses ketat berbasis peran (*role-based access control*). Seluruh endpoint admin hanya dapat dipanggil apabila pengguna sudah login dan memiliki peran *Admin*. Jika sesi tidak valid, backend mengembalikan respons *Unauthorized* (401). Jika pengguna login tetapi role bukan admin, backend mengembalikan respons *Forbidden* (403). Dengan pendekatan ini, fitur sensitif seperti reset password, pembuatan akun, dan penghapusan akun tidak dapat diakses oleh *User* maupun *Staff*.

Secara fungsional, backend menyediakan layanan untuk memuat daftar akun secara terotorisasi, menerapkan perubahan status aktif/nonaktif, memproses *reset password*, menghapus akun, serta membuat akun baru. Pemuatan daftar akun dilakukan secara *server-side* agar data pengguna dapat dibaca sesuai kewenangan admin. Setiap aksi administratif diproses melalui endpoint khusus admin sehingga validasi autentikasi, otorisasi, serta validasi input dapat diterapkan sebelum eksekusi dilakukan. Jika proses berhasil, backend mengembalikan respons JSON

sebagai indikator keberhasilan. Jika gagal, backend mengembalikan respons galat yang jelas, sementara detail error dicatat pada sisi server untuk membantu debugging.

Rangkuman status dan makna respons API manajemen akun ditampilkan pada Tabel 3.23. Contoh request dan respons API ditampilkan pada Tabel 3.24.

Tabel 3.23. Ringkasan Status dan Makna Respons API Manajemen Akun (Admin)

Kondisi	**Status HTTP**	**Makna dan Dampak**
Request berhasil	200	Backend mengeksekusi aksi admin (load akun, toggle status, reset password, hapus akun, atau buat akun) dan mengembalikan indikator keberhasilan.
Input tidak valid	400	Request tidak memenuhi validasi (misalnya field wajib kosong atau format tidak sesuai) sehingga proses dihentikan.
Belum login / sesi tidak valid	401	Backend menolak akses karena pengguna belum terautentikasi.
Role bukan admin	403	Backend menolak akses karena pengguna tidak memiliki hak akses untuk aksi admin.
Akun tidak ditemukan	404	Target akun yang dimaksud tidak ditemukan sehingga aksi tidak dapat dijalankan.
Gagal eksekusi di server	500	Terjadi kegagalan proses di server sehingga aksi admin tidak dapat diselesaikan.

Tabel 3.24. Contoh Request dan Respons API Manajemen Akun (Admin)

Skenario	**Contoh Request dan Respons**

Memuat daftar akun (200)	`Request:`
	`GET /api/admin/users?page=1`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"data":[...], "page":1, "totalPages":N}`

Toggle status akun (200)	`Request:`
	`PATCH /api/admin/users/{uid}/status`
	`Body: {"isActive":false}`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"message":"Status updated"}`

Reset password (200)	`Request:`
	`POST /api/admin/users/{uid}/reset-password`
	`Body: {"newPassword":"<PASSWORD_BARU>"}`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"message":"Password reset success"}`

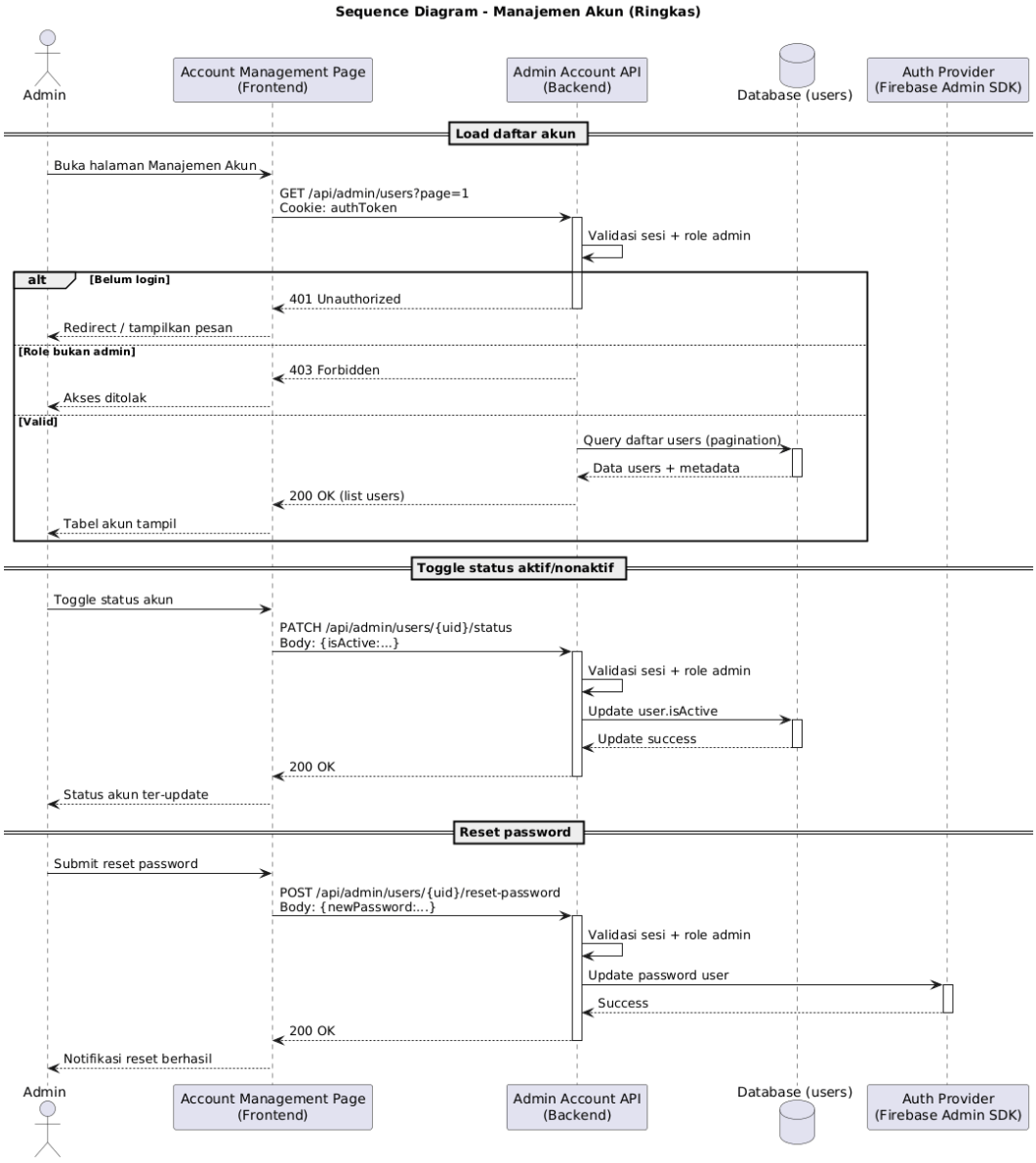
Hapus akun (200)	`Request:`
	`DELETE /api/admin/users/{uid}`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"message":"User deleted"}`

Belum login / sesi tidak valid (401)	`Request:`
	`GET /api/admin/users?page=1`
	
	`Response:`
	`HTTP/1.1 401 Unauthorized`
	`Body: {"message":"Unauthorized","error":"UNAUTHORIZED"}`

Role bukan admin (403)	`Request:`
	`GET /api/admin/users?page=1`
	
	`Response:`
	`HTTP/1.1 403 Forbidden`
	`Body: {"message":"Forbidden","error":"FORBIDDEN"}`

Sequence Diagram

Untuk memperjelas interaksi antara admin, *frontend*, dan *backend*, Gambar 3.26 menunjukkan alur utama pada modul Manajemen Akun, yaitu ketika admin memuat daftar akun (termasuk mekanisme validasi sesi dan role admin), kemudian menjalankan aksi administratif yang paling sering digunakan seperti perubahan status aktif/nonaktif serta *reset password*. Alur ini menekankan bahwa setiap permintaan diproteksi oleh kontrol akses sebelum backend melakukan pembaruan data.

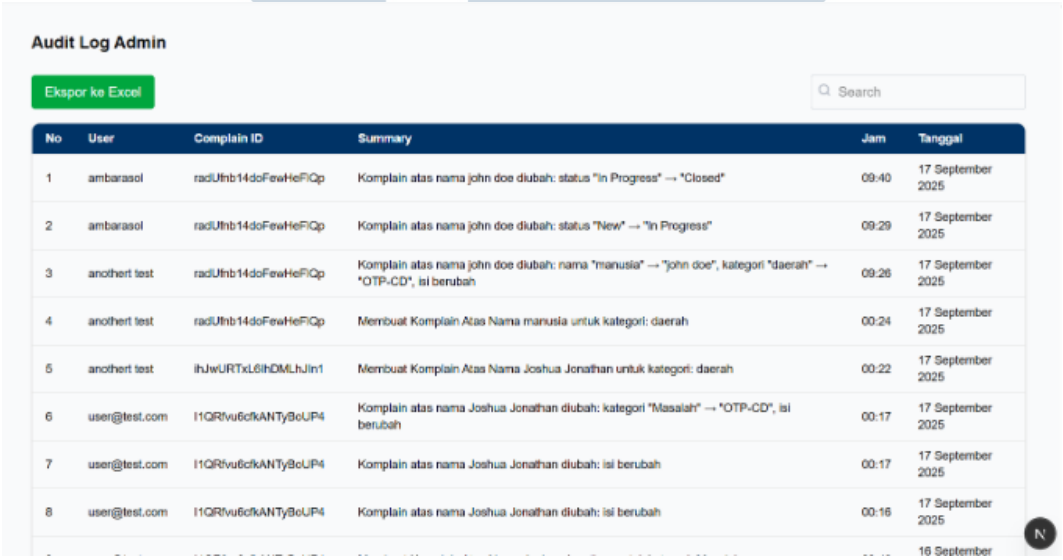


Gambar 3.26. Sequence diagram proses Manajemen Akun (Admin)

3.3.2.11 Audit Log Admin

Modul *Audit Log* merupakan fitur khusus *Admin* yang digunakan untuk melacak aktivitas penting di dalam sistem, seperti pembuatan, perubahan, dan penghapusan data komplain. Pencatatan ini bertujuan menjaga akuntabilitas, memudahkan penelusuran ketika terjadi kesalahan operasional, serta menyediakan jejak perubahan (*traceability*) yang dapat ditinjau kembali berdasarkan waktu, pelaku, dan target perubahan (misalnya ID tiket).

Secara umum, alur audit log dimulai ketika admin membuka halaman audit log dan sistem memuat data log terbaru. Admin dapat melakukan pencarian untuk menemukan aktivitas tertentu dengan cepat, menelusuri log melalui pagination, serta mengekspor data log sesuai kondisi tampilan yang sedang aktif (misalnya berdasarkan hasil pencarian atau halaman tertentu). Apabila sesi tidak valid atau role tidak sesuai, sistem menolak akses untuk memastikan audit log hanya dapat diakses oleh admin.



Gambar 3.27. Tampilan modul Audit Log Admin.

Implementasi Frontend

Dari sisi *frontend*, modul Audit Log diimplementasikan sebagai satu halaman monitoring pada dashboard admin. Halaman ini menyediakan dua kontrol utama, yaitu *search bar* untuk pencarian cepat dan tombol *Ekspor ke Excel* untuk mengunduh data log sesuai kondisi tampilan yang sedang aktif. Kolom pencarian membantu admin menemukan aktivitas berdasarkan kata kunci seperti nama pelaku,

ID tiket, atau ringkasan aktivitas, sehingga admin tidak perlu menelusuri halaman satu per satu.

Data log ditampilkan secara responsif. Pada perangkat *desktop*, data disajikan dalam bentuk tabel agar kolom-kolom penting dapat dibaca sekaligus, seperti pengguna, ID tiket, ringkasan aktivitas, waktu, dan tanggal. Pada perangkat *mobile*, data disajikan dalam bentuk kartu ringkas agar tetap nyaman dibaca dan mudah di-scroll. Untuk memperjelas konteks aktivitas, tampilan *mobile* dapat menampilkan penanda visual berdasarkan jenis aksi (misalnya *create*, *edit*, atau *delete*) melalui badge atau gaya kartu yang berbeda.

Agar pengalaman penggunaan tetap konsisten, halaman menampilkan indikator *loading* saat proses pengambilan data berlangsung, serta *empty state* ketika data tidak tersedia atau hasil pencarian tidak menemukan log. Untuk menjaga keterbacaan ketika jumlah log banyak, halaman menyediakan pagination dengan ukuran halaman tetap dan navigasi halaman yang jelas, serta menonaktifkan interaksi tertentu ketika data sedang dimuat agar tidak terjadi permintaan berulang yang tidak diperlukan.

Implementasi Backend

Pada sisi *backend*, akses audit log dibatasi ketat untuk peran *Admin* melalui validasi autentikasi dan otorisasi sebelum data dikembalikan. Jika sesi login tidak valid, backend mengembalikan respons *Unauthorized* (401). Jika pengguna login tetapi role bukan admin, backend mengembalikan respons *Forbidden* (403). Pembatasan ini memastikan data audit tidak dapat diakses oleh *User* maupun *Staff*.

Sumber data audit disimpan pada koleksi `audit_logs` di Cloud Firestore. Saat halaman dibuka, backend mengambil data log dengan urutan waktu terbaru terlebih dahulu (*order by timestamp desc*) dan menerapkan pagination agar pemuatan data tetap efisien. Setiap dokumen audit dipetakan menjadi entitas log yang memuat informasi inti seperti jenis aksi, ID target (misalnya ID tiket), nama pelaku, ringkasan perubahan, serta *timestamp*. Jika diperlukan untuk investigasi, data *before* dan *after* dapat disertakan sebagai jejak perubahan agar admin dapat menelusuri apa yang berubah pada suatu aksi.

Fitur ekspor memanfaatkan dataset log sesuai kondisi tampilan yang aktif, sehingga file Excel yang dihasilkan selaras dengan kebutuhan admin saat itu. Untuk keamanan di tingkat basis data, aturan *Security Rules* juga membatasi koleksi `audit_logs` agar hanya admin yang dapat membaca data audit, sedangkan

pencatatan log dilakukan oleh sistem ketika terjadi aksi penting yang perlu didokumentasikan.

Rangkuman status dan makna respons API audit log ditampilkan pada Tabel 3.25. Contoh request dan respons ditampilkan pada Tabel 3.26.

Tabel 3.25. Ringkasan Status dan Makna Respons API Audit Log (Admin)

Kondisi	**Status HTTP**	**Makna dan Dampak**
Request berhasil	200	Backend mengembalikan data audit log sesuai pagination atau filter agar dapat ditampilkan pada tabel/kartu dan diekspor bila diperlukan.
Parameter request tidak valid	400	Request tidak memenuhi validasi (misalnya parameter halaman tidak valid) sehingga proses dihentikan.
Belum login / sesi tidak valid	401	Backend menolak akses karena pengguna belum terautentikasi.
Role bukan admin	403	Backend menolak akses karena audit log bersifat sensitif dan hanya untuk admin.
Gagal mengambil data / gagal proses ekspor	500	Terjadi kegagalan proses di server sehingga data log tidak dapat dimuat atau file ekspor tidak dapat dihasilkan.

Tabel 3.26. Contoh Request dan Respons API Audit Log (Admin)

Skenario	**Contoh Request dan Respons**

Memuat audit log (200)	`Request:`
	`GET /api/admin/audit-logs?page=1`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"data":[...], "page":1, "totalPages":N}`

Memuat audit log dengan pencarian (200)	`Request:`
	`GET /api/admin/audit-logs?page=1`
	`&q=edit`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Body: {"data":[...], "page":1, "totalPages":N}`

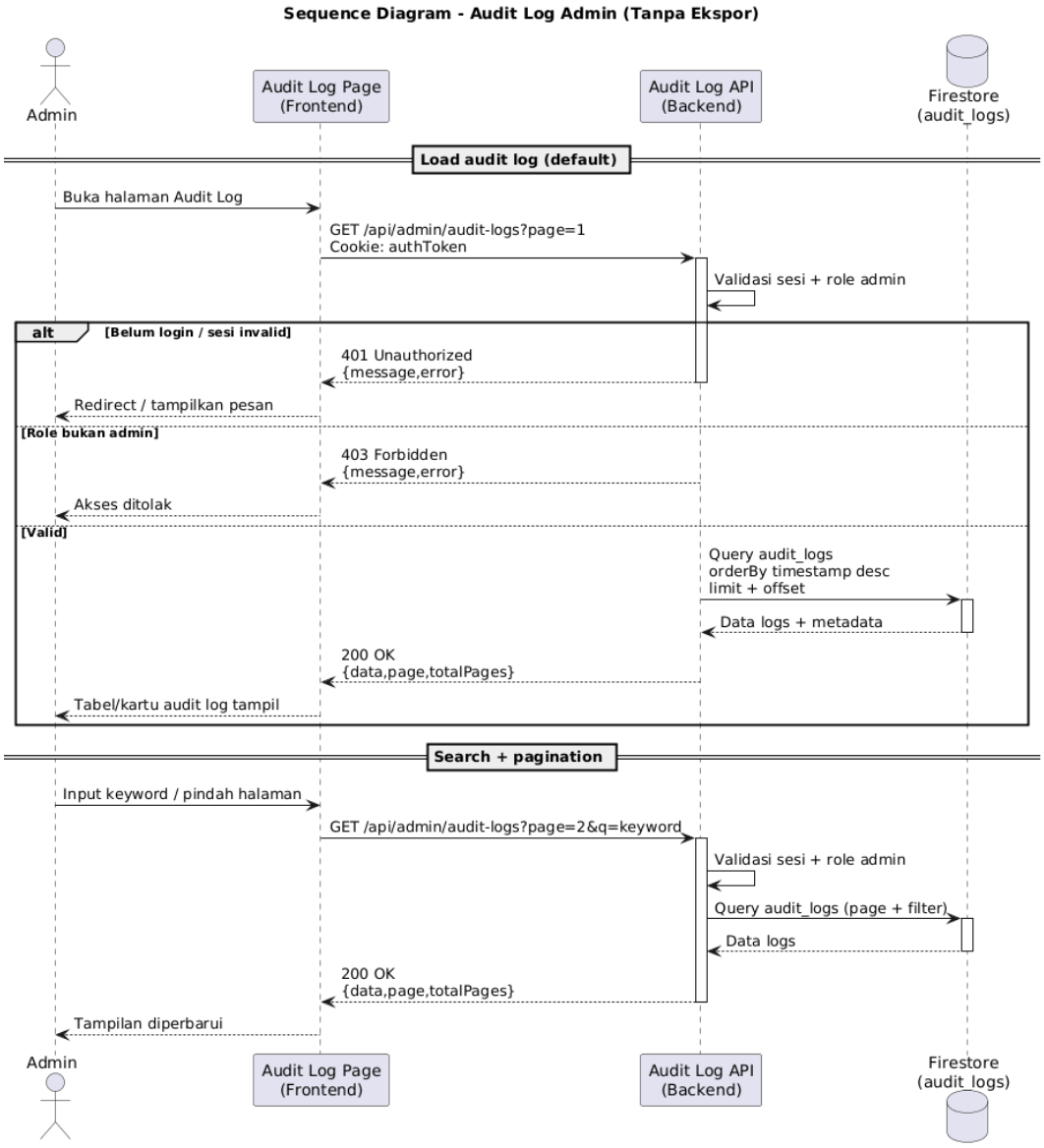
Ekspor audit log ke Excel (200)	`Request:`
	`GET /api/admin/audit-logs/export?q=edit`
	
	`Response:`
	`HTTP/1.1 200 OK`
	`Content-Type: application/vnd.openxmlformats-officedocument.`
	`spreadsheetml.sheet`

Belum login / sesi tidak valid (401)	`Request:`
	`GET /api/admin/audit-logs?page=1`
	
	`Response:`
	`HTTP/1.1 401 Unauthorized`
	`Body: {"message":"Unauthorized","error":"UNAUTHORIZED"}`

Role bukan admin (403)	`Request:`
	`GET /api/admin/audit-logs?page=1`
	
	`Response:`
	`HTTP/1.1 403 Forbidden`
	`Body: {"message":"Forbidden","error":"FORBIDDEN"}`

Sequence Diagram

Untuk memperjelas interaksi antara admin, *frontend*, dan *backend*, Gambar 3.28 menunjukkan alur ketika admin memuat data audit log, melakukan pencarian dan pagination, serta mengekspor data log sesuai kondisi tampilan yang aktif.



Gambar 3.28. Sequence diagram proses Audit Log Admin

3.3.3 Sesi Login Berakhir Otomatis Jika Tidak Ada Aktivitas Selama 5 Menit

Pada sistem yang dikembangkan, pengamanan sesi pada area dashboard diterapkan pada lapisan *middleware* sebagai gerbang utama akses (*server-side gate*). Pendekatan ini dipilih karena middleware dieksekusi sebelum halaman dashboard diproses dan sebelum konten internal dikirim ke klien, sehingga sistem dapat menolak akses lebih awal ketika sesi tidak valid tanpa bergantung pada validasi di sisi *frontend*. Mekanisme ini diterapkan khusus pada rute dashboard (area privat), sehingga halaman publik tidak terdampak.

Selain validasi sesi dasar, sistem menerapkan *idle timeout* selama lima menit sebagai pengamanan tambahan. Jika tidak ada aktivitas pengguna melewati batas tersebut, sistem mengakhiri sesi dan mengarahkan pengguna kembali ke halaman login. Kebijakan ini mengurangi risiko akses tidak sah ketika pengguna meninggalkan perangkat dalam keadaan masih login, serta membantu mengurangi permintaan yang tidak perlu pada layanan backend ketika sesi dibiarkan menganggur.

3.3.3.1 Pencatatan Aktivitas Pengguna

Sistem menyimpan waktu aktivitas terakhir pengguna dalam bentuk *timestamp* pada cookie `lastActive`. Pembaruan nilai `lastActive` dilakukan secara terkontrol melalui proses *server-side* agar keputusan validasi tidak bergantung pada manipulasi di sisi klien. Untuk mendukung pembaruan aktivitas pada interaksi tertentu yang tidak selalu memicu perpindahan halaman, disediakan endpoint khusus yang dipanggil oleh dashboard ketika terjadi interaksi (misalnya pemanggilan data AJAX atau aksi UI tertentu).

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

export const runtime = 'nodejs';

export async function POST() {
  const now = Date.now().toString();
  const res = NextResponse.json({ ok: true });

  res.cookies.set('lastActive', now, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'lax',
```

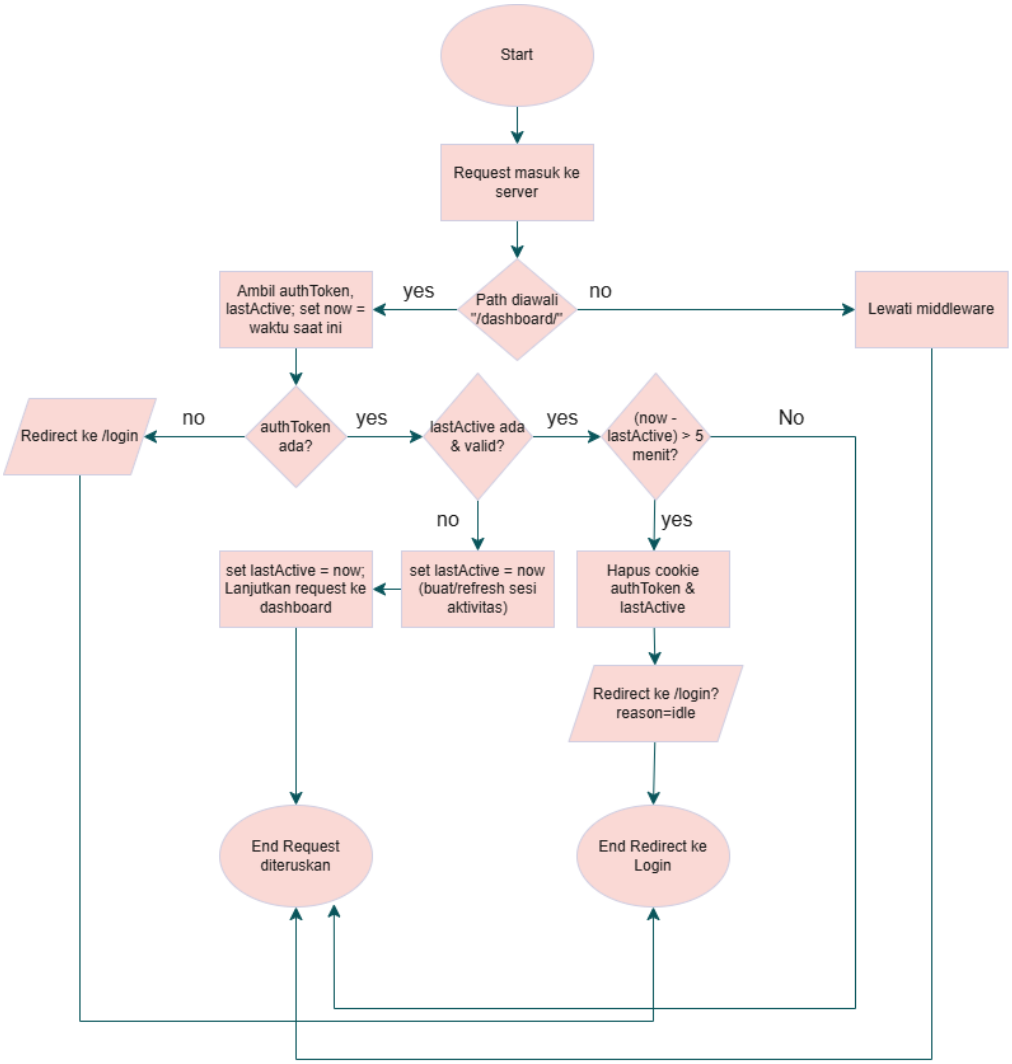
```
    path: '/',
    maxAge: 60 * 60,
  });

  return res;
}
```

Kode 3.2: Endpoint untuk memperbarui cookie aktivitas terakhir pengguna (lastActive).

Flowchart Validasi Idle Timeout

Untuk memberikan gambaran ringkas terhadap keputusan yang diambil sistem pada setiap akses rute dashboard, Gambar 3.29 menunjukkan alur validasi sesi pada middleware mulai dari pemeriksaan `authToken`, evaluasi selisih waktu idle berdasarkan `lastActive`, hingga keputusan untuk melanjutkan akses atau mengakhiri sesi dan mengarahkan pengguna ke halaman login.



Gambar 3.29. Flowchart validasi sesi dan idle timeout 5 menit pada middleware dashboard.

3.3.3.2 Validasi Idle Timeout dan Redirect Login di Middleware

Validasi dilakukan di middleware agar konsisten untuk seluruh rute dashboard. Middleware membaca `authToken` sebagai indikator autentikasi dan `lastActive` sebagai indikator aktivitas terakhir. Jika token tidak ada, pengguna dianggap belum terautentikasi dan diarahkan ke halaman login. Jika token ada tetapi selisih waktu antara `now` dan `lastActive` melebihi lima menit, middleware melakukan *logout paksa* dengan menghapus cookie sesi, lalu melakukan *redirect* ke halaman login dengan parameter alasan agar pengguna memahami penyebab pengalihan.

```
const IDLE_MS = 5 * 60 * 1000; // 5 menit
```

```
2
3 export function middleware(req: NextRequest) {
4   if (!req.nextUrl.pathname.startsWith("/dashboard")) return
      NextResponse.next();
5
6   const authToken  = req.cookies.get("authToken")?.value;
7   const lastActive = Number(req.cookies.get("lastActive")?.value);
8   const now = Date.now();
9
10  if (!authToken) {
11    return NextResponse.redirect(new URL("/login", req.url));
12  }
13
14  if (!Number.isNaN(lastActive) && now - lastActive > IDLE_MS) {
15    const res = NextResponse.redirect(new URL("/login?reason=idle
      ", req.url));
16    res.cookies.set("authToken", "", { path: "/", httpOnly: true,
      maxAge: 0 });
17    res.cookies.set("lastActive", "", { path: "/", httpOnly: true,
      maxAge: 0 });
18    return res;
19  }
20
21  const res = NextResponse.next();
22  res.cookies.set("lastActive", String(now), { path: "/", httpOnly
      : true });
23  return res;
24 }
25
26 export const config = { matcher: ["/dashboard/:path*"] };
```

Kode 3.3: Snippet inti validasi sesi dan idle timeout di middleware.

3.3.3.3 Lapisan Tambahan: Validasi Token Firebase di Server

Sebagai pengamanan berlapis (*defense in depth*), sistem juga memverifikasi session cookie menggunakan Firebase Admin SDK pada proses server-side tertentu. Verifikasi ini memastikan token benar-benar valid dan belum dicabut (*revoked*). Jika verifikasi gagal, akses ditolak dan pengguna diarahkan kembali ke halaman login.

```
1 'use server';
2
3 import { redirect } from 'next/navigation';
```

```
import { cookies } from 'next/headers';
import { adminAuth } from '@/lib/firebase-admin';

export async function requireSession(nextPath: string) {
  const cookieStore = await cookies();
  const token = cookieStore.get('authToken')?.value;

  if (!token) redirect(`/login?next=${encodeURIComponent(nextPath)
    }`);

  try {
    await adminAuth.verifySessionCookie(token, true);
  } catch {
    redirect(`/login?next=${encodeURIComponent(nextPath)}`);
  }
}
```

Kode 3.4: Validasi sesi server-side menggunakan Firebase Admin SDK.

3.3.3.4 Ringkasan Kondisi dan Dampak

Untuk memudahkan pemahaman perilaku sistem, Tabel **??** merangkum kondisi utama yang terjadi pada akses dashboard dan dampaknya terhadap pengguna.

Kondisi	**Respons HTTP**	**Dampak**
Cookie `authToken` tidak ada	3xx Redirect	Pengguna dianggap belum login dan diarahkan ke halaman login.
Idle melebihi 5 menit	3xx Redirect	Cookie sesi dihapus dan pengguna diarahkan ke login dengan alasan `reason=idle`.
Sesi valid dan masih aktif	200 OK	Akses dashboard diizinkan dan `lastActive` diperbarui.
Token dicabut / verifikasi Firebase gagal	3xx Redirect	Akses ditolak dan pengguna diarahkan ke halaman login.

3.3.3.5 Kesimpulan

Penempatan kontrol sesi pada middleware menjadikan validasi akses dashboard bersifat konsisten, server-side, dan sulit dilewati melalui manipulasi di sisi klien. Kebijakan *idle timeout* lima menit memperkuat keamanan akses dengan mengurangi risiko perangkat ditinggalkan dalam keadaan login, serta membantu menekan aktivitas sesi menganggur pada layanan backend.