Hands-On Lab Guide: OAuth 2.0 and OpenID Connect on Google Cloud

Objective: Build a Node.js web app that allows users to log in with Google, fetches their profile, and demonstrates OAuth 2.0 + OpenID Connect.

Step 1 — Create a Google Cloud Project

Interface: Google Cloud Console (web browser)

Instructions:

1. Open Google Cloud Console.
2. At the top, click **Select a project → New Project**.
3. Enter **Project name:** `OAuth Demo App`.
4. Optionally select a billing account (not required for free tier).
5. Click **Create**.

Explanation:

Each project in Google Cloud isolates resources, credentials, and APIs. OAuth credentials must belong to a project.

Step 2 — Configure OAuth Consent Screen

Interface: Google Cloud Console

Instructions:

1. In the left menu, go to **APIs & Services → OAuth consent screen**.
2. Select **External** (for personal Google accounts).
3. Fill in the following fields:
    - **App name**: e.g., OAuth Demo App
    - **User support email**: your email
    - **Developer contact email**: your email
4. Click **Save and Continue**.

Explanation:

The consent screen is what users see when your app requests access to their Google account. It ensures transparency and trust.

Step 3 — Create OAuth 2.0 Credentials

Interface: Google Cloud Console

Instructions:

1. Navigate to **APIs & Services → Credentials → Create Credentials → OAuth client ID**.
2. If prompted, configure the OAuth consent screen (see Step 2).
3. For **Application type**, select **Web application**.
4. Add the following:
   - **Authorized JavaScript origins:**
   - `http://localhost:8080`
   - **Authorized redirect URIs:**
   - `http://localhost:8080/callback`
5. Click **Create**.
6. Copy **Client ID** and **Client Secret**. Keep them safe.

Explanation:

OAuth client ID and secret identify your app to Google. The redirect URI ensures tokens are only sent to authorized locations.

Step 4 — (Optional) Enable People API

Interface: Google Cloud Console

Instructions:

1. Go to **APIs & Services → Library**.
2. Search for **People API**.
3. Click **Enable**.

Explanation:

The People API allows fetching additional profile info (phone number, contacts). Basic OIDC login provides name, email, and picture without enabling this API.

Step 5 — Build the Node.js Web App

Interface: Terminal / Code Editor

Step 5a — Initialize Project in Terminal

```
mkdir gcp-oauth-demo
```

```
cd gcp-oauth-demo
npm init -y
npm install express axios
```

Explanation:

Creates a Node.js project folder and installs required packages: Express (server) and Axios (HTTP requests).

Step 5b — Create `index.js` in a Code Editor

```
const express = require("express");
const axios = require("axios");
const app = express();

const CLIENT_ID = "YOUR_CLIENT_ID_HERE";
const CLIENT_SECRET = "YOUR_CLIENT_SECRET_HERE";
const REDIRECT_URI = "http://localhost:8080/callback";

// Redirect to Google login
app.get("/", (req, res) => {
  const authUrl =
`https://accounts.google.com/o/oauth2/v2/auth?client_id=${CLIENT_ID}&redire
ct_uri=${REDIRECT_URI}&response_type=code&scope=openid%20email%20profile`;
  res.send(`<a href="${authUrl}">Login with Google</a>`);
});

// Callback endpoint to exchange code for tokens
app.get("/callback", async (req, res) => {
  const code = req.query.code;

  const tokenResponse = await
axios.post("https://oauth2.googleapis.com/token", {
    code,
    client_id: CLIENT_ID,
    client_secret: CLIENT_SECRET,
    redirect_uri: REDIRECT_URI,
    grant_type: "authorization_code",
  });

  const { id_token, access_token } = tokenResponse.data;

  const userResponse = await
axios.get("https://openidconnect.googleapis.com/v1/userinfo", {
    headers: { Authorization: `Bearer ${access_token}` },
  });

  res.send(`
    <h1>Welcome, ${userResponse.data.name}</h1>
    <p>Email: ${userResponse.data.email}</p>
    <img src="${userResponse.data.picture}" width="100" />
    <pre>${JSON.stringify(userResponse.data, null, 2)}</pre>
  `);
});

app.listen(8080, () => console.log("Server running at
http://localhost:8080"));
```

Explanation:

- `/` route redirects users to Google login.
- `/callback` route exchanges the code for tokens and fetches user info.
- Access token allows API calls; ID token verifies identity.

Step 5c — Run the App

Terminal:

```
node index.js
```

Browser: Open `http://localhost:8080` → click **Login with Google** → approve consent.

Explanation:

You will see your Google profile info displayed. This demonstrates a full OAuth + OIDC flow.

Step 6 — Verify ID Token (Optional but Recommended)

Interface: Terminal / Code Editor

Step 6a — Install Verification Libraries

```
npm install jsonwebtoken jwks-rsa
```

Step 6b — Add Verification Code

```
const jwt = require("jsonwebtoken");
const jwksClient = require("jwks-rsa");

const client = jwksClient({ jwksUri:
"https://www.googleapis.com/oauth2/v3/certs" });

function getKey(header, callback) {
  client.getSigningKey(header.kid, (err, key) => callback(null,
key.getPublicKey()));
}

jwt.verify(id_token, getKey, { audience: CLIENT_ID, issuer:
"https://accounts.google.com" }, (err, decoded) => {
  if (err) console.error("Token verification failed:", err);
  else console.log("Verified ID Token:", decoded);
});
```

Explanation:

Verifying ID tokens ensures they are issued by Google, intended for your app, and not tampered with. Critical for production apps.

Step 7 — Summary

- Google Cloud project created.
- OAuth consent screen configured.
- OAuth 2.0 credentials created (Client ID & Secret).
- Node.js web app implemented to handle OAuth + OIDC.
- Optional ID token verification implemented for security.