解決 Google 登入狀態無法維持的技術建議

問題分析

當使用者透過 Google 登入後，網站無法保持其登入狀態。這通常與伺服器端的 Session（會話）和瀏覽器端的 Cookie（餅乾）設定不正確有關。

先前採用客戶端（前端）透過 API (`/api/auth/user`) 不斷向後端確認登入狀態的方式，會造成系統不必要的重複請求負擔。改用 Cookie 來處理是正確的方向，但登入狀態的遺失，很可能是 Cookie 或 Session 在設定或驗證環節出了問題。

標準的登入狀態維持流程

1. 登入請求：使用者透過 Google 登入成功後，伺服器端會從 Google 得到使用者的個人資料。
2. 建立 **Session**：伺服器端為該使用者建立一個 Session，並將使用者的部分資料（例如使用者 ID）存入這個 Session 中。
3. 發送 **Cookie**：伺服器產生一個獨一無二的 Session ID，並將這個 Session ID 透過 Cookie 的方式傳送給瀏覽器儲存。
4. 後續請求：此後，瀏覽器對該網站的任何請求，都會自動帶上這個 Cookie。
5. 狀態驗證：伺服器端收到請求後，根據 Cookie 中的 Session ID 找到對應的 Session，從而確認使用者的登入狀態與身份。

建議的解決方案 (以 Node.js + Express + Passport.js 為例)

這是在 Node.js 環境中處理第三方登入的常見且穩定的技術組合。您可以對照以下範例來檢查並修正您的程式碼。

1. 安裝必要的套件

請確保您的專案已安裝 `express-session` 和 `passport` 相關套件。

```
npm install express-session passport passport-google-oauth20
```

2. 設定 Express Session 中介軟體

這是維持登入狀態的核心。在您的主應用程式檔案（例如 `app.js` 或 `server.js`）中進行設定。

```
const express = require('express');
const session = require('express-session');
const passport = require('passport');

const app = express();

app.use(session({
  secret: 'your_secret_key', // 這是一個用來簽章 session ID cookie 的密鑰，請務必更換成您自己的密鑰
  resave: false,
  saveUninitialized: false, // 建議設定為 false，避免儲存空的 session
  cookie: {
    secure: false, // 如果您的網站是 HTTPS，請設定為 true
    maxAge: 24 * 60 * 60 * 1000 // cookie 的有效期限 (這裡是 24 小時)
  }
}));

// 初始化 Passport
app.use(passport.initialize());
app.use(passport.session());

// ... 您的其他程式碼
```

重點檢查項目

- `secret`: 必須設定一個複雜且唯一的字串，以確保安全性。
- `cookie.secure`: 如果您的網站不是透過 HTTPS 存取（例如在本地開發時），此值必須為 `false`。若設定為 `true`，瀏覽器將不會在 HTTP 連線下儲存 Cookie，直接導致登入狀態遺失。

3. 設定 **Passport Google** 策略

設定 Passport 以便使用 Google OAuth 2.0 進行驗證。

```
const GoogleStrategy = require('passport-google-oauth20').Strategy;

passport.use(new GoogleStrategy({
  clientID: 'YOUR_GOOGLE_CLIENT_ID',
  clientSecret: 'YOUR_GOOGLE_CLIENT_SECRET',
  callbackURL: '/auth/google/callback'
 },
 function(accessToken, refreshToken, profile, done) {
  // 此處您可以將使用者資料 (profile) 存入資料庫
  // 接著呼叫 done() 來完成驗證流程
  return done(null, profile);
 }
));
```

```
// 序列化使用者 (將使用者資料存入 session)
passport.serializeUser(function(user, done) {
  done(null, user);
});

// 反序列化使用者 (從 session 中讀取使用者資料)
passport.deserializeUser(function(user, done) {
  // 您可以根據 session 中的使用者資訊，從資料庫找到完整的使用者資料
  done(null, user);
});
4. 建立登入與回呼的路由 (Route)
// 此路由會將使用者導向 Google 登入頁面
app.get('/auth/google',
  passport.authenticate('google', { scope: ['profile', 'email'] }));

// Google 登入成功後，會將使用者導向此回呼 URL
app.get('/auth/google/callback',
  passport.authenticate('google', { failureRedirect: '/login' }), // 失敗時導向登入頁
  function(req, res) {
    // 登入成功，將使用者重新導向到指定頁面
    res.redirect('/profile');
  });
```

5. 建立受保護的路由

對於需要登入才能存取的服務或頁面，可以建立一個中介軟體來檢查使用者登入狀態。

```
function ensureAuthenticated(req, res, next) {
  // isAuthenticated() 是 Passport.js 提供的函式
  if (req.isAuthenticated()) {
    return next(); // 已登入，繼續執行下一個中介軟體或路由處理函式
  }
  // 未登入，導向登入頁
  res.redirect('/login');
}

// /profile 是一個受保護的頁面
app.get('/profile', ensureAuthenticated, (req, res) => {
  // 如果能執行到這裡，代表使用者已經成功登入
  // req.user 包含了登入者的資料 (來自 deserializeUser)
  res.json({ user: req.user });
});
```

常見問題與除錯方向

1. **Cookie** 未被設定：

    - 檢查瀏覽器開發者工具 (F12) 的 "Application" (或 "Storage") -> "Cookies" 分頁，確認網站是否有成功設定 `connect.sid` 或您自訂的 Cookie 名稱。
    - 再次確認 `cookie.secure` 的設定是否與您的環境（HTTP/HTTPS）相符。
2. 反向代理 **(Reverse Proxy)** 問題：
    - 如果您的 Node.js 應用程式是部署在 Nginx 或其他反向代理之後，您可能需要在 Express 中設定信任代理，以確保 `secure cookie` 能正常運作：`app.set('trust proxy', 1);`。
3. **Google Cloud Console** 設定錯誤：
    - 請再三確認您在 Google Cloud Console 中設定的「已授權的重新導向 URI」(`Authorized redirect URIs`) 是否與您程式碼中的 `callbackURL` 完全一致（包含 `http://` 或 `https://` 以及結尾的斜線）。