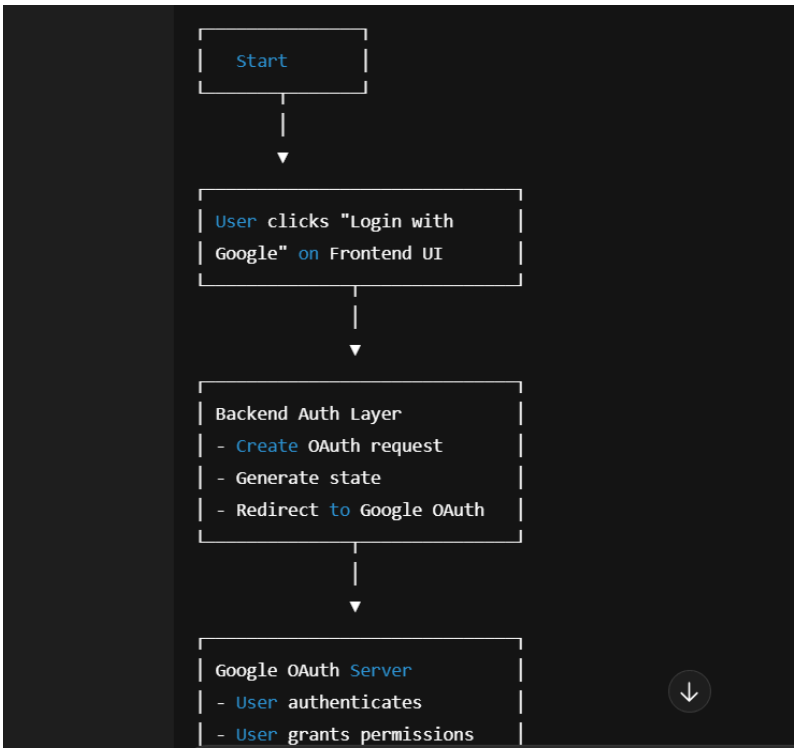
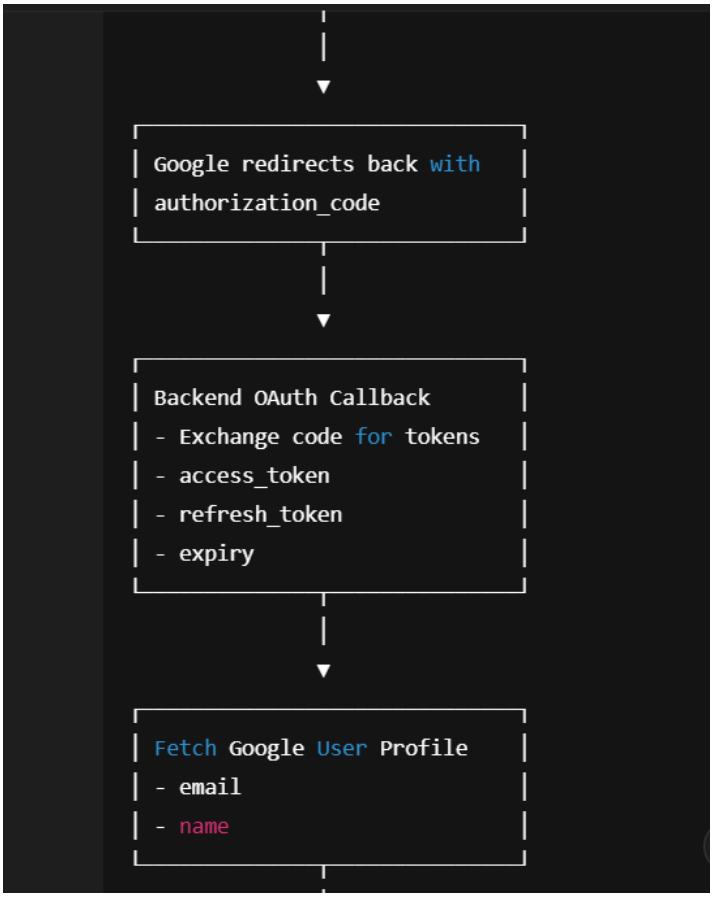
Google OAuth Authentication Flow

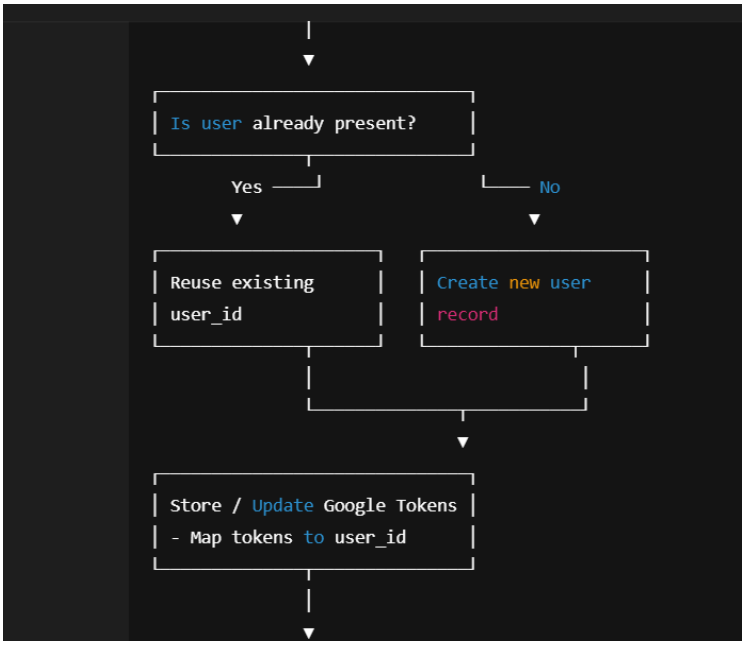
Overview

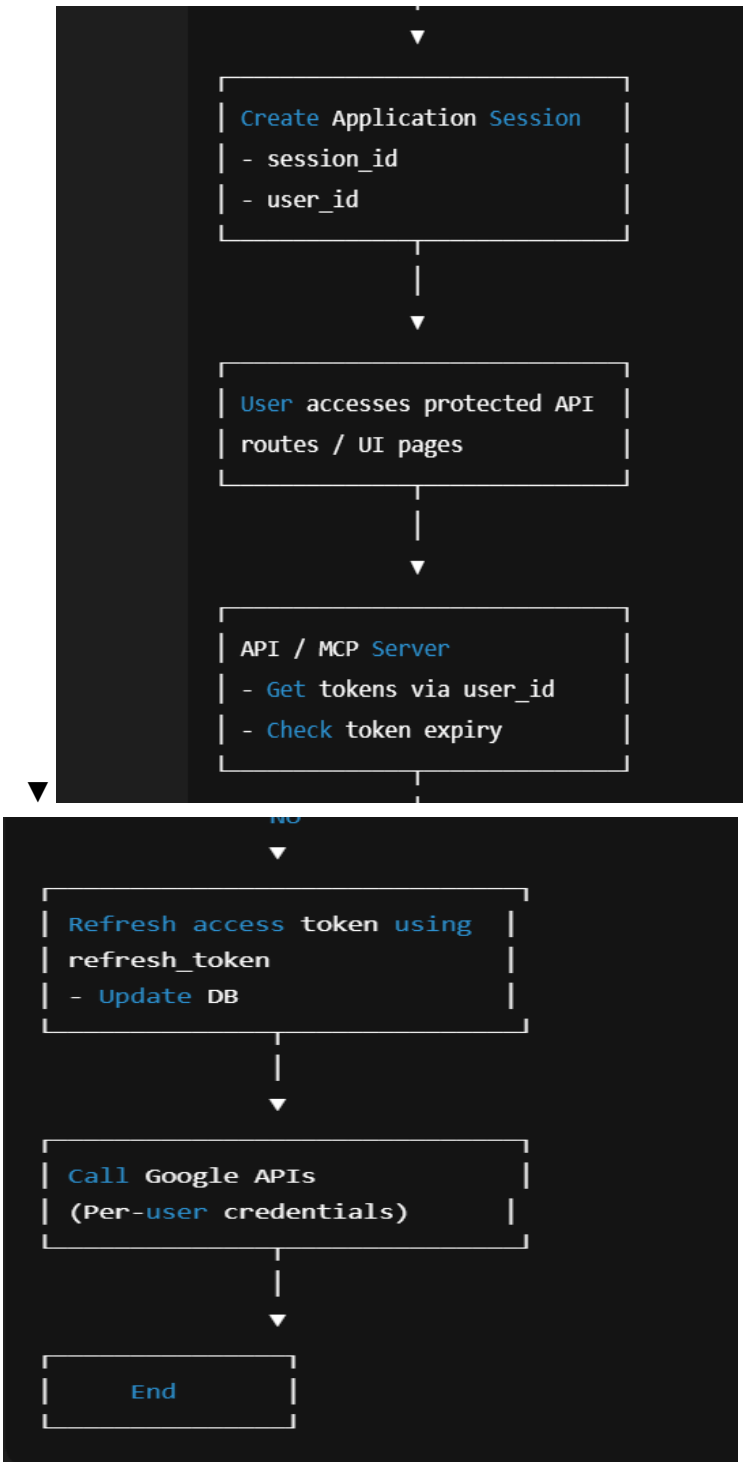
This system implements a **multi-user Google OAuth authentication flow** where each Google account maps to a unique internal user in the application. OAuth credentials (access and refresh tokens) are stored per user in the database, and each user interacts with Google APIs using their own credentials. The design supports multiple users and multiple sessions per user.

Architecture Flow Diagram for Google OAuth Authentication per user




Google redirects back with
authorization_code
Backend OAuth Callback
- Exchange code for tokens
- access_token
- refresh_token
- expiry
Fetch Google User Profile
- email
- name

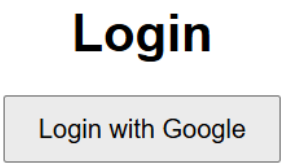

Is user already present?
Yes
No
Reuse existing
user_id
Create new user
record
Store / Update Google Tokens
- Map tokens to user_id


Create Application Session
- session_id
- user_id
User accesses protected API routes / UI pages
API / MCP Server
- Get tokens via user_id
- Check token expiry
Refresh access token using refresh_token
- Update DB
Call Google APIs (Per-user credentials)
End

Authentication & Session Flow

1. Login Initiation

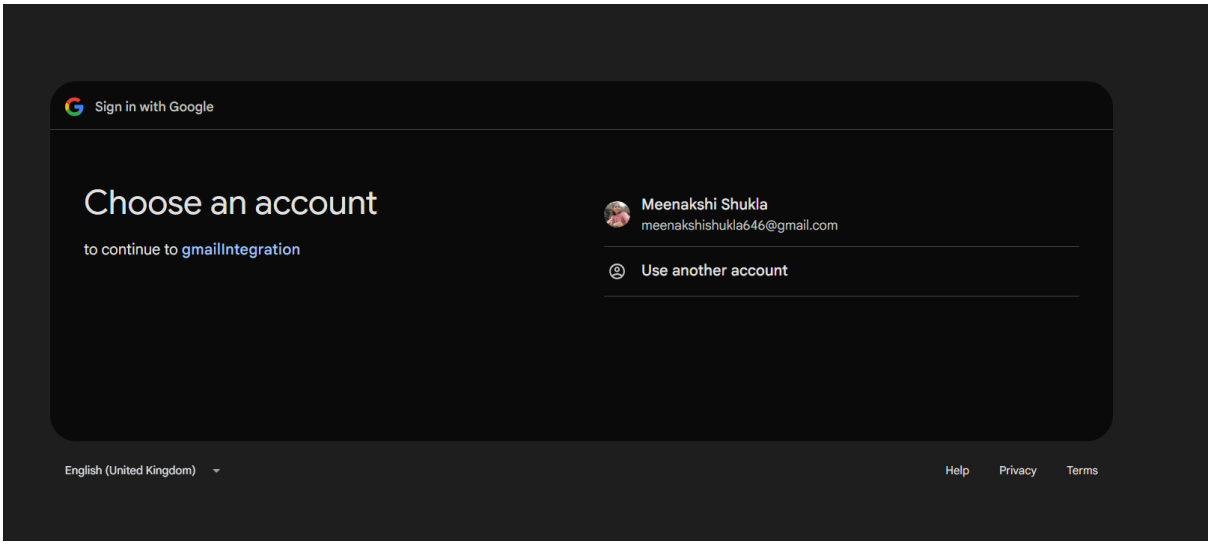
- User clicks **“Login with Google”** on the frontend
- User is redirected to the Google OAuth consent screen with required scopes

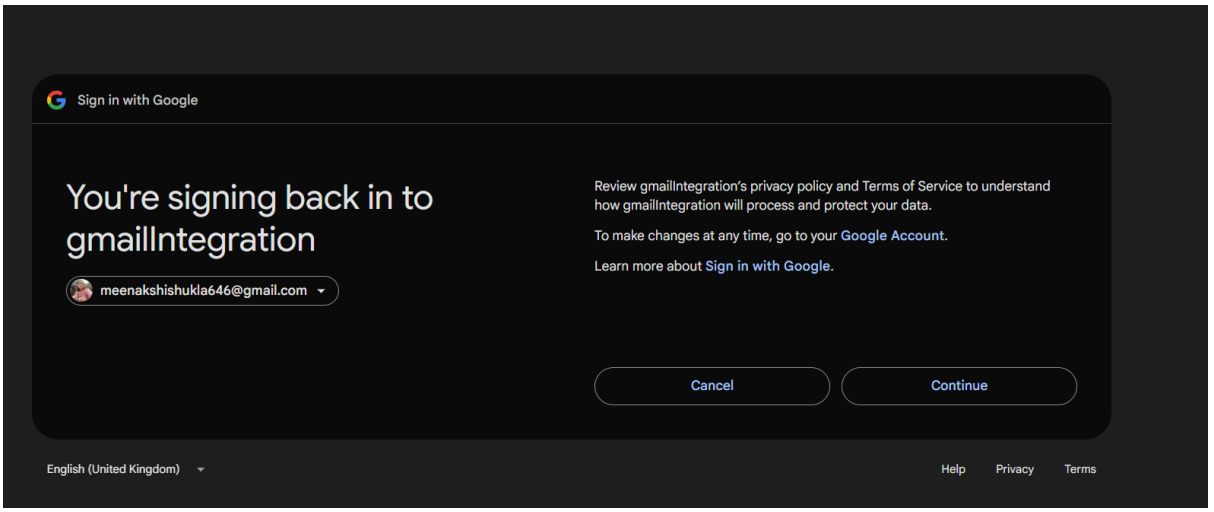


---

2. Google Authentication

- User authenticates and grants permissions on Google
- Google redirects back to the backend callback URL with an **authorization code**





---

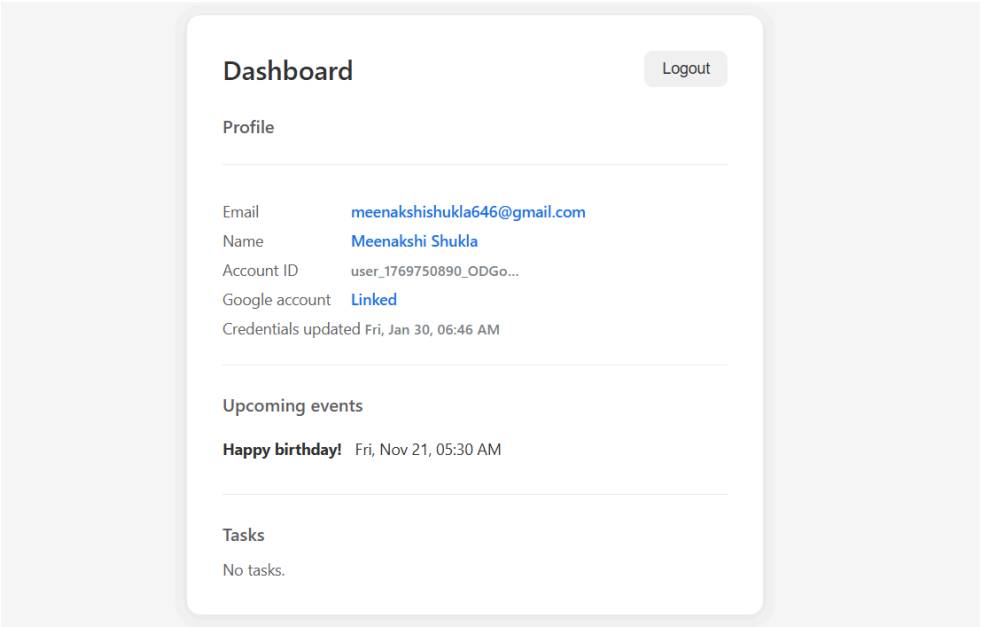
3. OAuth Callback Handling (Backend)

The backend performs the following steps:

- Exchanges the authorization code for:
    - `access_token`
    - `refresh_token`
    - `expires_in`

- Fetches user profile information (email and name) from Google
- Checks if the user already exists in the database using the email
    - If user exists → reuse existing `user_id`
    - If user does not exist → create a new user record
- Stores or updates Google OAuth credentials (access token, refresh token, expiry) for the user
- Creates an application session mapped to the `user_id`

At this point, the user is authenticated within the system.



Session-Based Access Control

- Each session stores the authenticated `user_id`
- All protected routes validate:
    - Session existence

    - Session expiry
- Data access is always scoped to the current `user_id`, ensuring user isolation

---

Google API Access

- For any Google API request:
    - The system fetches OAuth credentials using `session.user_id`
    - Each user's API calls use **their own Google access token**
- Tokens are never shared between users

Token Refresh Handling

When an access token expires:

- The system checks whether a refresh token exists
- If present:
    - Automatically requests a new access token from Google
    - Updates the token and expiry in the database
    - Continues the API request transparently
- If refresh fails:
    - Tokens are revoked
    - Stored credentials are deleted
    - User is logged out and required to re-authenticate

This ensures secure and uninterrupted access while preventing invalid token reuse.

Logout & Expiry Scenarios

The system handles multiple logout and expiry cases:

- **Normal Logout**
    - Clears the application session
    - Google tokens may optionally be revoked
- **Forced Logout**
    - Revokes Google access and refresh tokens
    - Deletes stored credentials
    - Clears the session completely
- **Session Expiry**
    - Clears the application session
    - Google credentials remain stored
    - User can log in again without re-consent (if refresh token is still valid)

Database Design

users

- Stores one record per Google account
- Identified uniquely by Google email
- Contains basic profile information

google_credentials

- Stores OAuth credentials per user
- Includes access token, refresh token, and expiry

- One row per user (extendable to per-device if required)

sessions

- Tracks active user sessions
- Supports multiple concurrent sessions per user
- Enforces session expiry independently of OAuth token lifetime

---

Supported Scenarios

- Multiple users logged in simultaneously
- Same user logged in from multiple devices
- User-specific Google credentials
- Transparent access-token refresh
- Secure logout and cleanup flow

Feature of this Apporach

- Per-user authentication
- Secure OAuth flow
- No shared credentials
- Scalable for many users