1. `GET /api/auth/google/callback?code=...` 請求成功，並且伺服器回應了 `302 Found`，這表示 Google 已經成功驗證使用者並將其重新導向回您的應用程式。
2. 緊接著，瀏覽器被重新導向到 `/` (首頁)。
3. 然後，前端馬上發出了一個 `GET /api/auth/user` 的請求。
4. 伺服器對於 `/api/auth/user` 的請求回應了 `401 Unauthorized`，這表示此時伺服器認為使用者是未登入狀態。

這個流程清楚地表明：問題的核心在於 **`express-session`** 並沒有成功地為使用者建立並維持 **session**。

在 Google 驗證成功並回呼 (`callback`) 之後，`passport.serializeUser` 應該要被觸發，並將使用者資訊存入 session，同時設定一個 session cookie 給瀏覽器。但從 `/api/auth/user` 回應 401 的情況來看，這個過程失敗了。當前端回頭詢問「我是誰」的時候，後端因為找不到有效的 session，所以回答「我不認識你」。

## 根本原因與解決方案

經過詳細檢查您的程式碼 (`server/index.ts` 和 `server/googleAuth.ts`)，我找到了問題的根源。

在您的 `server/index.ts` 中，您使用了 `cookie-session` 這個套件，而不是我之前範例中提到的 `express-session`。雖然兩者用途相似，但 `cookie-session` 的運作方式是將整個 **session** 的資料加密後直接儲存在 **cookie** 中，而不是只在 cookie 儲存一個 session ID。

問題出在 `cookie-session` 的設定與 Passport 的整合上。

修正步驟：

**1.** 移除 **`cookie-session`**，改用 **`express-session`**

`express-session` 是與 Passport.js 整合更常見且文件更豐富的選擇。它將 session 資料儲存在伺服器端（預設是記憶體，也可以是 Redis 或資料庫），只傳送一個 session ID 給客戶端，這通常是更好的實踐。

首先，請安裝 `express-session`：

```
npm install express-session
npm uninstall cookie-session
# 如果您需要 TypeScript 的類型定義
npm install @types/express-session --save-dev
```

**2.** 修改 **`server/index.ts`**

請將 `server/index.ts` 中關於 session 的部分替換掉。

找到並刪除這段程式碼：

```
// server/index.ts (舊的程式碼，請刪除)
import cookieSession from 'cookie-session';
// ...
app.use(
  cookieSession({
    name: 'session',
    keys: [process.env.COOKIE_KEY!],
    maxAge: 24 * 60 * 60 * 1000, // 24 hours
  }),
);
```

用以下程式碼替換：

```
// server/index.ts (新的程式碼)
import session from 'express-session';
// ...

// 確保這個密鑰存在於您的 .env 檔案中
if (!process.env.SESSION_SECRET) {
  console.error("FATAL ERROR: SESSION_SECRET is not defined.");
  process.exit(1);
}

app.use(session({
  secret: process.env.SESSION_SECRET, // 在您的 .env 檔案中新增一個 SESSION_SECRET
  resave: false,
  saveUninitialized: false, // 這點很重要，避免在沒有登入時也建立 session
  cookie: {
    maxAge: 24 * 60 * 60 * 1000, // 24 hours
    secure: process.env.NODE_ENV === 'production', // 在生產環境(HTTPS)中設為 true
    httpOnly: true, // 增加安全性
  }
}));

// 初始化 Passport，務必放在 app.use(session(...)) 之後
app.use(passport.initialize());
app.use(passport.session());
```

**3.** 在您的 **`.env`** 檔案中新增一個 **`SESSION_SECRET`**

這個密鑰是用來加密 session ID 的，請務必設定。

```
# .env # ... 其他的 key ... COOKIE_KEY=...
SESSION_SECRET=uX2cSPyxjQ2bpmDHdb+nRclE
```

## 為什麼這樣可以解決問題？

- 標準化整合：`express-session` 和 `passport` 的整合是經過長期驗證的標準作法。`passport.session()` 中介軟體就是被設計來與 `express-session`（或相容的 session store）協同工作的。
- 正確的執行順序：`app.use(session(...))` -> `app.use(passport.initialize())` -> `app.use(passport.session())`。這個順序至關重要。`passport.session()` 會從 `express-session` 所建立的 `req.session` 物件中讀取登入的使用者資訊，並將其還原到 `req.user`。
- 伺服器端儲存：透過在伺服器端儲存 session，cookie 的大小會保持很小，並且不會將敏感的使用者資訊（即使是加密的）暴露在客戶端。

完成以上修改後，請重新啟動您的伺服器。完整的登入流程應該會是：

1. 使用者點擊登入，導向 Google。
2. Google 驗證成功，導回 `/api/auth/google/callback`。
3. Passport 驗證 `code`，取得使用者資料。
4. `serializeUser` 被呼叫，將使用者 ID 存入伺服器端的 session。
5. `express-session` 將 session ID 透過 `Set-Cookie` header 傳送給瀏覽器。
6. 瀏覽器收到回應後，儲存 session cookie。
7. 當前端發送 `/api/auth/user` 請求時，會自動帶上這個 cookie。
8. 伺服器收到請求，`express-session` 透過 cookie 找到對應的 session，`passport.session()` 接著從 session 中還原使用者資料到 `req.user`。
9. `/api/auth/user` 的處理函式檢查到 `req.isAuthenticated()` 為 `true`，成功返回使用者資料。