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

---

1. Register (Email/Password)

Mendaftarkan user baru dengan email dan password.

Endpoint

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

PROF

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

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

PROF

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

PROF

---

2. Login (Email/Password)

Login user 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

Success Response

Code: `200 OK`

PROF

```
{
  "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`

Email tidak terdaftar

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

Code: `404 Not Found`

Validasi gagal

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

Code: `400 Bad Request`

Frontend Example (JavaScript)

```
// Login dengan email/password
async function loginUser(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);
    }
```

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

---

3. Login with Google

Login atau register otomatis menggunakan akun Google. Tidak perlu endpoint terpisah untuk register karena Google OAuth akan otomatis membuat user baru jika belum ada.

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;
  } 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`

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

PROF

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

Code	Description	Contoh Use Case
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

Frontend Integration Examples

React/Next.js Implementation

PROF

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();
  }, []);

  // 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);
  };

  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
          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>
  );
}
```

PROF

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));
```

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
```

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

2. Password Requirements

- ✅ Minimal 6 karakter (dapat ditingkatkan)
- ✅ Hindari password yang umum
- ✅ Implementasi rate limiting untuk login

3. CORS Configuration

- ✅ Pastikan domain frontend ada di authorized domains Firebase
- ✅ Set proper CORS headers di backend

4. Error Handling

PROF

- ✅ Jangan expose informasi sensitif dalam error message
- ✅ Log error di backend untuk debugging
- ✅ Tampilkan user-friendly message di frontend

Testing

Manual Testing Checklist

Email/Password Authentication

- [ ] 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

Google Authentication

- [ ] Google popup muncul dengan benar
- [ ] Login dengan akun Google berhasil
- [ ] User baru otomatis terdaftar
- [ ] User existing dapat login
- [ ] Error handling untuk popup closed
- [ ] Error handling untuk popup blocked

Token Verification

- [ ] Token valid diverifikasi dengan benar
- [ ] Token invalid ditolak
- [ ] Token expired ditolak

PROF

Support & Contact

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

- **Email**: ameliaochamaharani1@gmail.com
- **GitHub**: Raharinda/ai-customerservice

Changelog

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

Last Updated: January 18, 2026