API Documentation - Customer Service AI Platform

Dokumentasi lengkap untuk API Authentication, Ticketing System, dan AI Analysis.

Base URLs

```
Auth API:    http://localhost:3000/api/auth
Tickets API: http://localhost:3000/api/tickets
AI API:      http://localhost:3000/api/ai
Agent API:   http://localhost:3000/api/agent
```

Untuk production, ganti dengan domain production Anda.

Table of Contents

1. Overview
2. Architecture
3. Authentication API
   - Register
   - Customer Login
   - Agent Login
   - Verify Token
4. Tickets API
   - Create Ticket
   - Get All Tickets
   - Get Ticket by ID
   - Send Message
   - Get Messages
5. AI Analysis API
   - AI Worker (Auto-triggered)
   - Re-analyze Ticket
6. Agent API
   - Get Agent Tickets
   - Get Agent Messages
7. Database Structure
8. AI Analysis Flow
9. Environment Setup
10. Frontend Integration
11. Changelog

PROF

Overview

Platform customer service dengan AI-powered ticket analysis menggunakan Google Gemini 2.5 Flash.

Key Features

- ✅ **Dual Authentication**: Customer & Agent login terpisah dengan agent key verification
- ✅ **Unified Tickets API**: Semua data (ticket + messages) dalam satu collection dengan subcollection

- ✅ **AI-Powered Analysis**: Automatic sentiment, urgency, summary, dan suggested reply
- ✅ **Re-analyze Feature**: Agent dapat trigger re-analysis dengan full conversation context
- ✅ **Real-time Updates**: Firestore onSnapshot listeners untuk instant UI updates
- ✅ **Role-based Access**: Customer hanya lihat tickets mereka, Agent lihat semua + AI results

---

Architecture

Database Structure (Firestore)

```
tickets/ (collection)
  ├─ {ticketId}/ (document)
  │   ├─ ticketId: string
  │   ├─ subject: string
  │   ├─ description: string
  │   ├─ category: string
  │   ├─ status: "open" | "in-progress" | "resolved"
  │   ├─ customerId: string
  │   ├─ customerName: string
  │   ├─ customerEmail: string
  │   ├─ createdAt: timestamp
  │   ├─ updatedAt: timestamp
  │   ├─ aiAnalysis: {
  │   │    status: "pending" | "processing" | "done" | "error"
  │   │    mood: string (e.g., "frustrated", "calm", "urgent")
  │   │    sentiment: string (e.g., "negative", "neutral", "positive")
  │   │    urgency: "low" | "medium" | "high" | "critical"
  │   │    urgencyScore: number (1-100)
  │   │    summary: string (max 150 chars)
  │   │    suggestedResponse: string (max 200 chars)
  │   │    processedAt: timestamp
  │   │    reprocessRequested: boolean
  │   │    reprocessCount: number
  │   │    lastProcessedAt: timestamp
  │   │    conversationLength: number
  │   │    error: string | null
  │   │  }
  │   └─ messages/ (subcollection)
  │       └─ {messageId}/ (document)
  │           ├─ messageId: string
  │           ├─ message: string
  │           ├─ senderId: string
  │           ├─ senderRole: "customer" | "agent"
  │           ├─ createdAt: timestamp
  │           └─ read: boolean
  │
users/ (collection)
  └─ {userId}/ (document)
      ├─ uid: string
      ├─ email: string
      ├─ displayName: string
      ├─ role: "customer" | "agent"
      ├─ createdAt: timestamp
      └─ updatedAt: timestamp
```

PROF

Routes Structure

```
/login                          → Customer login
/register                       → Customer register
/customer                       → Customer dashboard
/customer/profile               → Customer profile

/agent                          → Agent login
/agent/dashboard                → Agent dashboard (with AI analysis)
/agent/tickets/{ticketId}       → Ticket detail with messages
```

API Flow

```
Customer Creates Ticket
  ↓
POST /api/tickets/create
  ↓
Save to Firestore (tickets collection)
  ↓
Auto-trigger AI Worker
  ↓
POST /api/ai/analyze-ticket { ticketId }
  ↓
Worker: status = "processing"
  ↓
Send to Gemini AI (initial message only)
  ↓
Get: mood, urgency, summary, suggested_reply
  ↓
Worker: status = "done"
  ↓
Frontend: Real-time update (onSnapshot)
```

```
Agent Clicks "Re-analyze"
  ↓
POST /api/agent/tickets/{ticketId}/reanalyze
  ↓
Backend: Fetch ALL messages from subcollection
  ↓
Build conversation context
  ↓
Set reprocessRequested = true, status = "pending"
  ↓
Trigger AI Worker with conversationContext
  ↓
POST /api/ai/analyze-ticket { ticketId, conversationContext, isReanalysis: true }
  ↓
Worker: Use full conversation prompt
  ↓
Send to Gemini AI (FULL CONVERSATION)
  ↓
Get updated: mood, urgency, summary, suggested_reply
  ↓
Overwrite AI results, increment reprocessCount
  ↓
Frontend: Real-time update with new analysis
```

PROF

API Endpoints

1. Register (Email/Password)

Mendaftarkan user baru dengan email dan password.

Endpoint

```
POST /api/auth/register
```

Request Headers

```
{
  "Content-Type": "application/json"
}
```

Request Body

```
{
  "email": "user@example.com",
  "password": "password123",
  "name": "John Doe"
}
```

Request Body Parameters

Parameter	Type	Required	Description
email	string	Yes	Email address user (harus valid)
password	string	Yes	Password (minimal 6 karakter)
name	string	Yes	Nama lengkap user

PROF

Success Response

Code: `200 OK`

```
{
  "message": "User berhasil didaftarkan",
  "customToken": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "uid": "abc123xyz",
    "email": "user@example.com",
    "name": "John Doe",
    "role": "customer",
    "createdAt": "2026-01-18T12:00:00.000Z"
  }
}
```

Error Responses

Email sudah terdaftar

```
{
  "error": "Email sudah terdaftar"
}
```

Code: `400 Bad Request`

Validasi gagal

```
{
  "error": "Email, password, dan name wajib diisi"
}
```

Code: `400 Bad Request`

Password terlalu pendek

```
{
  "error": "Password minimal 6 karakter"
}
```

Code: `400 Bad Request`

Frontend Example (JavaScript)

```
// Register user baru
async function registerUser(email, password, name) {
  try {
    const response = await fetch('/api/auth/register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email, password, name }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Registration failed');
    }

    // Auto login dengan custom token
    if (data.customToken) {
      // Gunakan Firebase SDK untuk sign in
      await signInWithCustomToken(auth, data.customToken);
    }

    return data;
  } catch (error) {
```

PROF

```
    console.error('Register error:', error);
    throw error;
  }
}
```

---

2. Customer Login

Login customer dengan email dan password yang sudah terdaftar.

Endpoint

```
POST /api/auth/login
```

Request Headers

```
{
  "Content-Type": "application/json"
}
```

Request Body

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

Request Body Parameters

Parameter	Type	Required	Description
email	string	Yes	Email address user
password	string	Yes	Password user

PROF

Success Response

Code: `200 OK`

```
{
  "message": "Login berhasil",
  "customToken": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "uid": "abc123xyz",
    "email": "user@example.com",
    "name": "John Doe",
    "role": "customer"
  }
}
```

Error Responses

Email atau password salah

```
{
  "error": "Email atau password salah"
}
```

Code: `401 Unauthorized`

Validasi gagal

```
{
  "error": "Email dan password wajib diisi"
}
```

Code: `400 Bad Request`

Frontend Example (JavaScript)

```
// Login customer dengan email/password
async function loginCustomer(email, password) {
  try {
    const response = await fetch('/api/auth/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email, password }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Login failed');
    }

    // Sign in dengan custom token
    if (data.customToken) {
      await signInWithCustomToken(auth, data.customToken);
    }

    // Redirect ke customer dashboard
    window.location.href = '/customer';

    return data;
  } catch (error) {
    console.error('Login error:', error);
    throw error;
  }
}
```

PROF

---

3. Agent Login (with Key)

Login support agent dengan email, password, **DAN agent key**. Endpoint ini terpisah dari customer login untuk keamanan tambahan.

Endpoint

```
POST /api/auth/agent/login
```

Request Headers

```
{
  "Content-Type": "application/json"
}
```

Request Body

```
{
  "email": "agent@example.com",
  "password": "password123",
  "agentKey": "support-agent-key-2026-secure"
}
```

Request Body Parameters

Parameter	Type	Required	Description
email	string	Yes	Email address agent
password	string	Yes	Password agent
agentKey	string	Yes	Agent key (dari environment variable)

PROF

Success Response

Code: `200 OK`

```
{
  "message": "Login berhasil sebagai Support Agent",
  "customToken": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "uid": "abc123xyz",
    "email": "agent@example.com",
    "name": "Agent John",
    "role": "agent"
  }
}
```

Error Responses

Agent key tidak valid

```
{
  "error": "Agent key tidak valid"
}
```

Code: `401 Unauthorized`

Email atau password salah

```
{
  "error": "Email atau password salah"
}
```

Code: `401 Unauthorized`

Validasi gagal

```
{
  "error": "Email, password, dan agent key diperlukan"
}
```

Code: `400 Bad Request`

Frontend Example (JavaScript)

```
// Login agent dengan email/password + agent key
async function loginAgent(email, password, agentKey) {
  try {
    const response = await fetch('/api/auth/agent/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email, password, agentKey }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Agent login failed');
    }

    // Sign in dengan custom token
    if (data.customToken) {
      await signInWithCustomToken(auth, data.customToken);
    }

    // Redirect ke agent dashboard
    window.location.href = '/agent/dashboard';

    return data;
  } catch (error) {
```

PROF

```
    console.error('Agent login error:', error);
    throw error;
  }
}
```

Important Notes

1. **Agent Key**: Disimpan di environment variable server (`AGENT_KEY`)
2. **Security**: Agent key hanya diverifikasi di endpoint ini, tidak di customer login
3. **Role Update**: User role otomatis diupdate menjadi 'agent' setelah login berhasil
4. **Access**: Agent dapat upgrade dari customer dengan login menggunakan agent key

---

4. Login with Google

Login atau register otomatis menggunakan akun Google. **Hanya tersedia untuk customer**, tidak untuk support agent.

Client-Side Implementation (Firebase SDK)

Google login dilakukan sepenuhnya di **client-side** menggunakan Firebase SDK, kemudian token diverifikasi di backend.

Step 1: Trigger Google Sign-in (Client-Side)

```
import { signInWithPopup, GoogleAuthProvider } from 'firebase/auth';
import { auth } from '@/lib/firebase';

async function loginWithGoogle() {
  try {
    // Buat Google provider
    const provider = new GoogleAuthProvider();
    provider.setCustomParameters({
      prompt: 'select_account' // Selalu tampilkan account picker
    });

    // Sign in dengan popup
    const result = await signInWithPopup(auth, provider);
    const user = result.user;

    // Dapatkan ID token untuk verifikasi di backend
    const idToken = await user.getIdToken();

    // Verifikasi dan simpan user di backend
    const response = await fetch('/api/auth/verify', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ idToken }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Verification failed');
    }

    return data;
```

PROF

```
  } catch (error) {
    // Handle errors
    if (error.code === 'auth/popup-closed-by-user') {
      throw new Error('Login dibatalkan');
    }
    if (error.code === 'auth/popup-blocked') {
      throw new Error('Popup diblokir. Izinkan popup untuk login dengan Google.');
    }
    throw error;
  }
}
```

Step 2: Verify Token (Backend API)

Setelah Google Sign-in berhasil, kirim ID token ke backend untuk verifikasi.

Endpoint

```
POST /api/auth/verify
```

Request Headers

```
{
  "Content-Type": "application/json"
}
```

Request Body

```
{
  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjE5ZmU..."
}
```

PROF

Request Body Parameters

Parameter	Type	Required	Description
idToken	string	Yes	Firebase ID token dari Google Sign-in

Success Response

Code: `200 OK`

```
{
  "message": "Token valid",
  "user": {
    "uid": "google_abc123xyz",
    "email": "user@gmail.com",
    "name": "John Doe",
    "role": "customer",
    "provider": "google.com",
    "photoURL": "https://lh3.googleusercontent.com/..."
  }
}
```

Error Responses

Token tidak valid

```
{
  "error": "Token tidak valid"
}
```

Code: `401 Unauthorized`

Token kadaluarsa

```
{
  "error": "Token sudah kadaluarsa"
}
```

Code: `401 Unauthorized`

Complete Google Login Example

```
// Complete implementation dengan error handling
async function handleGoogleLogin() {
  try {
    // Import yang diperlukan
    const { signInWithPopup, GoogleAuthProvider } = await import('firebase/auth');
    const { auth, googleProvider } = await import('@/lib/firebase');

    console.log('🔐 Starting Google login');

    // Sign in dengan Google popup
    const result = await signInWithPopup(auth, googleProvider);
    const firebaseUser = result.user;

    console.log('✅ Google sign-in successful:', firebaseUser.email);

    // Dapatkan ID token
    const idToken = await firebaseUser.getIdToken();

    // Verifikasi di backend
    const response = await fetch('/api/auth/verify', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ idToken }),
    });

    const data = await response.json();

    if (!response.ok) {
```

PROF

```
      throw new Error(data.error || 'Verification failed');
    }

    console.log('✅ Google login verified');

    // Redirect atau update UI
    window.location.href = '/customer';

    return data;
  } catch (error) {
    console.error('❌ Google login error:', error);

    // User-friendly error messages
    if (error.code === 'auth/popup-closed-by-user') {
      alert('Login dibatalkan');
    } else if (error.code === 'auth/popup-blocked') {
      alert('Popup diblokir. Izinkan popup untuk login dengan Google.');
    } else if (error.code === 'auth/cancelled-popup-request') {
      // Silent error - user cancelled
    } else {
      alert('Login gagal. Silakan coba lagi.');
    }

    throw error;
  }
}
```

---

4. Verify Token

Memverifikasi Firebase ID token dan mendapatkan informasi user. Endpoint ini digunakan untuk:

- Verifikasi setelah Google Sign-in
- Refresh user data
- Validasi session

Endpoint

PROF

```
POST /api/auth/verify
```

Request Headers

```
{
  "Content-Type": "application/json"
}
```

Request Body

```
{
  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjE5ZmU..."
}
```

Success Response

Code: `200 OK`

```
{
  "message": "Token valid",
  "user": {
    "uid": "abc123xyz",
    "email": "user@example.com",
    "name": "John Doe",
    "role": "customer",
    "provider": "password",
    "photoURL": null
  }
}
```

Frontend Example

```
// Verify current user token
async function verifyUserToken() {
  try {
    // Dapatkan current user dari Firebase
    const user = auth.currentUser;

    if (!user) {
      throw new Error('No user logged in');
    }

    // Dapatkan fresh token
    const idToken = await user.getIdToken(true); // force refresh

    const response = await fetch('/api/auth/verify', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ idToken }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Verification failed');
    }

    return data.user;
  } catch (error) {
    console.error('Verify token error:', error);
    throw error;
  }
}
```

PROF

Error Responses

Semua error response mengikuti format yang konsisten:

Format Error Response

```
{
  "error": "Deskripsi error yang user-friendly"
}
```

HTTP Status Codes

Code	Description	Contoh Use Case
200	Success	Request berhasil
400	Bad Request	Validasi gagal, parameter salah
401	Unauthorized	Token invalid, password salah
404	Not Found	Email tidak terdaftar
500	Internal Server Error	Error di server

Common Error Messages

Error Message	Meaning
"Email dan password wajib diisi"	Required fields kosong
"Email, password, dan name wajib diisi"	Required fields kosong saat register
"Password minimal 6 karakter"	Password terlalu pendek
"Email sudah terdaftar"	Email sudah digunakan user lain
"Email tidak terdaftar"	Email belum pernah register
"Email atau password salah"	Kredensial tidak cocok
"Token tidak valid"	ID token Firebase tidak valid
"Token sudah kadaluarsa"	ID token sudah expired
"Login dibatalkan"	User menutup popup Google
"Popup diblokir. Izinkan popup..."	Browser memblokir popup

PROF

Frontend Integration Examples

React/Next.js Implementation

1. Setup Firebase (Client-Side)

```
// src/lib/firebase.js
import { initializeApp, getApps } from 'firebase/app';
import { getAuth, GoogleAuthProvider } from 'firebase/auth';

const firebaseConfig = {
  apiKey: process.env.NEXT_PUBLIC_FIREBASE_API_KEY,
  authDomain: process.env.NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID,
  storageBucket: process.env.NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.NEXT_PUBLIC_FIREBASE_APP_ID,
};

const app = getApps().length === 0 ? initializeApp(firebaseConfig) : getApps()[0];
const auth = getAuth(app);

const googleProvider = new GoogleAuthProvider();
googleProvider.setCustomParameters({
  prompt: 'select_account'
});

export { app, auth, googleProvider };
```

2. Auth Context Provider

```
// src/contexts/AuthContext.js
'use client';

import { createContext, useContext, useState, useEffect } from 'react';
import {
  signOut,
  onAuthStateChanged,
  signInWithCustomToken,
  signInWithPopup,
} from 'firebase/auth';
import { auth, googleProvider } from '@/lib/firebase';

const AuthContext = createContext({});

export const useAuth = () => useContext(AuthContext);

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, async (firebaseUser) => {
      if (firebaseUser) {
        const idToken = await firebaseUser.getIdToken();
        setUser({
          uid: firebaseUser.uid,
          email: firebaseUser.email,
          displayName: firebaseUser.displayName,
          photoURL: firebaseUser.photoURL,
          idToken,
        });
      } else {
        setUser(null);
      }
      setLoading(false);
    });

    return () => unsubscribe();
  }, []);
```

PROF

```
// Register function
const register = async (email, password, name) => {
  const response = await fetch('/api/auth/register', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ email, password, name }),
  });

  const data = await response.json();
  if (!response.ok) throw new Error(data.error);

  if (data.customToken) {
    await signInWithCustomToken(auth, data.customToken);
  }

  return data;
};

// Login function
const login = async (email, password) => {
  const response = await fetch('/api/auth/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ email, password }),
  });

  const data = await response.json();
  if (!response.ok) throw new Error(data.error);

  if (data.customToken) {
    await signInWithCustomToken(auth, data.customToken);
  }

  return data;
};

// Google login function
const loginWithGoogle = async () => {
  const result = await signInWithPopup(auth, googleProvider);
  const idToken = await result.user.getIdToken();

  const response = await fetch('/api/auth/verify', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ idToken }),
  });

  const data = await response.json();
  if (!response.ok) throw new Error(data.error);

  return data;
};

// Logout function
const logout = async () => {
  await signOut(auth);
  setUser(null);
};
```

PROF

```
  return (
    <AuthContext.Provider value={{ user, loading, register, login, loginWithGoogle, 
logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

3. Login Form Component

```
// src/components/LoginForm.js
'use client';

import { useState } from 'react';
import { useAuth } from '@/contexts/AuthContext';
import { useRouter } from 'next/navigation';

export default function LoginForm() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const { login, loginWithGoogle } = useAuth();
  const router = useRouter();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    setLoading(true);

    try {
      await login(email, password);
      router.push('/dashboard');
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  const handleGoogleLogin = async () => {
    setError('');
    setLoading(true);

    try {
      await loginWithGoogle();
      router.push('/dashboard');
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div>
      <form onSubmit={handleSubmit}>
        <input
```

PROF

```
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          placeholder="Email"
          required
        />
        <input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder="Password"
          required
        />
        <button type="submit" disabled={loading}>
          {loading ? 'Loading...' : 'Login'}
        </button>
      </form>

      <button onClick={handleGoogleLogin} disabled={loading}>
        Login dengan Google
      </button>

      {error && <div className="error">{error}</div>}
    </div>
  );
}
```

Vanilla JavaScript Implementation

```
// Vanilla JS example
class AuthService {
  constructor() {
    this.baseURL = '/api/auth';
  }

  async register(email, password, name) {
    const response = await fetch(`${this.baseURL}/register`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password, name }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error);
    }

    return data;
  }

  async login(email, password) {
    const response = await fetch(`${this.baseURL}/login`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password }),
    });
    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error);
    }

    return data;
  }

  async verifyToken(idToken) {
    const response = await fetch(`${this.baseURL}/verify`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ idToken }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error);
    }

    return data;
  }
}

// Usage
const authService = new AuthService();

// Register
authService.register('user@example.com', 'password123', 'John Doe')
  .then(data => console.log('Registered:', data))
  .catch(error => console.error('Error:', error.message));

// Login
authService.login('user@example.com', 'password123')
  .then(data => console.log('Logged in:', data))
  .catch(error => console.error('Error:', error.message));
```

PROF

Ticketing System API Endpoints

Sistem ticketing memungkinkan customer untuk membuat ticket support dan berkomunikasi dengan agent melalui pesan.

Overview

Struktur Database:

- **Collection** `tickets`: Menyimpan informasi ticket
- **Subcollection** `tickets/{ticketId}/messages`: Menyimpan pesan-pesan dalam ticket

Status Ticket:

- `open` - Ticket baru dibuat
- `in-progress` - Ticket sedang ditangani agent
- `resolved` - Masalah sudah diselesaikan
- `closed` - Ticket ditutup

Category Types:

- `technical` - Masalah teknis/bug
- `billing` - Pertanyaan pembayaran/tagihan
- `general` - Pertanyaan umum (default)
- `account` - Masalah akun/profil
- `feature-request` - Permintaan fitur baru

---

6. Create Ticket

Membuat ticket support baru dengan pesan pertama.

Endpoint

```
POST /api/tickets/create
```

Authentication Required

✅ **Yes** - Requires Bearer Token

Request Headers

```
{
  "Content-Type": "application/json",
  "Authorization": "Bearer <Firebase_ID_Token>"
}
```

Request Body

```
{
  "subject": "Masalah login aplikasi",
  "message": "Saya tidak bisa login ke aplikasi, muncul error authentication
failed",
  "category": "technical"
}
```

PROF

Request Body Parameters

Parameter	Type	Required	Description
subject	string	Yes	Judul ticket (minimal 5 karakter)
message	string	Yes	Pesan awal (minimal 10 karakter)
category	string	No	Category type (default: "general")

Category Values:

- `technical` - Masalah teknis/bug
- `billing` - Pertanyaan pembayaran/tagihan
- `general` - Pertanyaan umum (default)

- `account` - Masalah akun/profil
- `feature-request` - Permintaan fitur baru

Success Response

Code: `201 Created`

```
{
  "success": true,
  "message": "Ticket berhasil dibuat",
  "data": {
    "ticketId": "abc123xyz",
    "customerId": "uid123",
    "customerName": "John Doe",
    "customerEmail": "john@example.com",
    "subject": "Masalah login aplikasi",
    "category": "technical",
    "status": "open",
    "createdAt": "2026-01-20T10:30:00.000Z",
    "updatedAt": "2026-01-20T10:30:00.000Z",
    "assignedTo": null,
    "messageCount": 1,
    "lastMessageAt": "2026-01-20T10:30:00.000Z"
  }
}
```

Error Responses

Subject atau message kosong

```
{
  "error": "Subject dan message wajib diisi"
}
```

Code: `400 Bad Request`

PROF

Subject terlalu pendek

```
{
  "error": "Subject minimal 5 karakter"
}
```

Code: `400 Bad Request`

Message terlalu pendek

```
{
  "error": "Message minimal 10 karakter"
}
```

Code: `400 Bad Request`

Unauthorized

```
{
  "error": "Unauthorized - Token tidak valid"
}
```

Code: `401 Unauthorized`

Frontend Example

```
import { useAuth } from '@/contexts/AuthContext';

const { getIdToken } = useAuth();

async function createTicket(subject, message, category = 'general') {
  try {
    const token = await getIdToken();

    const response = await fetch('/api/tickets/create', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`,
      },
      body: JSON.stringify({
        subject,
        message,
        category,
      }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Gagal membuat ticket');
    }

    console.log('Ticket created:', data.data.ticketId);
    return data.data;
  } catch (error) {
    console.error('Error:', error);
    throw error;
  }
}

// Usage
createTicket(
  'Tidak bisa login',
  'Saya tidak bisa login ke aplikasi, muncul error...',
  'technical'
)
  .then(ticket => console.log('Success:', ticket))
  .catch(error => console.error('Failed:', error));
```

PROF

7. Get All Tickets

Mengambil daftar tickets milik user yang sedang login. Customer hanya bisa melihat tickets milik sendiri, sedangkan agent bisa melihat semua tickets dengan filter.

Endpoint

```
GET /api/tickets
```

Authentication Required

✅ **Yes** - Requires Bearer Token

Request Headers

```
{
  "Authorization": "Bearer <Firebase_ID_Token>"
}
```

Query Parameters (Agent Only)

Parameter	Type	Required	Description
filter	string	No	Filter tickets (all/assigned/unassigned)

Filter Values (Agent Only):

- `all` - Semua tickets (default)
- `assigned` - Hanya tickets yang assigned ke agent
- `unassigned` - Hanya tickets yang belum di-assign

Note: Customer tidak bisa menggunakan filter, hanya melihat tickets milik sendiri.

Success Response (Customer)

PROF

Code: `200 OK`

```
{
  "success": true,
  "data": {
    "tickets": [
      {
        "ticketId": "abc123",
        "customerId": "uid123",
        "customerName": "John Doe",
        "customerEmail": "john@example.com",
        "subject": "Masalah login",
        "priority": "high",
        "status": "open",
        "createdAt": "2026-01-20T10:30:00.000Z",
        "updatedAt": "2026-01-20T11:00:00.000Z",
        "assignedTo": null,
        "messageCount": 3,
        "lastMessageAt": "2026-01-20T11:00:00.000Z"
      },
      {
        "ticketId": "xyz789",
        "customerId": "uid123",
        "customerName": "John Doe",
        "customerEmail": "john@example.com",
        "subject": "Fitur tidak berfungsi",
        "priority": "medium",
        "status": "in-progress",
        "createdAt": "2026-01-19T15:20:00.000Z",
        "updatedAt": "2026-01-20T09:00:00.000Z",
        "assignedTo": "agent_uid_456",
        "messageCount": 8,
        "lastMessageAt": "2026-01-20T09:00:00.000Z"
      }
    ],
    "totalTickets": 2,
    "userRole": "customer"
  }
}
```

Success Response (Agent)

Code: `200 OK`

```
{
  "success": true,
  "data": {
    "tickets": [
      {
        "ticketId": "abc123",
        "customerId": "uid123",
        "customerName": "John Doe",
        "customerEmail": "john@example.com",
        "subject": "Masalah login",
        "priority": "high",
        "status": "open",
        "createdAt": "2026-01-20T10:30:00.000Z",
        "updatedAt": "2026-01-20T11:00:00.000Z",
        "assignedTo": null,
        "messageCount": 3,
        "lastMessageAt": "2026-01-20T11:00:00.000Z"
      }
      // ... more tickets
    ],
    "totalTickets": 25,
    "userRole": "agent"
  }
}
```

Frontend Example

```
import { useAuth } from '@/contexts/AuthContext';

const { getIdToken } = useAuth();
```

```
// Customer - get my tickets
async function getMyTickets() {
  try {
    const token = await getIdToken();

    const response = await fetch('/api/tickets', {
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Gagal memuat tickets');
    }

    return data.data.tickets;
  } catch (error) {
    console.error('Error:', error);
    throw error;
  }
}

// Agent - get tickets with filter
async function getAgentTickets(filter = 'all') {
  try {
    const token = await getIdToken();

    const url = filter
      ? `/api/tickets?filter=${filter}`
      : '/api/tickets';

    const response = await fetch(url, {
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Gagal memuat tickets');
    }

    return data.data.tickets;
  } catch (error) {
    console.error('Error:', error);
    throw error;
  }
}

// Usage
getMyTickets()
  .then(tickets => console.log('My tickets:', tickets))
  .catch(error => console.error('Failed:', error));

// Agent usage
getAgentTickets('unassigned')
```

PROF

```
  .then(tickets => console.log('Unassigned tickets:', tickets))
  .catch(error => console.error('Failed:', error));
```

---

8. Send Message to Ticket

Mengirim pesan baru ke ticket yang sudah ada. Customer hanya bisa mengirim pesan ke tickets milik sendiri, agent bisa mengirim ke semua tickets.

Endpoint

```
POST /api/tickets/[ticketId]/messages
```

Authentication Required

✅ **Yes** - Requires Bearer Token

Request Headers

```
{
  "Content-Type": "application/json",
  "Authorization": "Bearer <Firebase_ID_Token>"
}
```

URL Parameters

Parameter	Type	Required	Description
ticketId	string	Yes	ID ticket tujuan

Request Body

```
{
  "message": "Terima kasih atas laporannya, kami akan segera cek masalahnya..."
}
```

PROF

Request Body Parameters

Parameter	Type	Required	Description
message	string	Yes	Isi pesan (tidak boleh kosong)

Success Response

Code: `201 Created`

```
{
  "success": true,
  "message": "Pesan berhasil dikirim",
  "data": {
    "messageId": "msg123",
    "ticketId": "abc123",
    "senderId": "uid456",
    "senderName": "Agent Smith",
    "senderRole": "agent",
    "message": "Terima kasih atas laporannya...",
    "createdAt": "2026-01-20T11:00:00.000Z",
    "isRead": false
  }
}
```

Error Responses

Message kosong

```
{
  "error": "Message tidak boleh kosong"
}
```

Code: `400 Bad Request`

Ticket tidak ditemukan

```
{
  "error": "Ticket tidak ditemukan"
}
```

Code: `404 Not Found`

Tidak punya akses

```
{
  "error": "Anda tidak memiliki akses ke ticket ini"
}
```

PROF

Code: `403 Forbidden`

Frontend Example

```
import { useAuth } from '@/contexts/AuthContext';

const { getIdToken } = useAuth();

async function sendMessage(ticketId, message) {
  try {
    const token = await getIdToken();

    const response = await fetch(`/api/tickets/${ticketId}/messages`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`,
      },
      body: JSON.stringify({ message }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Gagal mengirim pesan');
    }

    return data.data;
  } catch (error) {
    console.error('Error:', error);
    throw error;
  }
}

// Usage
sendMessage('abc123', 'Terima kasih atas bantuannya!')
  .then(msg => console.log('Message sent:', msg))
  .catch(error => console.error('Failed:', error));
```

---

9. Get Ticket Messages

Mengambil semua pesan dari sebuah ticket. Customer hanya bisa melihat pesan dari tickets milik sendiri, agent bisa melihat semua.

Endpoint

```
GET /api/tickets/[ticketId]/messages
```

Authentication Required

✅ **Yes** - Requires Bearer Token

PROF

Request Headers

```
{
  "Authorization": "Bearer <Firebase_ID_Token>"
}
```

URL Parameters

Parameter	Type	Required	Description
ticketId	string	Yes	ID ticket

Success Response

Code: `200 OK`

```
{
  "success": true,
  "data": {
    "ticketId": "abc123",
    "ticketInfo": {
      "subject": "Masalah login",
      "status": "open",
      "priority": "high",
      "createdAt": "2026-01-20T10:30:00.000Z"
    },
    "messages": [
      {
        "messageId": "msg1",
        "ticketId": "abc123",
        "senderId": "uid123",
        "senderName": "John Doe",
        "senderRole": "customer",
        "message": "Saya tidak bisa login ke aplikasi...",
        "createdAt": "2026-01-20T10:30:00.000Z",
        "isRead": true
      },
      {
        "messageId": "msg2",
        "ticketId": "abc123",
        "senderId": "uid456",
        "senderName": "Agent Smith",
        "senderRole": "agent",
        "message": "Terima kasih atas laporannya...",
        "createdAt": "2026-01-20T10:45:00.000Z",
        "isRead": false
      },
      {
        "messageId": "msg3",
        "ticketId": "abc123",
        "senderId": "uid123",
        "senderName": "John Doe",
        "senderRole": "customer",
        "message": "Sudah dicoba, masih error...",
        "createdAt": "2026-01-20T11:00:00.000Z",
        "isRead": false
      }
    ],
    "totalMessages": 3
  }
}
```

PROF

Error Responses

Ticket tidak ditemukan

```
{
  "error": "Ticket tidak ditemukan"
}
```

Code: `404 Not Found`

Tidak punya akses

```
{
  "error": "Anda tidak memiliki akses ke ticket ini"
}
```

Code: `403 Forbidden`

Frontend Example

```
import { useAuth } from '@/contexts/AuthContext';

const { getIdToken } = useAuth();

async function getMessages(ticketId) {
  try {
    const token = await getIdToken();

    const response = await fetch(`/api/tickets/${ticketId}/messages`, {
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Gagal memuat pesan');
    }

    return data.data.messages;
  } catch (error) {
    console.error('Error:', error);
    throw error;
  }
}

// Usage
getMessages('abc123')
  .then(messages => {
    console.log('Ticket messages:', messages);
    // Render messages in chat UI
  })
  .catch(error => console.error('Failed:', error));
```

PROF

Complete Chat Component Example

```
'use client';

import { useState, useEffect } from 'react';
import { useAuth } from '@/contexts/AuthContext';

export default function TicketChat({ ticketId }) {
  const [messages, setMessages] = useState([]);
  const [newMessage, setNewMessage] = useState('');
  const [loading, setLoading] = useState(false);
  const { getIdToken } = useAuth();

  // Load messages saat component mount
  useEffect(() => {
    loadMessages();
  }, [ticketId]);

  const loadMessages = async () => {
    try {
      const token = await getIdToken();
      const response = await fetch(`/api/tickets/${ticketId}/messages`, {
        headers: { 'Authorization': `Bearer ${token}` },
      });
      const data = await response.json();
      if (response.ok) {
        setMessages(data.data.messages);
      }
    } catch (error) {
      console.error('Error loading messages:', error);
    }
  };

  const sendMessage = async (e) => {
    e.preventDefault();
    if (!newMessage.trim()) return;

    setLoading(true);
    try {
      const token = await getIdToken();
      const response = await fetch(`/api/tickets/${ticketId}/messages`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${token}`,
        },
        body: JSON.stringify({ message: newMessage }),
      });

      const data = await response.json();
      if (response.ok) {
        setMessages([...messages, data.data]);
        setNewMessage('');
      }
    } catch (error) {
      console.error('Error sending message:', error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="chat-container">
      <div className="messages">
        {messages.map((msg) => (
          <div
            key={msg.messageId}
            className={`message ${msg.senderRole}`}
          >
```

PROF

```
            <strong>{msg.senderName}</strong>
            <p>{msg.message}</p>
            <small>{new Date(msg.createdAt).toLocaleString()}</small>
          </div>
        ))}
      </div>

      <form onSubmit={sendMessage}>
        <input
          type="text"
          value={newMessage}
          onChange={(e) => setNewMessage(e.target.value)}
          placeholder="Tulis pesan..."
          disabled={loading}
        />
        <button type="submit" disabled={loading || !newMessage.trim()}>
          {loading ? 'Mengirim...' : 'Kirim'}
        </button>
      </form>
    </div>
  );
}
```

---

10. Get Customer Messages (Agent Only)

Endpoint khusus untuk support agent mengambil semua pesan dari customer. Agent dapat memfilter berdasarkan status, ticket, atau customer tertentu.

Endpoint

```
GET /api/agent/messages
```

Authentication Required

✅ **Yes** - Requires Bearer Token (Agent Role Only)

PROF

Request Headers

```
{
  "Authorization": "Bearer <Firebase_ID_Token>"
}
```

Query Parameters

Parameter	Type	Required	Description
filter	string	No	Filter messages: 'all', 'unread', 'today'
ticketId	string	No	Filter by specific ticket ID
customerId	string	No	Filter by specific customer ID
limit	number	No	Max messages to return (default: 50)

Filter Values:

- `all` - Semua pesan dari customer (default)
- `unread` - Hanya pesan yang belum dibaca
- `today` - Pesan hari ini saja

Success Response

Code: `200 OK`

Example 1: All customer messages

```
{
  "success": true,
  "data": {
    "messages": [
      {
        "messageId": "msg1",
        "ticketId": "ticket123",
        "ticketSubject": "Masalah login",
        "ticketStatus": "open",
        "ticketCategory": "technical",
        "senderId": "customer_uid_123",
        "senderName": "John Doe",
        "senderRole": "customer",
        "senderEmail": "john@example.com",
        "message": "Saya tidak bisa login ke aplikasi...",
        "createdAt": "2026-01-20T14:30:00.000Z",
        "isRead": false
      },
      {
        "messageId": "msg2",
        "ticketId": "ticket456",
        "ticketSubject": "Error saat checkout",
        "ticketStatus": "in-progress",
        "ticketCategory": "billing",
        "senderId": "customer_uid_456",
        "senderName": "Jane Smith",
        "senderRole": "customer",
        "senderEmail": "jane@example.com",
        "message": "Muncul error 500 saat checkout...",
        "createdAt": "2026-01-20T13:15:00.000Z",
        "isRead": true
      }
    ],
    "totalMessages": 2,
    "filter": "all",
    "ticketId": null,
    "customerId": null
  }
}
```

PROF

Example 2: Unread messages only

```
GET /api/agent/messages?filter=unread
```

```
{
  "success": true,
  "data": {
    "messages": [
      {
        "messageId": "msg1",
        "ticketId": "ticket123",
        "ticketSubject": "Masalah login",
        "ticketStatus": "open",
        "ticketCategory": "technical",
        "senderId": "customer_uid_123",
        "senderName": "John Doe",
        "senderRole": "customer",
        "message": "Saya tidak bisa login...",
        "createdAt": "2026-01-20T14:30:00.000Z",
        "isRead": false
      }
    ],
    "totalMessages": 1,
    "filter": "unread",
    "ticketId": null,
    "customerId": null
  }
}
```

Example 3: Messages from specific ticket

```
GET /api/agent/messages?ticketId=ticket123
```

PROF

```
{
  "success": true,
  "data": {
    "messages": [
      {
        "messageId": "msg1",
        "ticketId": "ticket123",
        "senderId": "customer_uid_123",
        "senderName": "John Doe",
        "senderRole": "customer",
        "message": "Saya tidak bisa login ke aplikasi...",
        "createdAt": "2026-01-20T14:30:00.000Z",
        "isRead": false
      },
      {
        "messageId": "msg3",
        "ticketId": "ticket123",
        "senderId": "customer_uid_123",
        "senderName": "John Doe",
        "senderRole": "customer",
        "message": "Sudah dicoba, masih error...",
        "createdAt": "2026-01-20T12:00:00.000Z",
        "isRead": true
      }
    ],
    "totalMessages": 2,
    "filter": "all",
    "ticketId": "ticket123",
    "customerId": null
  }
}
```

Example 4: Messages from specific customer

```
GET /api/agent/messages?customerId=customer_uid_123&limit=10
```

Error Responses

Not an agent

```
{
  "error": "Forbidden - Hanya agent yang dapat mengakses endpoint ini"
}
```

Code: `403 Forbidden`

Unauthorized

```
{
  "error": "Unauthorized - Token tidak valid"
}
```

Code: `401 Unauthorized`

User not found

```
{
  "error": "User tidak ditemukan"
}
```

PROF

Code: `404 Not Found`

Frontend Example

```
import { useAuth } from '@/contexts/AuthContext';

const { getIdToken } = useAuth();

// Get all customer messages
async function getAllCustomerMessages() {
  try {
    const token = await getIdToken();

    const response = await fetch('/api/agent/messages', {
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Gagal memuat pesan');
    }

    return data.data.messages;
  } catch (error) {
    console.error('Error:', error);
    throw error;
  }
}

// Get unread customer messages
async function getUnreadMessages() {
  try {
    const token = await getIdToken();

    const response = await fetch('/api/agent/messages?filter=unread', {
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Gagal memuat pesan');
    }

    return data.data.messages;
  } catch (error) {
    console.error('Error:', error);
    throw error;
  }
}

// Get messages from specific ticket
async function getTicketMessages(ticketId) {
  try {
    const token = await getIdToken();

    const response = await fetch(`/api/agent/messages?ticketId=${ticketId}`, {
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Gagal memuat pesan');
    }

    return data.data.messages;
  } catch (error) {
```

PROF

```
      console.error('Error:', error);
      throw error;
    }
  }

  // Get messages from specific customer
  async function getCustomerMessages(customerId, limit = 50) {
    try {
      const token = await getIdToken();

      const response = await fetch(
        `/api/agent/messages?customerId=${customerId}&limit=${limit}`,
        {
          headers: {
            'Authorization': `Bearer ${token}`,
          },
        }
      );

      const data = await response.json();

      if (!response.ok) {
        throw new Error(data.error || 'Gagal memuat pesan');
      }

      return data.data.messages;
    } catch (error) {
      console.error('Error:', error);
      throw error;
    }
  }

  // Get today's messages
  async function getTodayMessages() {
    try {
      const token = await getIdToken();

      const response = await fetch('/api/agent/messages?filter=today', {
        headers: {
          'Authorization': `Bearer ${token}`,
        },
      });

      const data = await response.json();

      if (!response.ok) {
        throw new Error(data.error || 'Gagal memuat pesan');
      }

      return data.data.messages;
    } catch (error) {
      console.error('Error:', error);
      throw error;
    }
  }
```

PROF

Complete Agent Dashboard Component Example

```
'use client';

import { useState, useEffect } from 'react';
import { useAuth } from '@/contexts/AuthContext';

export default function AgentMessagesDashboard() {
  const [messages, setMessages] = useState([]);
  const [loading, setLoading] = useState(true);
  const [filter, setFilter] = useState('all');
  const [unreadCount, setUnreadCount] = useState(0);
  const { getIdToken } = useAuth();

  useEffect(() => {
    loadMessages();
  }, [filter]);

  const loadMessages = async () => {
    setLoading(true);
    try {
      const token = await getIdToken();
      const url = `/api/agent/messages?filter=${filter}`;

      const response = await fetch(url, {
        headers: { 'Authorization': `Bearer ${token}` },
      });

      const data = await response.json();

      if (response.ok) {
        setMessages(data.data.messages);

        // Count unread messages
        const unread = data.data.messages.filter(m => !m.isRead).length;
        setUnreadCount(unread);
      }
    } catch (error) {
      console.error('Error loading messages:', error);
    } finally {
      setLoading(false);
    }
  };

  const handleFilterChange = (newFilter) => {
    setFilter(newFilter);
  };

  return (
    <div className="agent-dashboard">
      <div className="dashboard-header">
        <h1>Customer Messages</h1>
        <div className="unread-badge">
          {unreadCount} Unread
        </div>
      </div>

      <div className="filter-buttons">
        <button
          onClick={() => handleFilterChange('all')}
          className={filter === 'all' ? 'active' : ''}
```

PROF

```
          >
            All Messages
          </button>
          <button
            onClick={() => handleFilterChange('unread')}
            className={filter === 'unread' ? 'active' : ''}
          >
            Unread
          </button>
          <button
            onClick={() => handleFilterChange('today')}
            className={filter === 'today' ? 'active' : ''}
          >
            Today
          </button>
        </div>

        {loading ? (
          <div className="loading">Loading messages...</div>
        ) : (
          <div className="messages-list">
            {messages.length === 0 ? (
              <div className="no-messages">No messages found</div>
            ) : (
              messages.map((msg) => (
                <div
                  key={msg.messageId}
                  className={`message-card ${!msg.isRead ? 'unread' : ''}`}
                >
                  <div className="message-header">
                    <div className="customer-info">
                      <strong>{msg.senderName}</strong>
                      <span className="email">{msg.senderEmail}</span>
                    </div>
                    <div className="ticket-info">
                      <span className={`category ${msg.ticketCategory}`}>
                        {msg.ticketCategory}
                      </span>
                      <span className={`status ${msg.ticketStatus}`}>
                        {msg.ticketStatus}
                      </span>
                    </div>
                  </div>

                  <div className="ticket-subject">
                    🏷️ {msg.ticketSubject}
                  </div>

                  <div className="message-content">
                    {msg.message}
                  </div>

                  <div className="message-footer">
                    <span className="timestamp">
                      {new Date(msg.createdAt).toLocaleString()}
                    </span>
                    <a
                      href={`/agent/tickets/${msg.ticketId}`}
                      className="view-ticket"
                    >
```

PROF

```
                      View Full Ticket →
                    </a>
                  </div>
                </div>
              ))
            )}
          </div>
        )}
      </div>
    );
  }
```

Use Cases

1. Agent Inbox Dashboard

- Display all unread customer messages
- Sort by priority and timestamp
- Quick access to ticket details

2. Customer Support Monitoring

- Track response times
- Identify high-priority issues
- Monitor today's customer inquiries

3. Ticket Management

- Filter messages by specific ticket
- Review customer communication history
- Assign tickets to agents

4. Customer Relationship Management

- View all messages from specific customer
- Track customer interaction history
- Identify repeat customers or issues

PROF

Important Notes

1. **Agent Only**: Endpoint hanya dapat diakses oleh user dengan role 'agent'
2. **Customer Messages Only**: Hanya menampilkan pesan yang dikirim oleh customer (senderRole: 'customer')
3. **Performance**: Default limit adalah 50 messages untuk optimasi
4. **Real-time**: Untuk real-time updates, consider implementing polling atau WebSocket
5. **Read Status**: Messages memiliki `isRead` flag untuk tracking
6. **Sorting**: Messages diurutkan dari terbaru ke terlama (descending)

Environment Setup

Backend Environment Variables

Backend membutuhkan environment variables berikut di `.env.local`:

```
# Firebase Admin SDK
FIREBASE_PROJECT_ID=your-project-id
```

```
FIREBASE_PRIVATE_KEY="-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----\n"
FIREBASE_CLIENT_EMAIL=firebase-adminsdk-xxxxx@your-project.iam.gserviceaccount.com

# Firebase Web Config (untuk REST API)
NEXT_PUBLIC_FIREBASE_API_KEY=your-api-key
NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN=your-project.firebaseapp.com
NEXT_PUBLIC_FIREBASE_PROJECT_ID=your-project-id
NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET=your-project.firebasestorage.app
NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID=123456789
NEXT_PUBLIC_FIREBASE_APP_ID=1:123456789:web:xxxxx

# Server Configuration
PORT=3001

# Agent Key for Support Agent Login
# IMPORTANT: Change this to a secure random key in production!
# Recommended: Use at least 20 characters with mix of uppercase, lowercase, numbers,
and symbols
AGENT_KEY=support-agent-key-2026-secure
```

Frontend Environment Variables

Pastikan frontend memiliki environment variables berikut:

```
# .env.local
NEXT_PUBLIC_FIREBASE_API_KEY=your_api_key
NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN=your_project.firebaseapp.com
NEXT_PUBLIC_FIREBASE_PROJECT_ID=your_project_id
NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET=your_project.appspot.com
NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID=your_sender_id
NEXT_PUBLIC_FIREBASE_APP_ID=your_app_id
```

Agent Key Setup

Development

```
# Default agent key untuk development
AGENT_KEY=support-agent-key-2026-secure
```

Production

```
# Generate strong random key
# Contoh: AG3nt-K3y-2026!$ecur3-R@nd0m-P4sSw0rd
AGENT_KEY=<your-strong-random-key>
```

Best Practices

1. **Minimal 20 karakter**
2. **Kombinasi huruf besar, kecil, angka, dan simbol**
3. **Rotate key setiap 3-6 bulan**
4. **Jangan commit ke git** (sudah di `.gitignore`)

5. **Store securely** di environment variables production

Security Best Practices

1. Token Management

- ✅ Selalu gunakan HTTPS di production
- ✅ ID token hanya valid 1 jam, perlu refresh
- ✅ Jangan simpan token di localStorage (gunakan Firebase SDK)
- ✅ Verifikasi token di backend untuk setiap request sensitif
- ✅ Custom token hanya untuk initial authentication

2. Agent Key Security

- ✅ **Never commit** agent key ke repository
- ✅ Store di environment variables yang secure
- ✅ **Minimal 20 karakter** dengan kombinasi:
  - Huruf besar (A-Z)
  - Huruf kecil (a-z)
  - Angka (0-9)
  - Simbol (!@#$%^&*)
- ✅ Rotate agent key setiap 3-6 bulan
- ✅ Different keys untuk dev, staging, production
- ✅ Audit log untuk penggunaan agent key

3. Password Requirements

- ✅ Minimal 6 karakter (dapat ditingkatkan menjadi 8+)
- ✅ Hindari password yang umum (dapat tambahkan validation)
- ✅ Implementasi rate limiting untuk login attempts
- ✅ Consider password strength indicator di frontend
- ✅ Optional: Require uppercase, lowercase, number, symbol

4. CORS Configuration

- ✅ Pastikan domain frontend ada di authorized domains Firebase
- ✅ Set proper CORS headers di backend
- ✅ Whitelist specific origins di production
- ✅ Block unknown origins

PROF

5. Error Handling

- ✅ **Jangan expose** informasi sensitif dalam error message
- ✅ Log error detail di backend untuk debugging
- ✅ Tampilkan user-friendly message di frontend
- ✅ Different error messages untuk production vs development

6. Rate Limiting

- ✅ Implement rate limiting untuk:
  - Login endpoints (max 5 attempts per 15 minutes)
  - Register endpoint (max 3 per hour per IP)
  - Agent login (max 3 attempts per 15 minutes)
- ✅ Track failed attempts per IP and per user

- ✅ Temporary block setelah multiple failed attempts

7. Separation of Concerns

- ✅ Customer dan Agent memiliki endpoint terpisah
- ✅ Agent key **hanya** diverifikasi di `/api/auth/agent/login`
- ✅ Reduce attack surface dengan isolasi endpoint
- ✅ Different logging dan monitoring per endpoint

8. Production Deployment

- ✅ Use strong, random agent keys
- ✅ Enable Firebase App Check
- ✅ Configure proper security rules di Firestore
- ✅ Monitor authentication logs
- ✅ Set up alerts untuk suspicious activities
- ✅ Regular security audits

Testing Guide

Manual Testing Checklist

✅ Email/Password Authentication (Customer)

- ☐ Register dengan email baru berhasil
- ☐ Register dengan email yang sudah ada ditolak
- ☐ Login dengan kredensial yang benar berhasil
- ☐ Login dengan kredensial yang salah ditolak
- ☐ Password kurang dari 6 karakter ditolak
- ☐ Redirect ke `/customer` setelah login

✅ Agent Login dengan Agent Key

- ☐ Login agent dengan key valid berhasil
- ☐ Login agent dengan key invalid ditolak
- ☐ Login agent tanpa key ditolak
- ☐ Role user berubah menjadi 'agent'
- ☐ Redirect ke `/agent/dashboard` setelah login

PROF

✅ Google Authentication (Customer Only)

- ☐ Google popup muncul dengan benar
- ☐ Login dengan akun Google berhasil
- ☐ User baru otomatis terdaftar dengan role 'customer'
- ☐ User existing dapat login
- ☐ Error handling untuk popup closed
- ☐ Error handling untuk popup blocked
- ☐ Redirect ke `/customer` setelah login

✅ Token Verification

- ☐ Token valid diverifikasi dengan benar
- ☐ Token invalid ditolak
- ☐ Token expired ditolak

- [ ] User role ter-fetch dengan benar

✅ Ticketing System

- [ ] Customer bisa membuat ticket baru
- [ ] Subject dan message divalidasi dengan benar
- [ ] Ticket tersimpan di Firestore dengan data lengkap
- [ ] Pesan pertama otomatis dibuat
- [ ] Customer bisa melihat semua tickets milik sendiri
- [ ] Agent bisa melihat semua tickets
- [ ] Agent bisa filter tickets (all/assigned/unassigned)
- [ ] Customer bisa kirim pesan ke tickets milik sendiri
- [ ] Agent bisa kirim pesan ke semua tickets
- [ ] Customer tidak bisa kirim pesan ke tickets orang lain
- [ ] Messages diurutkan berdasarkan waktu
- [ ] Message count di ticket auto-update
- [ ] Last message time di ticket auto-update

✅ Agent Messages Endpoint

- [ ] Agent bisa mengambil semua pesan customer
- [ ] Filter 'all' menampilkan semua pesan
- [ ] Filter 'unread' hanya menampilkan pesan belum dibaca
- [ ] Filter 'today' hanya menampilkan pesan hari ini
- [ ] Filter by ticketId berfungsi dengan benar
- [ ] Filter by customerId berfungsi dengan benar
- [ ] Limit parameter membatasi jumlah messages
- [ ] Customer tidak bisa akses endpoint ini
- [ ] Messages include ticket information (subject, status, priority)
- [ ] Messages diurutkan dari terbaru ke terlama

✅ Role-Based Access Control

- [ ] Customer tidak bisa akses `/agent/dashboard`
- [ ] Agent tidak bisa akses `/customer`
- [ ] Role persisten setelah page reload
- [ ] Logout berfungsi untuk kedua role
- [ ] Auto redirect berdasarkan role

PROF

Test Scenarios

Scenario 1: Customer Registration & Login

1. **Register**

```
curl -X POST http://localhost:3001/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "email": "customer@test.com",
    "password": "password123",
    "name": "Test Customer"
  }'
```

Expected: `200 OK` dengan customToken

2. **Login**

```
curl -X POST http://localhost:3001/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "customer@test.com",
    "password": "password123"
  }'
```

Expected: `200 OK` dengan role "customer"

Scenario 2: Agent Login dengan Key

1. **Login dengan key valid**

```
curl -X POST http://localhost:3001/api/auth/agent/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "agent@test.com",
    "password": "password123",
    "agentKey": "support-agent-key-2026-secure"
  }'
```

Expected: `200 OK` dengan role "agent"

2. **Login dengan key invalid**

```
curl -X POST http://localhost:3001/api/auth/agent/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "agent@test.com",
    "password": "password123",
    "agentKey": "wrong-key"
  }'
```

PROF

Expected: `401 Unauthorized` dengan error "Agent key tidak valid"

Scenario 3: Role Upgrade (Customer → Agent)

1. Login sebagai customer pertama kali
2. Logout
3. Login kembali melalui `/agent` dengan agent key
4. Check Firestore: role harus berubah dari "customer" ke "agent"

Scenario 4: Create Ticket (Customer)

1. **Login sebagai customer** (dapatkan token)

```
# Simpan token dari response login
TOKEN="your_firebase_token_here"
```

2. **Create ticket**

```
curl -X POST http://localhost:3001/api/tickets/create \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{
    "subject": "Masalah login aplikasi",
    "message": "Saya tidak bisa login ke aplikasi, muncul error authentication
failed",
    "priority": "high"
  }'
```

Expected: `201 Created` dengan ticketId

3. **Verify di Firestore**
   - Check collection `tickets` → ticket baru muncul
   - Check subcollection `messages` → pesan pertama ada

Scenario 5: Get Tickets

1. **Customer get their tickets**

```
curl -X GET http://localhost:3001/api/tickets \
  -H "Authorization: Bearer $CUSTOMER_TOKEN"
```

Expected: `200 OK` dengan array tickets milik customer

2. **Agent get all tickets**

```
curl -X GET http://localhost:3001/api/tickets \
  -H "Authorization: Bearer $AGENT_TOKEN"
```

PROF

Expected: `200 OK` dengan semua tickets

3. **Agent get unassigned tickets**

```
curl -X GET "http://localhost:3001/api/tickets?filter=unassigned" \
  -H "Authorization: Bearer $AGENT_TOKEN"
```

Expected: `200 OK` dengan tickets yang belum di-assign

Scenario 6: Send Message to Ticket

1. **Customer send message**

```
curl -X POST http://localhost:3001/api/tickets/abc123/messages \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $CUSTOMER_TOKEN" \
  -d '{
    "message": "Apakah sudah ada update untuk masalah saya?"
  }'
```

Expected: `201 Created`

2. **Agent reply**

```
curl -X POST http://localhost:3001/api/tickets/abc123/messages \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $AGENT_TOKEN" \
  -d '{
    "message": "Kami sedang mengecek masalahnya, mohon tunggu sebentar"
  }'
```

Expected: `201 Created`

3. **Customer try to send to other customer ticket**

```
curl -X POST http://localhost:3001/api/tickets/xyz789/messages \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $CUSTOMER_TOKEN" \
  -d '{
    "message": "Test message"
  }'
```

Expected: `403 Forbidden` (jika ticket bukan milik customer)

Scenario 7: Get Ticket Messages

1. **Get all messages from ticket**

```
curl -X GET http://localhost:3001/api/tickets/abc123/messages \
  -H "Authorization: Bearer $CUSTOMER_TOKEN"
```

Expected: `200 OK` dengan array messages, sorted by createdAt

2. **Verify message order**

   - Messages diurutkan dari yang terlama (ascending)
   - Setiap message punya sender info (name, role)

Scenario 8: End-to-End Ticket Flow

1. Customer login
2. Customer buat ticket baru
3. Customer kirim pesan kedua
4. Agent login

5. Agent lihat semua tickets
6. Agent balas pesan customer
7. Customer refresh dan lihat balasan agent
8. Customer kirim pesan lagi
9. Verify di Firestore: semua data tersimpan dengan benar

Scenario 9: Agent Get Customer Messages

1. **Agent get all customer messages**

   ```
   curl -X GET http://localhost:3001/api/agent/messages \
     -H "Authorization: Bearer $AGENT_TOKEN"
   ```

   Expected: `200 OK` dengan array messages dari semua customer

2. **Agent get unread messages**

   ```
   curl -X GET "http://localhost:3001/api/agent/messages?filter=unread" \
     -H "Authorization: Bearer $AGENT_TOKEN"
   ```

   Expected: `200 OK` dengan hanya messages yang isRead = false

3. **Agent get today's messages**

   ```
   curl -X GET "http://localhost:3001/api/agent/messages?filter=today" \
     -H "Authorization: Bearer $AGENT_TOKEN"
   ```

   Expected: `200 OK` dengan messages hari ini saja

4. **Agent get messages from specific ticket**

   ```
   curl -X GET "http://localhost:3001/api/agent/messages?ticketId=abc123" \
     -H "Authorization: Bearer $AGENT_TOKEN"
   ```

   Expected: `200 OK` dengan messages dari ticket abc123

5. **Agent get messages from specific customer**

   ```
   curl -X GET "http://localhost:3001/api/agent/messages?
   customerId=customer_uid_123&limit=10" \
     -H "Authorization: Bearer $AGENT_TOKEN"
   ```

   Expected: `200 OK` dengan max 10 messages dari customer tersebut

6. **Customer try to access agent endpoint**

   ```
   curl -X GET http://localhost:3001/api/agent/messages \
     -H "Authorization: Bearer $CUSTOMER_TOKEN"
   ```

PROF

Expected: `403 Forbidden` dengan error "Hanya agent yang dapat mengakses endpoint ini"

7. **Verify message structure**
   - Each message harus punya: messageId, ticketId, senderId, senderName, senderRole, message, createdAt, isRead
   - Jika bukan dari ticketId spesifik, harus ada: ticketSubject, ticketStatus, ticketCategory
   - senderRole harus selalu 'customer'
   - Messages sorted descending by createdAt

Troubleshooting

Common Issues

1. "Agent key tidak valid"

Cause: Agent key yang dimasukkan tidak cocok dengan `AGENT_KEY` di environment variable

Solution:

```
# Check environment variable
cat .env.local | grep AGENT_KEY

# Restart dev server setelah mengubah .env.local
npm run dev
```

2. User tidak redirect setelah login

Cause: Error di client-side atau custom token tidak di-apply

Solution:

- Check browser console (F12) untuk error
- Pastikan `signInWithCustomToken()` dipanggil
- Verify `router.push()` atau `window.location.href` dipanggil

PROF

3. Role tidak update di Firestore

Cause: Firestore rules atau permission issue

Solution:

- Check Firestore security rules
- Verify backend memiliki admin permission
- Check server logs untuk error

4. "Email sudah terdaftar" saat register

Cause: Email sudah digunakan user lain

Solution:

- Gunakan email yang berbeda

- Atau login dengan email existing

5. Google login popup diblokir

Cause: Browser memblokir popup

Solution:

- Izinkan popup untuk domain aplikasi
- Atau gunakan `signInWithRedirect()` sebagai alternatif

6. Token expired

Cause: Firebase ID token hanya valid 1 jam

Solution:

```
// Force refresh token
const idToken = await user.getIdToken(true); // true = force refresh
```

Debugging Tips

Server-Side Logs

Check terminal yang menjalankan `npm run dev`:

```
🔐 Login API called
📧 Login with email/password: user@example.com
✅ User found: abc123xyz
🔑 Creating custom token for user: abc123xyz
✅ Login successful
```

Emoji markers:

- 🔐 = Login attempt
- 🔑 = Agent key verification / Token creation
- ✅ = Success
- ❌ = Error
- ⚠ = Warning

Client-Side Logs

Check browser console (F12):

```
// Check user state
console.log(auth.currentUser);

// Check token
auth.currentUser.getIdToken().then(token => console.log(token));

// Check role
const idToken = await auth.currentUser.getIdToken();
const response = await fetch('/api/auth/verify', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ idToken })
});
const data = await response.json();
console.log('User role:', data.user.role);
```

Firestore Console

1. Buka Firebase Console > Firestore Database
2. Check collection `users`
3. Verify user document memiliki:
   - `email`
   - `name`
   - `role` (customer atau agent)
   - `createdAt` / `updatedAt`

Environment Variable Check

```
# Verify .env.local exists
ls -la .env.local

# Check content (jangan share output ini!)
cat .env.local | grep AGENT_KEY
cat .env.local | grep FIREBASE_PROJECT_ID

# Test environment variable loaded
node -e "console.log(process.env.AGENT_KEY)"
```

Changelog

Version 3.1.0 (2026-01-20)

- ✅ **NEW**: Agent Messages Endpoint (`GET /api/agent/messages`)
- ✅ Agent dapat mengambil semua pesan dari customer
- ✅ Filter support: all, unread, today
- ✅ Filter by ticketId atau customerId
- ✅ Customizable limit untuk pagination
- ✅ Complete dashboard component examples
- ✅ Updated documentation with use cases

PROF

Version 3.0.0 (2026-01-20)

- ✅ **NEW**: Ticketing System API
- ✅ **NEW**: Create ticket endpoint (`POST /api/tickets/create`)
- ✅ **NEW**: Get tickets endpoint (`GET /api/tickets`)
- ✅ **NEW**: Send message endpoint (`POST /api/tickets/[ticketId]/messages`)
- ✅ **NEW**: Get messages endpoint (`GET /api/tickets/[ticketId]/messages`)
- ✅ Firestore structure for tickets and messages
- ✅ Role-based access control for tickets
- ✅ Agent filter support (all/assigned/unassigned)
- ✅ Complete documentation with examples

- ✅ Test scenarios for ticketing API
- ✅ Frontend integration examples

Version 2.0.0 (2026-01-19)

- ✅ **BREAKING**: Separated agent login to `/api/auth/agent/login`
- ✅ **BREAKING**: Agent dashboard moved to `/agent/dashboard`
- ✅ Agent key verification on dedicated endpoint
- ✅ Security isolation between customer and agent flows
- ✅ Updated documentation with new architecture
- ✅ Added comprehensive testing guide
- ✅ Added troubleshooting section

Version 1.0.0 (2026-01-18)

- ✅ Initial release
- ✅ Email/Password authentication
- ✅ Google Sign-in integration
- ✅ Token verification endpoint
- ✅ Comprehensive error handling
- ✅ Frontend integration examples

GET Methods Guide - Panduan Lengkap Mendapatkan Data

Section ini menjelaskan **semua method GET yang tersedia**, cara menggunakannya, dan **kapan menggunakan API mana** untuk mendapatkan data yang Anda butuhkan.

📋 Overview Semua GET Endpoints

Endpoint	Authentication	Role	Description	Use Case
`GET /api/tickets`	✅ Required	Customer/Agent	Mendapatkan daftar tickets	Menampilkan semua tickets user
`GET /api/tickets/[ticketId]/messages`	✅ Required	Customer/Agent	Mendapatkan pesan dalam ticket	Chat/conversation view
`GET /api/agent/messages`	✅ Required	Agent Only	Mendapatkan semua pesan customer	Agent inbox/dashboard

---

1️⃣ GET All Tickets - `/api/tickets`

🎯 **Tujuan**: Mendapatkan daftar tickets milik user yang login

👤 **Siapa yang bisa akses**:

- **Customer**: Hanya tickets milik sendiri
- **Agent**: Semua tickets (dengan filter)

Method GET Tanpa Parameter (Customer)

PROF

Kapan Digunakan:

- Menampilkan daftar tickets di dashboard customer
- Melihat history semua tickets yang pernah dibuat
- Monitoring status tickets

Cara Pakai:

```
# Browser
http://localhost:3000/api/tickets

# cURL
curl http://localhost:3000/api/tickets

# JavaScript/Fetch
fetch('http://localhost:3000/api/tickets')
  .then(res => res.json())
  .then(data => console.log(data.data.tickets))
```

Response Format:

```
{
  "success": true,
  "data": {
    "tickets": [
      {
        "ticketId": "abc123",
        "customerId": "uid123",
        "customerName": "John Doe",
        "customerEmail": "john@example.com",
        "subject": "Masalah login",
        "priority": "high",
        "status": "open",
        "createdAt": "2026-01-20T10:30:00.000Z",
        "updatedAt": "2026-01-20T11:00:00.000Z",
        "assignedTo": null,
        "messageCount": 3,
        "lastMessageAt": "2026-01-20T11:00:00.000Z"
      }
    ],
    "totalTickets": 1,
    "userRole": "customer"
  }
}
```

PROF

Data yang Didapat:

- ✅ `ticketId` - ID unik ticket (untuk fetch messages)
- ✅ `subject` - Judul ticket
- ✅ `status` - Status (open/in-progress/resolved/closed)
- ✅ `category` - Category type (technical/billing/general/account/feature-request)
- ✅ `messageCount` - Jumlah pesan dalam ticket
- ✅ `lastMessageAt` - Timestamp pesan terakhir
- ✅ `assignedTo` - Agent yang handle (jika sudah di-assign)

Method GET Dengan Filter (Agent)

Kapan Digunakan (Agent Only):

- Melihat semua tickets yang masuk
- Filter tickets yang sudah/belum di-assign
- Filter tickets yang di-handle sendiri

Cara Pakai:

```
# Semua tickets
curl "http://localhost:3000/api/tickets?filter=all"

# Tickets yang sudah di-assign
curl "http://localhost:3000/api/tickets?filter=assigned"

# Tickets yang belum di-assign
curl "http://localhost:3000/api/tickets?filter=unassigned"
```

Query Parameters (Agent):

- `filter=all` - Semua tickets (default)
- `filter=assigned` - Hanya tickets assigned ke agent
- `filter=unassigned` - Hanya tickets yang belum di-assign

JavaScript Example (Agent):

```
// Fetch all tickets
async function getAllTickets() {
  const response = await fetch('/api/tickets?filter=all');
  const data = await response.json();
  return data.data.tickets;
}

// Fetch unassigned tickets (untuk queue)
async function getUnassignedTickets() {
  const response = await fetch('/api/tickets?filter=unassigned');
  const data = await response.json();
  return data.data.tickets;
}
```

2️⃣ GET Ticket Messages - `/api/tickets/[ticketId]/messages`

🎯 **Tujuan**: Mendapatkan semua pesan dalam sebuah ticket

👤 **Siapa yang bisa akses**:

- **Customer**: Hanya tickets milik sendiri
- **Agent**: Semua tickets

Kapan Digunakan:

- Menampilkan chat/conversation dalam ticket
- Melihat history komunikasi customer-agent

- Load messages saat user klik ticket

Cara Pakai

Step 1: Dapatkan `ticketId` dari endpoint `/api/tickets`

```
// Ambil tickets dulu
const ticketsResponse = await fetch('/api/tickets');
const ticketsData = await ticketsResponse.json();
const firstTicket = ticketsData.data.tickets[0];
const ticketId = firstTicket.ticketId; // Simpan ticketId ini
```

Step 2: Fetch messages dengan ticketId

```
# Browser (ganti [ticketId] dengan ID sebenarnya)
http://localhost:3000/api/tickets/abc123/messages

# cURL
curl http://localhost:3000/api/tickets/abc123/messages

# JavaScript/Fetch
const ticketId = 'abc123';
fetch(`http://localhost:3000/api/tickets/${ticketId}/messages`)
  .then(res => res.json())
  .then(data => console.log(data.data.messages))
```

Response Format:

```
{
  "success": true,
  "data": {
    "ticketId": "abc123",
    "ticketInfo": {
      "subject": "Masalah login",
      "status": "open",
      "priority": "high",
      "createdAt": "2026-01-20T10:30:00.000Z"
    },
    "messages": [
      {
        "messageId": "msg1",
        "ticketId": "abc123",
        "senderId": "uid123",
        "senderName": "John Doe",
        "senderRole": "customer",
        "message": "Saya tidak bisa login...",
        "createdAt": "2026-01-20T10:30:00.000Z",
        "isRead": true
      },
      {
        "messageId": "msg2",
        "ticketId": "abc123",
        "senderId": "agent_uid_456",
        "senderName": "Agent Smith",
        "senderRole": "agent",
        "message": "Terima kasih atas laporannya...",
        "createdAt": "2026-01-20T10:45:00.000Z",
        "isRead": false
      }
    ],
    "totalMessages": 2
  }
}
```

Data yang Didapat:

- ✅ `ticketInfo` - Informasi ticket (subject, status, priority)
- ✅ `messages[]` - Array semua pesan
- ✅ `senderRole` - Role pengirim (customer/agent)
- ✅ `senderName` - Nama pengirim
- ✅ `message` - Isi pesan
- ✅ `createdAt` - Timestamp pengiriman
- ✅ `isRead` - Status sudah dibaca atau belum

Complete Example - Chat Component:

```
async function loadTicketChat(ticketId) {
  try {
    // Fetch messages
    const response = await fetch(`/api/tickets/${ticketId}/messages`);
    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error);
    }

    // Tampilkan ticket info
    console.log('Subject:', data.data.ticketInfo.subject);
    console.log('Status:', data.data.ticketInfo.status);

    // Loop messages untuk chat UI
    data.data.messages.forEach(msg => {
      console.log(`[${msg.senderRole}] ${msg.senderName}: ${msg.message}`);
    });

    return data.data.messages;
  } catch (error) {
    console.error('Error loading chat:', error);
  }
}

// Usage
loadTicketChat('abc123');
```

PROF

3️⃣ GET Agent Messages - `/api/agent/messages` (Agent Only)

🎯 **Tujuan**: Mendapatkan semua pesan dari customer (agent view)

👤 **Siapa yang bisa akses**: **Agent Only** (role: agent)

Kapan Digunakan:

- Agent inbox/dashboard
- Monitoring unread messages
- Melihat pesan hari ini
- Track customer communication
- Filter pesan per ticket/customer

3.1 GET All Customer Messages (Default)

Kapan Digunakan:

- Load agent inbox untuk pertama kali
- Menampilkan semua pesan yang masuk dari customer
- Overview komunikasi

```
# Browser
http://localhost:3000/api/agent/messages

# cURL
curl http://localhost:3000/api/agent/messages

# JavaScript
fetch('/api/agent/messages')
  .then(res => res.json())
  .then(data => console.log(data.data.messages))
```

Response:

```
{
  "success": true,
  "data": {
    "messages": [
      {
        "messageId": "msg1",
        "ticketId": "ticket123",
        "ticketSubject": "Masalah login",
        "ticketStatus": "open",
        "ticketCategory": "technical",
        "senderId": "customer_uid_123",
        "senderName": "John Doe",
        "senderRole": "customer",
        "senderEmail": "john@example.com",
        "message": "Saya tidak bisa login...",
        "createdAt": "2026-01-20T14:30:00.000Z",
        "isRead": false
      }
    ],
    "totalMessages": 1,
    "filter": "all",
    "ticketId": null,
    "customerId": null
  }
}
```

PROF

3.2 GET Unread Messages

Kapan Digunakan:

- Notification badge (unread count)
- Priority inbox (messages belum dibaca)
- Real-time monitoring

```
# cURL
curl "http://localhost:3000/api/agent/messages?filter=unread"

# JavaScript
async function getUnreadMessages() {
  const response = await fetch('/api/agent/messages?filter=unread');
  const data = await response.json();
  console.log('Unread:', data.data.totalMessages);
  return data.data.messages;
}
```

3.3 GET Today's Messages

Kapan Digunakan:

- Daily dashboard
- Monitoring aktivitas hari ini
- Daily report

```
# cURL
curl "http://localhost:3000/api/agent/messages?filter=today"

# JavaScript
async function getTodayMessages() {
  const response = await fetch('/api/agent/messages?filter=today');
  const data = await response.json();
  return data.data.messages;
}
```

PROF

3.4 GET Messages from Specific Ticket

Kapan Digunakan:

- Melihat semua pesan customer dalam ticket tertentu
- Similar dengan `/api/tickets/[ticketId]/messages` tapi khusus customer messages saja

```
# cURL
curl "http://localhost:3000/api/agent/messages?ticketId=abc123"

# JavaScript
async function getTicketCustomerMessages(ticketId) {
  const response = await fetch(`/api/agent/messages?ticketId=${ticketId}`);
  const data = await response.json();
  return data.data.messages;
}
```

3.5 GET Messages from Specific Customer

Kapan Digunakan:

- Melihat history komunikasi dengan customer tertentu
- Customer profile view
- Track customer interaction

```
# cURL
curl "http://localhost:3000/api/agent/messages?customerId=uid123&limit=10"

# JavaScript
async function getCustomerHistory(customerId, limit = 50) {
  const url = `/api/agent/messages?customerId=${customerId}&limit=${limit}`;
  const response = await fetch(url);
  const data = await response.json();
  return data.data.messages;
}
```

3.6 Kombinasi Filter

```
// Unread messages hari ini
fetch('/api/agent/messages?filter=unread&filter=today')

// Messages dari ticket tertentu, max 20
fetch('/api/agent/messages?ticketId=abc123&limit=20')

// Messages dari customer tertentu, max 10, unread
fetch('/api/agent/messages?customerId=uid123&limit=10&filter=unread')
```

🗺️ Decision Flow - API Mana yang Harus Dipanggil?

Gunakan decision tree ini untuk memilih endpoint yang tepat:

PROF

```
📌 Apa yang ingin kamu dapatkan?

┌─ Daftar Tickets
│  ├─ Role: Customer
│  │  └─ GET /api/tickets
│  │     → Mendapatkan semua tickets milik sendiri
│  │
│  └─ Role: Agent
│     └─ GET /api/tickets?filter=...
│        ├─ filter=all → Semua tickets
│        ├─ filter=assigned → Tickets yang di-handle sendiri
│        └─ filter=unassigned → Tickets yang belum di-assign
│
├─ Pesan dalam Ticket Tertentu
│  ├─ Sudah tahu ticketId?
│  │  ├─ Ya → GET /api/tickets/[ticketId]/messages
│  │  │     → Mendapatkan semua pesan (customer + agent)
│   │   │
│   │   └─ Tidak → GET /api/tickets dulu untuk dapat ticketId
│   │
│   └─ Hanya pesan dari customer saja (Agent)?
│      └─ GET /api/agent/messages?ticketId=...
│         → Hanya pesan customer dalam ticket
│
└─ Inbox Agent / Customer Messages
   └─ Role: Agent Only
      └─ GET /api/agent/messages
         ├─ Tanpa parameter → Semua pesan customer
         ├─ ?filter=unread → Hanya yang belum dibaca
         ├─ ?filter=today → Pesan hari ini
         ├─ ?ticketId=... → Dari ticket tertentu
         ├─ ?customerId=... → Dari customer tertentu
         └─ ?limit=N → Batasi jumlah messages
```

📊 Use Case Examples

Use Case 1: Customer Dashboard - Daftar Tickets Saya

Goal: Tampilkan semua tickets yang pernah saya buat

API: `GET /api/tickets`

```
async function loadMyTickets() {
  const response = await fetch('/api/tickets');
  const data = await response.json();

  // Render tickets
  data.data.tickets.forEach(ticket => {
    console.log(`[${ticket.status}] ${ticket.subject} - ${ticket.messageCount}
messages`);
  });
}
```

PROF

Use Case 2: Customer Chat View - Baca Pesan dalam Ticket

Goal: Buka ticket dan lihat semua pesan conversation

API:

1. `GET /api/tickets` → Dapatkan daftar tickets
2. `GET /api/tickets/[ticketId]/messages` → Load messages

```
async function openTicketChat(ticketId) {
  const response = await fetch(`/api/tickets/${ticketId}/messages`);
  const data = await response.json();

  console.log('Ticket:', data.data.ticketInfo.subject);

  // Render chat
  data.data.messages.forEach(msg => {
    const alignment = msg.senderRole === 'customer' ? 'left' : 'right';
    console.log(`[${alignment}] ${msg.senderName}: ${msg.message}`);
  });
}
```

Use Case 3: Agent Dashboard - Semua Tickets Queue

Goal: Lihat semua tickets yang masuk, prioritaskan yang unassigned

API: `GET /api/tickets?filter=unassigned`

```
async function loadTicketQueue() {
  const response = await fetch('/api/tickets?filter=unassigned');
  const data = await response.json();

  // Sort by priority
  const sorted = data.data.tickets.sort((a, b) => {
    const priority = { urgent: 4, high: 3, medium: 2, low: 1 };
    return priority[b.priority] - priority[a.priority];
  });

  console.log('Queue:', sorted);
}
```

Use Case 4: Agent Inbox - Unread Customer Messages

Goal: Notification badge dengan jumlah pesan belum dibaca

API: `GET /api/agent/messages?filter=unread`

```
async function getUnreadCount() {
  const response = await fetch('/api/agent/messages?filter=unread');
  const data = await response.json();

  const count = data.data.totalMessages;

  // Update badge
  document.querySelector('.unread-badge').textContent = count;

  return count;
}

// Polling setiap 30 detik
setInterval(getUnreadCount, 30000);
```

PROF

Use Case 5: Agent - Customer History

Goal: Lihat semua komunikasi dengan customer tertentu

API: `GET /api/agent/messages?customerId=...`

```
async function viewCustomerHistory(customerId) {
  const response = await fetch(`/api/agent/messages?
customerId=${customerId}&limit=100`);
```

```
  const data = await response.json();

  console.log(`Total messages from customer: ${data.data.totalMessages}`);

  // Group by ticket
  const byTicket = data.data.messages.reduce((acc, msg) => {
    if (!acc[msg.ticketId]) acc[msg.ticketId] = [];
    acc[msg.ticketId].push(msg);
    return acc;
  }, {});

  console.log('Grouped by ticket:', byTicket);
}
```

🔄 Real-World Workflow Examples

Workflow 1: Customer Creates Ticket and Checks Status

```
// 1. Customer login (dapatkan token)
// 2. Create ticket
const createResponse = await fetch('/api/tickets/create', {
  method: 'POST',
  body: JSON.stringify({
    subject: 'Masalah login',
    message: 'Tidak bisa login ke aplikasi',
    priority: 'high'
  })
});
const { data: ticket } = await createResponse.json();
console.log('Ticket created:', ticket.ticketId);

// 3. Load my tickets (verify ticket muncul)
const ticketsResponse = await fetch('/api/tickets');
const { data: tickets } = await ticketsResponse.json();
console.log('My tickets:', tickets.tickets);

// 4. Load messages from ticket
const messagesResponse = await fetch(`/api/tickets/${ticket.ticketId}/messages`);
const { data: messages } = await messagesResponse.json();
console.log('Messages:', messages.messages);
```

PROF

Workflow 2: Agent Responds to Customer

```
// 1. Agent login
// 2. Get unread messages
const unreadResponse = await fetch('/api/agent/messages?filter=unread');
const { data: unread } = await unreadResponse.json();
console.log(`${unread.totalMessages} unread messages`);

// 3. Pick first unread message
const firstMessage = unread.messages[0];
const ticketId = firstMessage.ticketId;

// 4. Load full ticket conversation
```

```
const chatResponse = await fetch(`/api/tickets/${ticketId}/messages`);
const { data: chat } = await chatResponse.json();
console.log('Full conversation:', chat.messages);

// 5. Reply (POST)
const replyResponse = await fetch(`/api/tickets/${ticketId}/messages`, {
  method: 'POST',
  body: JSON.stringify({
    message: 'Terima kasih atas laporannya, kami akan cek'
  })
});

// 6. Refresh messages
const updatedChat = await fetch(`/api/tickets/${ticketId}/messages`);
console.log('Updated:', await updatedChat.json());
```

Workflow 3: Agent Daily Monitoring

```
// Morning routine: Check today's activity
async function agentMorningRoutine() {
  // 1. Get today's messages
  const todayResponse = await fetch('/api/agent/messages?filter=today');
  const { data: today } = await todayResponse.json();
  console.log(`Today: ${today.totalMessages} new messages`);

  // 2. Get unread messages
  const unreadResponse = await fetch('/api/agent/messages?filter=unread');
  const { data: unread } = await unreadResponse.json();
  console.log(`Unread: ${unread.totalMessages} messages`);

  // 3. Get unassigned tickets
  const queueResponse = await fetch('/api/tickets?filter=unassigned');
  const { data: queue } = await queueResponse.json();
  console.log(`Queue: ${queue.totalTickets} tickets`);

  return {
    todayMessages: today.totalMessages,
    unreadMessages: unread.totalMessages,
    queueTickets: queue.totalTickets
  };
}
```

PROF

⚡ Performance Tips

Tip 1: Gunakan Limit untuk Pagination

```
// Jangan load semua messages sekaligus
// BAD
fetch('/api/agent/messages'); // Bisa ratusan messages

// GOOD
fetch('/api/agent/messages?limit=50'); // Load 50 terbaru dulu
```

Tip 2: Filter yang Spesifik

```
// BAD: Load semua lalu filter di client
const all = await fetch('/api/agent/messages');
const unread = all.data.messages.filter(m => !m.isRead);

// GOOD: Filter di server
const unread = await fetch('/api/agent/messages?filter=unread');
```

Tip 3: Cache untuk Data yang Jarang Berubah

```
// Cache tickets list (refresh every 5 minutes)
let ticketsCache = null;
let cacheTime = 0;

async function getTickets(forceRefresh = false) {
  const now = Date.now();
  if (!forceRefresh && ticketsCache && (now - cacheTime < 300000)) {
    return ticketsCache;
  }

  const response = await fetch('/api/tickets');
  ticketsCache = await response.json();
  cacheTime = now;

  return ticketsCache;
}
```

📝 Summary - Cheat Sheet

Saya Ingin...	API yang Digunakan	Method	Role
Lihat tickets saya	`/api/tickets`	GET	Customer
Lihat semua tickets	`/api/tickets?filter=all`	GET	Agent
Lihat tickets unassigned	`/api/tickets?filter=unassigned`	GET	Agent
Baca chat dalam ticket	`/api/tickets/[id]/messages`	GET	Customer/Agent
Lihat inbox agent	`/api/agent/messages`	GET	Agent
Lihat pesan belum dibaca	`/api/agent/messages?filter=unread`	GET	Agent
Lihat pesan hari ini	`/api/agent/messages?filter=today`	GET	Agent
Lihat pesan dari ticket X	`/api/agent/messages?ticketId=X`	GET	Agent
Lihat pesan dari customer Y	`/api/agent/messages?customerId=Y`	GET	Agent

PROF

Quick Reference:

```
# Customer - My tickets
GET /api/tickets
```

```
# Customer - Chat in ticket
GET /api/tickets/{ticketId}/messages

# Agent - All tickets
GET /api/tickets?filter=all

# Agent - Inbox
GET /api/agent/messages

# Agent - Unread
GET /api/agent/messages?filter=unread
```

Support & Contact

Jika ada pertanyaan atau issues terkait API ini, silakan hubungi:

- **Email**: ameliaochamaharani1@gmail.com
- **GitHub**: Raharinda/ai-customerservice
- **Issues**: Check Firebase Console dan server logs terlebih dahulu

Helpful Resources

- Firebase Documentation
- Next.js Documentation
- Ticketing System Documentation
- Ticketing Setup Guide
- Troubleshooting Guide
- Agent Key Setup Guide

Last Updated: January 20, 2026