API Documentation - Authentication Service

Dokumentasi lengkap untuk API Authentication yang dapat digunakan oleh frontend developer.

Base URL

```
http://localhost:3000/api/auth
```

Untuk production, ganti dengan domain production Anda.

Table of Contents

PROF

Overview

Sistem autentikasi ini menggunakan **dual authentication** dengan endpoint terpisah untuk Customer dan Support Agent:

- **Customer**: Login normal dengan email/password atau Google
- **Support Agent**: Login dengan email/password + Agent Key khusus

Key Features

- ✅ Separate login endpoints untuk keamanan maksimal
- ✅ Agent key verification untuk support agent
- ✅ Role-based access control (customer/agent)
- ✅ Firebase Authentication integration
- ✅ Google Sign-in support (customer only)
- ✅ Automatic role assignment
- ✅ Server-side token verification

Architecture

Routes Structure

```
/login                    → Customer login page
/customer                 → Customer dashboard (protected)
/agent                    → Agent login page
/agent/dashboard          → Agent dashboard (protected)
```

API Endpoints

```
POST /api/auth/register          → User registration
POST /api/auth/login             → Customer login
POST /api/auth/agent/login       → Agent login (with key verification)
POST /api/auth/verify            → Token verification
```

Authentication Flow

```
User → Homepage
  ├─> Customer
  │      ├─> Navigate to /login
  │      ├─> Email/Password or Google
  │      ├─> API: /api/auth/login
  │      └─> Redirect to /customer
  │
  └─> Support Agent
         ├─> Navigate to /agent
         ├─> Email/Password + Agent Key
         ├─> API: /api/auth/agent/login
         ├─> Verify agent key
         ├─> Update role to 'agent'
         └─> Redirect to /agent/dashboard
```

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"
}
```

PROF

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) {
    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"
}
```

PROF

Request Body

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

Request Body Parameters

Parameter	Type	Required	Description

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

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`

PROF

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;
  }
}
```

---

3. Agent Login (with Key)

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

PROF

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)

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"
  }
}
```

PROF

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

PROF

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');
    }
```

PROF

```
    return data;
  } 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"
}
```

PROF

Request Body

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

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`

PROF

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) {
      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;
  }
}
```

PROF

---

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

```
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
  }
}
```

PROF

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;
  }
}
```

Error Responses

Semua error response mengikuti format yang konsisten:

PROF

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

Code	Description	Contoh Use Case
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

Frontend Integration Examples

React/Next.js Implementation

1. Setup Firebase (Client-Side)

PROF

```
// 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;
```

PROF

```
  };

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

  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('');
```

PROF

```
    setLoading(true);

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

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

PROF

```
// 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));
```

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

Priority Levels:

- `low` - Pertanyaan umum
- `medium` - Masalah biasa
- `high` - Masalah penting
- `urgent` - Sangat mendesak

---

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",
  "priority": "high"
}
```

Request Body Parameters

Parameter	Type	Required	Description
subject	string	Yes	Judul ticket (minimal 5 karakter)
message	string	Yes	Pesan awal (minimal 10 karakter)
priority	string	No	Priority level (default: "medium")

Priority Values:

- `low` - Pertanyaan umum, tidak mendesak
- `medium` - Masalah biasa (default)
- `high` - Masalah penting
- `urgent` - Sangat mendesak

PROF

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",
    "priority": "high",
    "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`

Subject terlalu pendek

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

Code: `400 Bad Request`

Message terlalu pendek

PROF

```
{
  "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, priority = 'medium') {
  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,
        priority,
      }),
    });

    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...',
  'high'
)
  .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.

PROF

Success Response (Customer)

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}`,
      },
    });
```

PROF

```
    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')
  .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
```

PROF

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..."
}
```

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
  }
}
```

PROF

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"
}
```

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;
  }
}
```

PROF

```
// 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

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) {
```

PROF

```
      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}`}
          >
            <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>
  );
}
```

PROF

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

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

PROF

✅ 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

✅ 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

PROF

✅ 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

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"
  }'
```

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"
  }'
```

PROF

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"
```

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`

PROF

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

PROF

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

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

PROF

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 3.0.0 (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

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