Google Authentication

Copyright © 2025 by
Robert M. Dondero, Ph.D.
Princeton University

Google Authentication

- Google authentication
  - **Part 1**: User logs into Google server
    - Unnecessary if user is already logged into Google server
    - User must provide credentials
  - **Part 2**: User logs into PennyAdmin
    - User need not provide credentials

Google Authentication

- See **PennyAdminGoogle** app

Agenda

- Google authentication
  - **The code**
  - Running locally
  - Running on Render
  - How it works
  - Assessment

The Code

- See **PennyAdminGoogle** app (cont.)
  - See the codegoogle.zip file on the course website
    - runserver.py
    - penny.sql, penny.sqlite
    - database.py
    - header.html, footer.html
    - index.html, show.html,
    - add.html, delete.html, reportresults.html
    - top.py, penny.py, auth.py

Agenda

- Google authentication
  - The code
  - **Running locally**
  - Running on Render
  - How it works
  - Assessment

Running Locally

- See **PennyAdminGoogle** app (cont.)
  - How to run it on your local computer…

Running Locally

1. Create a project Google account (that is, a gmail address) for your project team. Use your project Google account exclusively for Google authentication setup and subsequent app testing on your local computer. For your security, don't use your personal Google account.

Running Locally

2. Register your app (**https://localhost:5000**) with Google.
    2.1. Log into Google using your project Google account
    2.2. Browse to
        https://console.developers.google.com/apis/credentials
    2.3. Click CREATE PROJECT
    2.4. For Project name enter Penny
    2.5. Click CREATE
    2.6. Click CONFIGURE CONSENT SCREEN
    2.7. For User Type choose External
    2.8. Click CREATE
    2.9. For App name enter Penny
    2.10. For User support email enter your your project gmail address
    2.11. For Developer contact information enter your project gmail address
    2.12. Click SAVE AND CONTINUE a few times to finish the consent
    2.13. Click Credentials
    2.14. Click Create Credentials, OAuth client ID, Web Application

Running Locally

2.15. In Authorized JavaScript origins:
    2.15.1. Click ADD URI
    2.15.2. Enter **https://localhost:5000**
    2.15.3. Typically you also would add a URI for your app on Render or Heroku.
2.16. In Authorized redirect URIs:
    2.16.1. Click ADD URI
    2.16.2. Enter **https://localhost:5000/login/callback**
    2.16.3. Typically you also would add a callback URI for your app on Render or Heroku.
2.17. Google provides **GOOGLE_CLIENT_ID** and **GOOGLE_CLIENT_SECRET**. Take note of them!

Running Locally

3. Create and activate an appropriate Python virtual environment. To do that, issue these commands:

```
mkdir ~/.virtualenvs
python -m venv ~/.virtualenvs/somedir
source ~/.virtualenvs/somedir/bin/activate
```

4. Install modules into your Python virtual environment. To do that, in your app directory issue this command:

```
python -m pip install -r requirements.txt
```

Running Locally

5. Create a self-signed certificate consisting of files **key.pem** and **cert.pem** in your app directory. To do that, issue this command on a Unix-like computer:

```
openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem
   -keyout key.pem -days 365
     Country Name (2 letter code) [AU]US
     State or Province Name (full name) [Some-State]NJ
     Locality Name (eg, city) []:Princeton
     Organization Name (eg, company) [Internet Widgits
     Pty Ltd]:Princeton University
     Organizational Unit Name (eg, section) []:
     Common Name (e.g. server FQDN or YOUR name)
     []:localhost
     Email Address []:
```

Running Locally

6. Define these environment variables. It would be common to define them in a **.env** file in your app directory:

   GOOGLE_CLIENT_ID=*yourGoogleClientId*
   GOOGLE_CLIENT_SECRET=*yourGoogleClientSecret*

5. Run the server. To do that, in your app directory issue this command:

   python runserver.py

6. Browse to **https://localhost:5000**

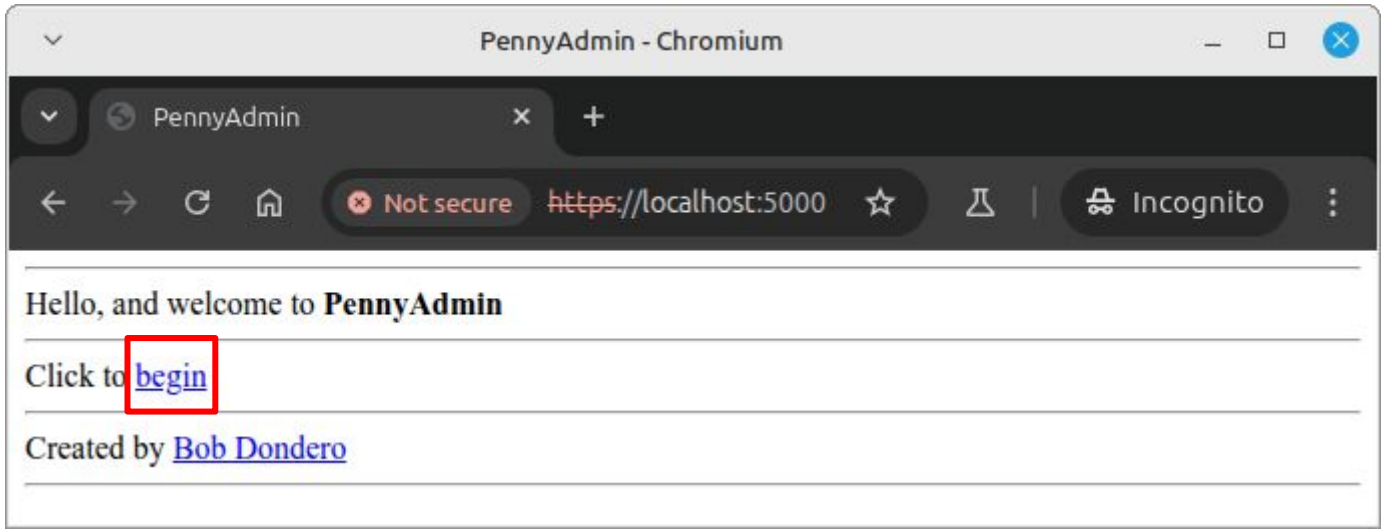
   You must use **localhost** as the host. Using the real IP address of your computer won't work. Using **127.0.0.1** won't work.

7. Authenticate using Google.

8. When prompted for an author name, enter **Ker**.

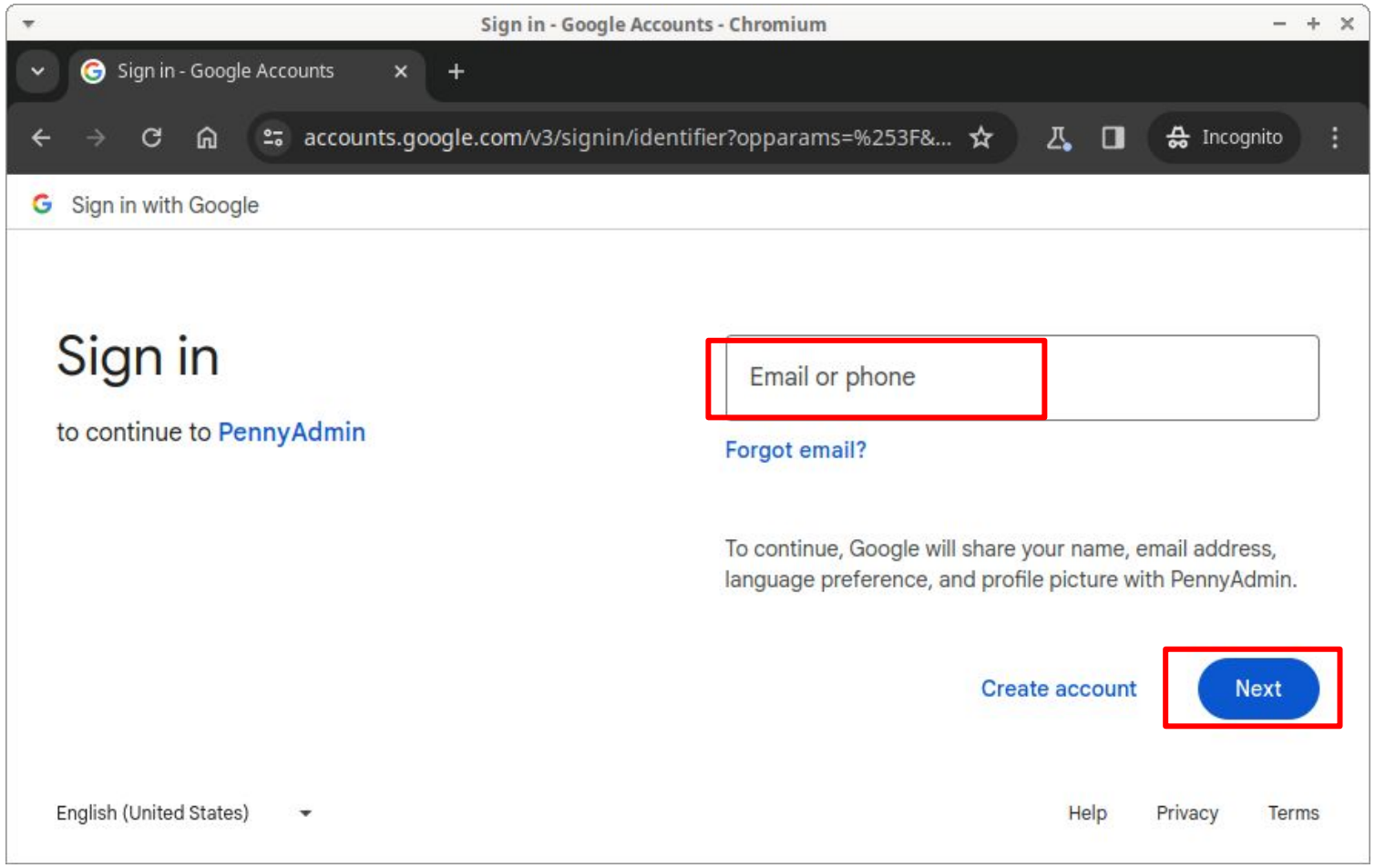
Running Locally

- See **PennyAdminGoogle** app (cont.)



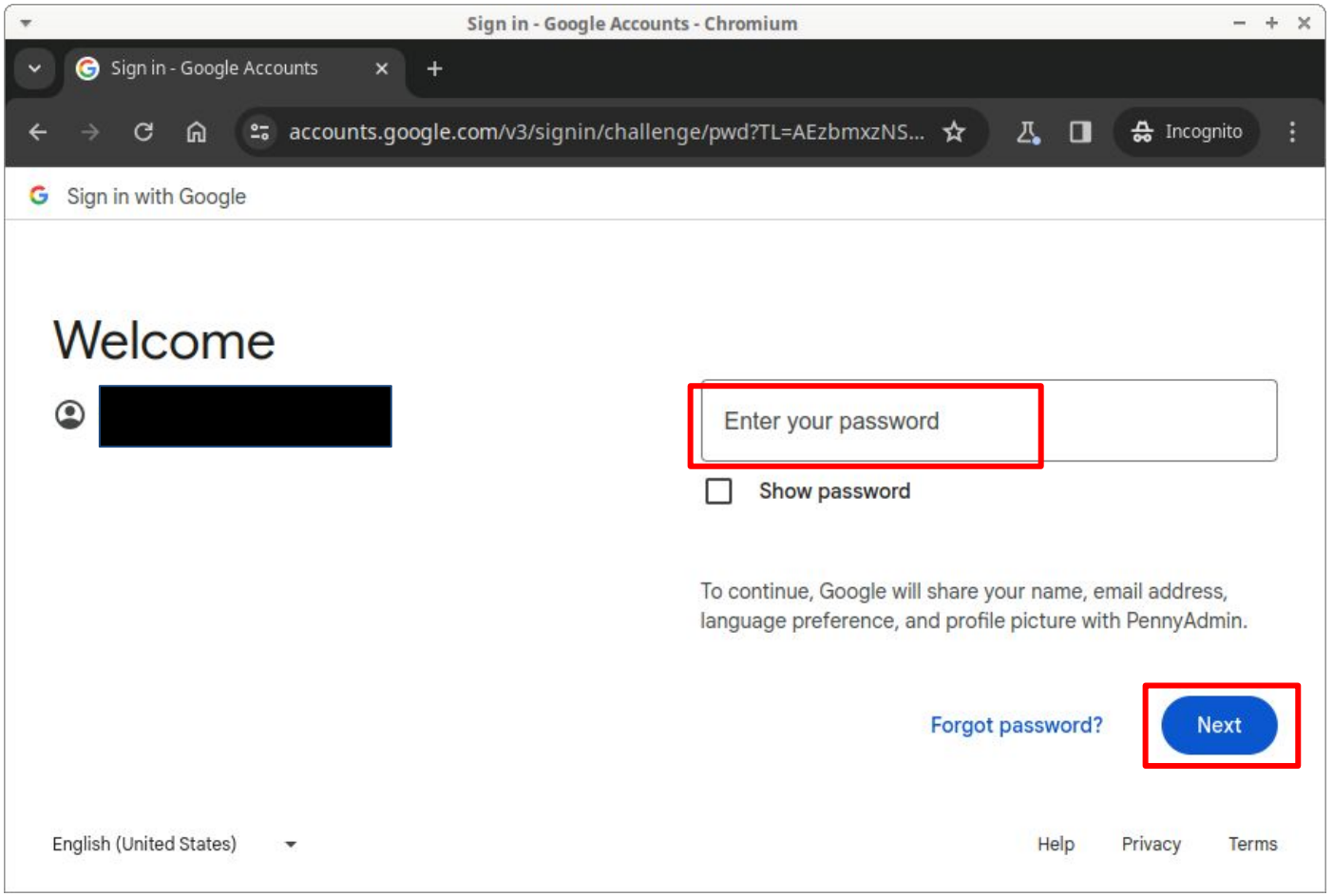
Running Locally

- See **PennyAdminGoogle** app (cont.)



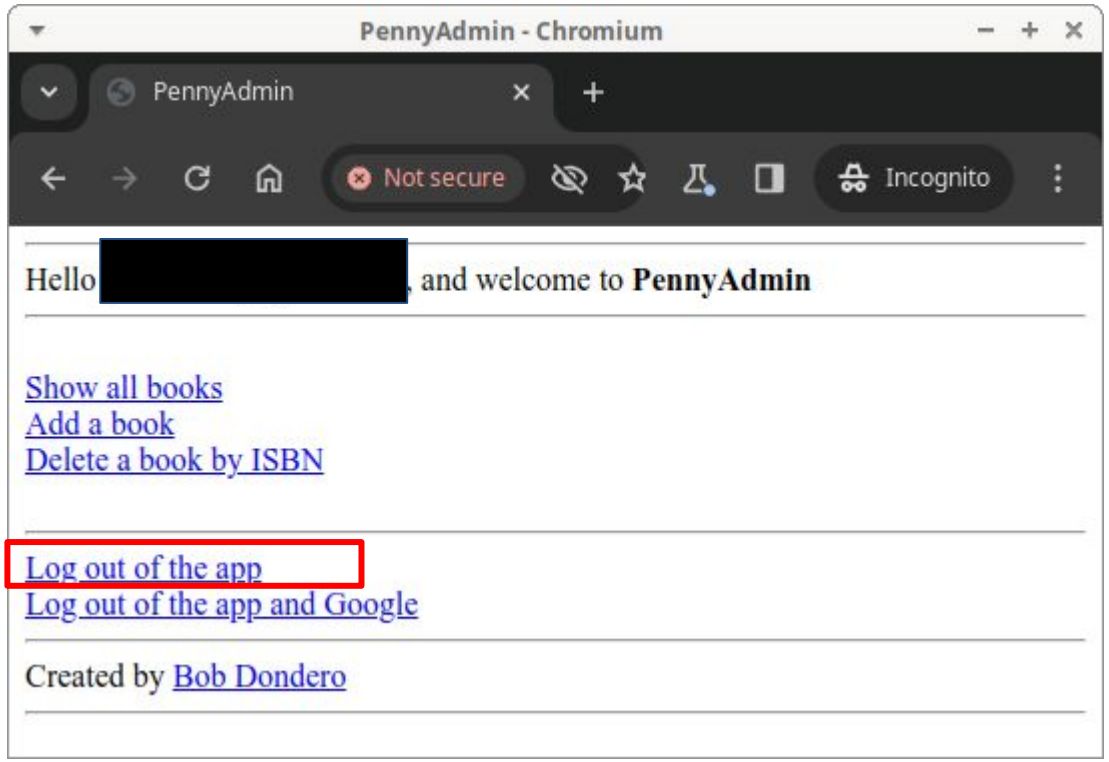
Running Locally

- See **PennyAdminGoogle** app (cont.)



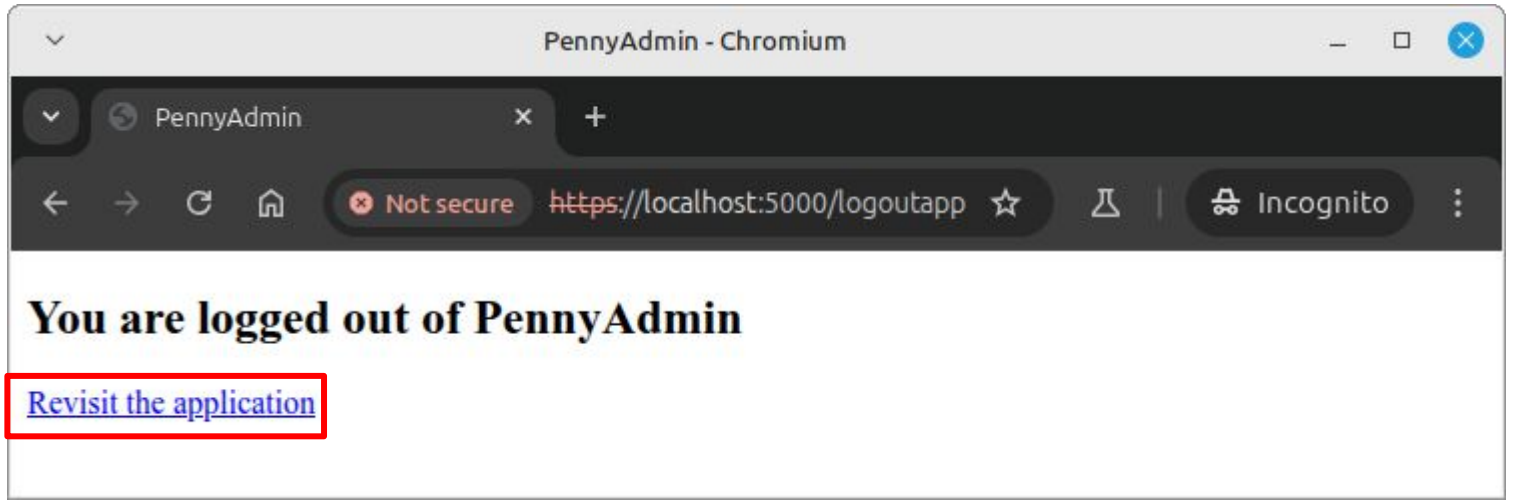
Running Locally

- See **PennyAdminGoogle** app (cont.)



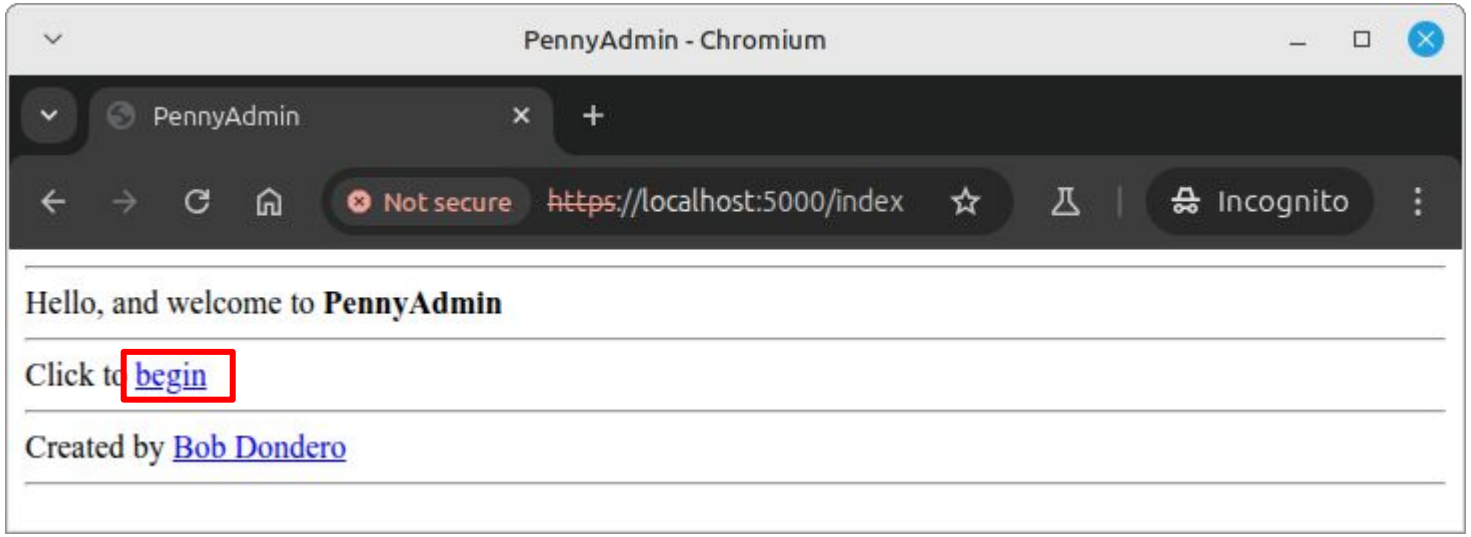
Running Locally

- See **PennyAdminGoogle** app (cont.)



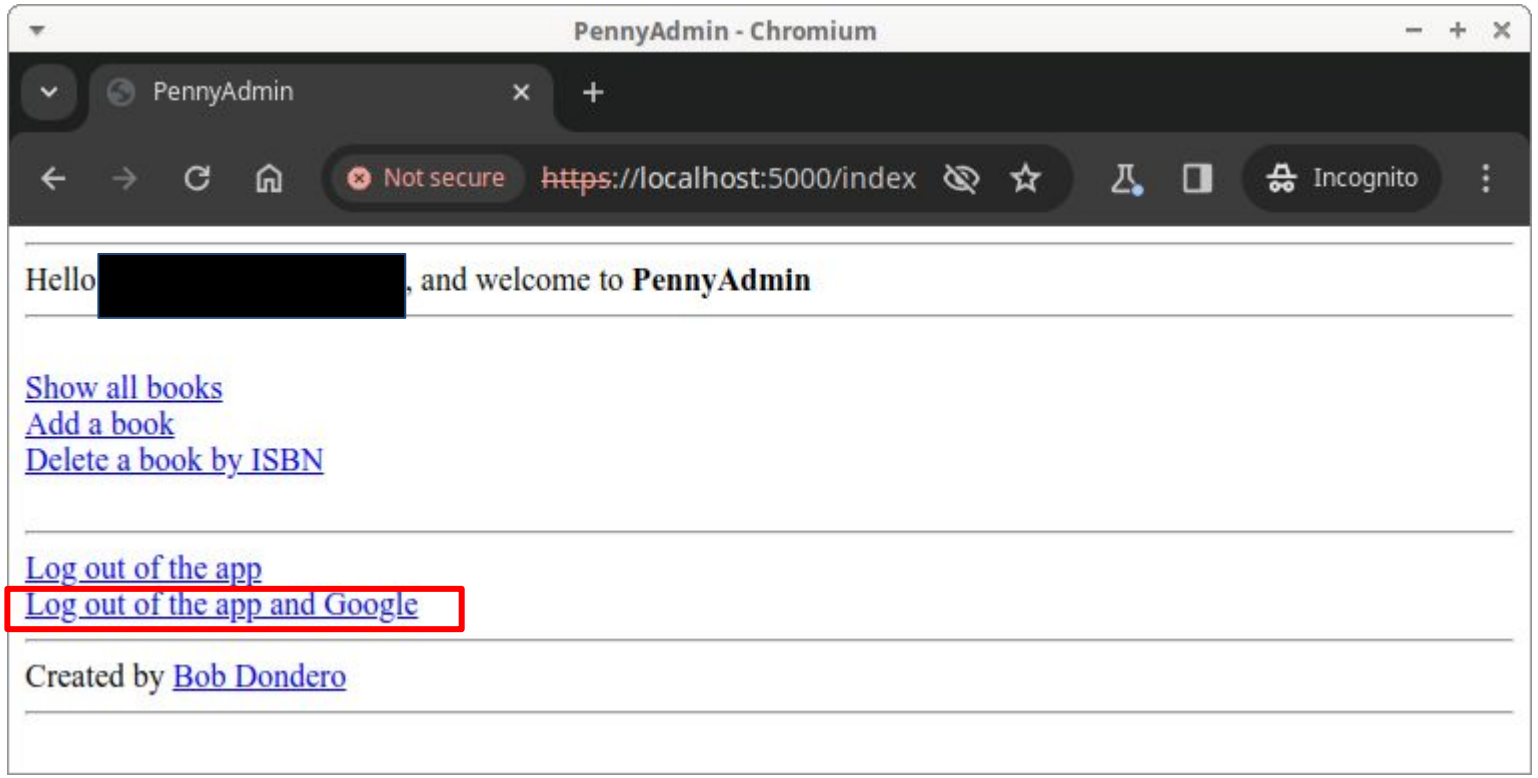
Running Locally

- See **PennyAdminGoogle** app (cont.)



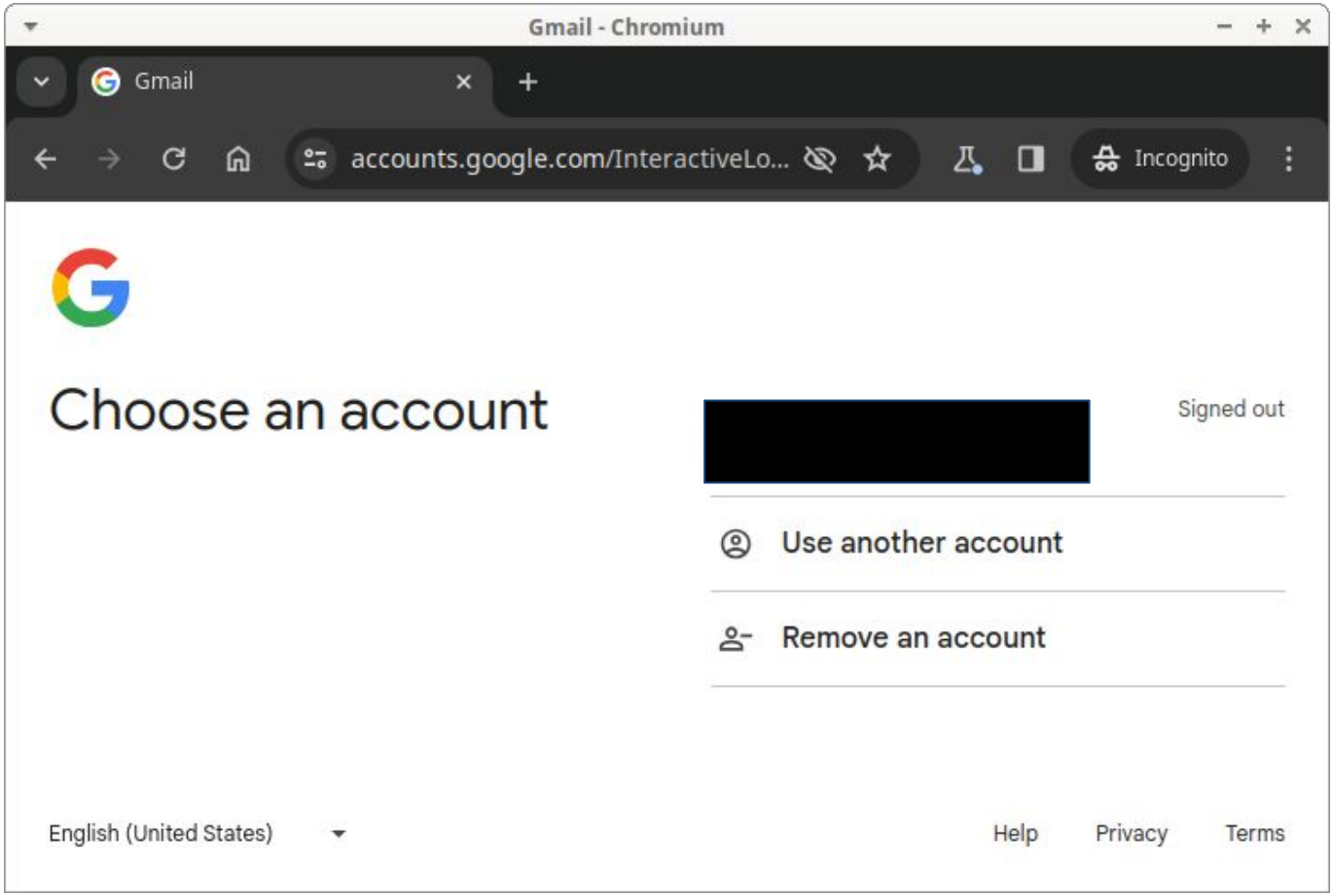
Running Locally

- See **PennyAdminGoogle** app (cont.)



Running Locally

- See **PennyAdminGoogle** app (cont.)



How to show loggedout page?

Agenda

- Google authentication
  - The code
  - Running locally
  - **Running on Render**
  - How it works
  - Assessment

Running on Render

- See **PennyAdminGoogle** app (cont.)
  - How to deploy it to Render and run it there…

Running on Render

1. Deploy the app to Render as a ***Web Service***.

    1.1. Provide this as the build command:

        pip install -r requirements.txt

    1.2. Provide this as the start command:

        gunicorn penny:app

    1.3. Define these environment variables:

        GOOGLE_CLIENT_ID=*yourGoogleClientId*
        GOOGLE_CLIENT_SECRET=*yourGoogleClientSecret*

Running on Render

2. Register your Render app with Google.

    2.2. Again browse to
        https://console.developers.google.com/apis/credentials

    2.3. Follow the instructions given above, except...

        1.3.1. For "Authorized JavaScript origins" enter
            https://yourappname.onrender.com.

        2.3.2. For "Authorized redirect URIs" enter
            https://yourappname.onrender.com/login/callback.

3. Browse to `https://yourappname.onrender.com

4. Authenticate using Google.

5. When prompted for an author name, enter **Ker.**

Agenda

- Google authentication
  - The code
  - Running locally
  - Running on Render
  - **How it works**
  - Assessment

How It Works

- Procedure
  - **Part 1**: User logs into Google
    - User must provide credentials
  - **Part 2**: User logs into PennyAdmin
    - User need not provide credentials

How It Works

- ***OAuth2***

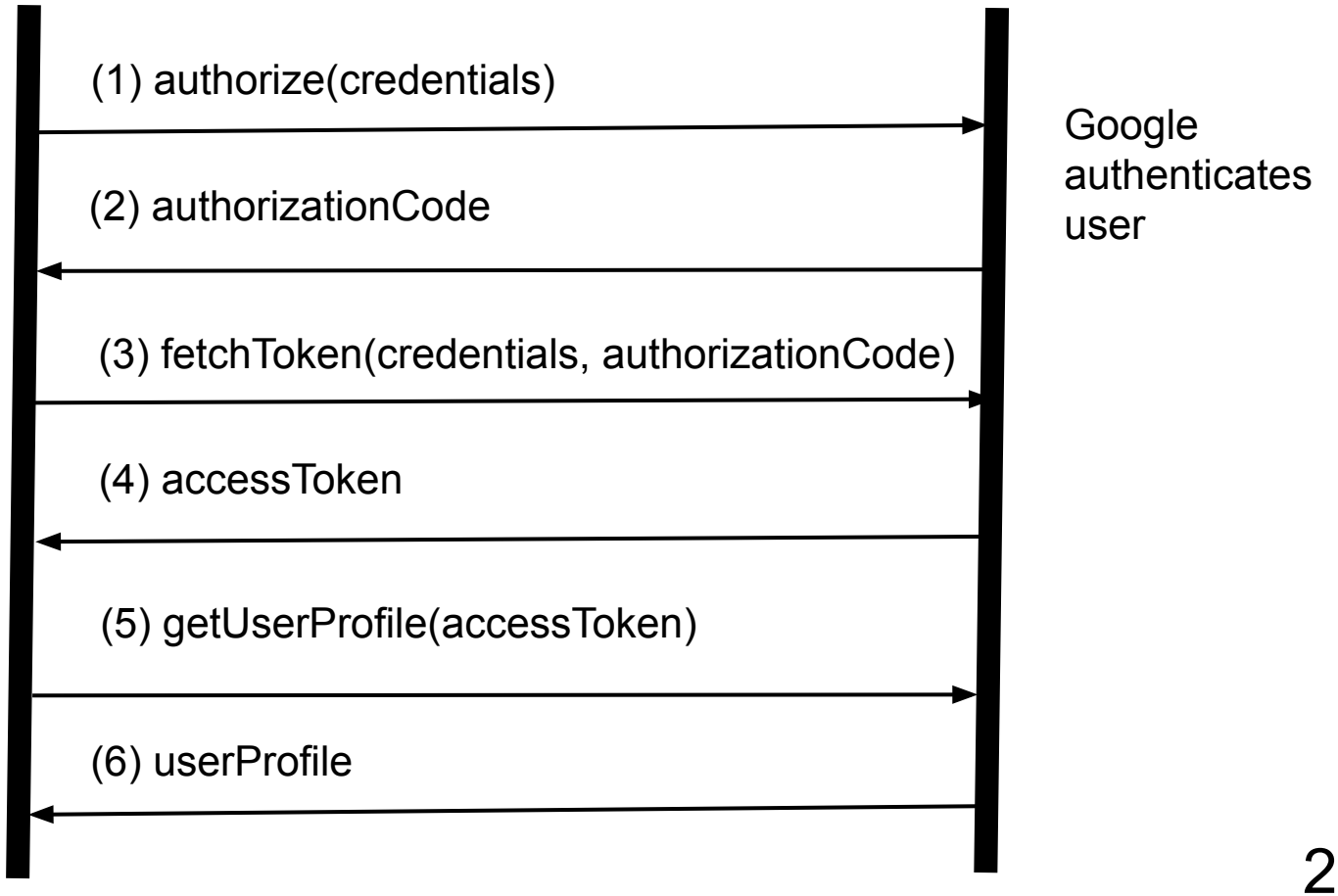
OAuth ("Open Authorization") is an open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information on other websites but without giving them the passwords. This mechanism is used by companies such as Amazon, **Google**, Facebook, Microsoft, and Twitter to permit the users to share information about their accounts with third-party applications or websites.

– https://en.wikipedia.org/wiki/OAuth

How It Works

OAuth2 Flow Overview:

Ahead of time: register PennyAdmin with Google; get credentials



How It Works

- See **PennyAdminGoogle** app (cont.)
  - The flow:

How It Works

First use of PennyAdmin in browser session, browser session not Google authenticated…

How It Works

(1) User
Type: `https://localhost:5000/XXX`

(2) Browser
Send GET request: `https://localhost:5000/XXX`

(3) PennyAdmin (in /index endpoint)
Email in session? **No**
Return redirect: `https://localhost:5000/login`

(4) Browser
Send GET request: `https://localhost:5000/login`

(5) PennyAdmin (in /login endpoint)
Return redirect to the Google authorization endpoint, passing `GOOGLE_CLIENT_ID` and `https://localhost:5000/login/callback` as parameters

(6) Browser
Send request to the Google authorization endpoint, passing `GOOGLE_CLIENT_ID` and `https://localhost:5000/login/callback` as parameters

How It Works

(7) Google

Are the application (identified by `GOOGLE_CLIENT_ID`) and the given callback (`https://localhost:5000/login/callback`) registered? **Yes**.
Do cookies indicate that the browser session is already Google authenticated? **No**.
Return Google login page to browser

(8) Browser

Render Google login page

(9) User

Enter Google email and password and submit form

(10) Browser

Send POST request to Google, with email and password in body

(11) Google

Does the user authenticate? **Yes**.
Return redirect:

`https://localhost:5000/login/callback?code=`*`authorizationcode`*

END OF PART 1; BEGINNING OF PART 2

How It Works

(12) Browser

Send GET request:

`https://localhost:5000/login/callback?code=`*`authorizationcode`*

(13) PennyAdmin (in login/callback endpoint)

Send POST request to Google with the *authorizationcode*, `GOOGLE_CLIENT_ID`, and `GOOGLE_CLIENT_SECRET` in the body

(14) Google

Return access token

(15) PennyAdmin (in login/callback endpoint)

Send GET request to Google with the access token as a header

(16) Google

Return user's profile data

(17) PennyAdmin (in login/callback endpoint)

Add user's profile data (notably email) to the session

Return redirect: `https://localhost:5000/XXX`

How It Works

(18) Browser
Send GET request: `https://localhost:5000/XXX`

(19) PennyAdmin
Email in session? **Yes**
Return XXX page

(20) Browser
Render XXX page

How It Works

Second use of PennyAdmin in browser session…

How It Works

(21) User
In index page, click on `https://localhost:5000/YYY` link

(22) Browser
Send GET request: `https://localhost:5000/YYY`

(23) PennyAdmin
Email in session? **Yes**
Return show page

(24) Browser
Render YYY page

How It Works

First use of PennyAdmin in browser session, browser session already Google authenticated…

How It Works

(25) User

Type: `https://localhost:5000/XXX`

(26) Browser

Send GET request: `https://localhost:5000/XXX`

(27) PennyAdmin (in /index endpoint)

Email in session? **No**

Return redirect: `https://localhost:5000/login`

(28) Browser

Send GET request: `https://localhost:5000/login`

(29) PennyAdmin (in /login endpoint)

Return redirect to the Google authorization endpoint, passing `GOOGLE_CLIENT_ID` and `https://localhost:5000/login/callback` as parameters

How It Works

(30) Browser

Send request to the Google authorization endpoint, passing `GOOGLE_CLIENT_ID` and `https://localhost:5000/login/callback` as parameters

(32) Google

Are the application (identified by `GOOGLE_CLIENT_ID`) and the given callback (`https://localhost:5000/login/callback`) registered? **Yes**

Do cookies indicate that the browser session is already Google authenticated? **Yes**

Return redirect:

`https://localhost:5000/login/callback?code=`*`authorizationcode`*

CONTINUE AT STEP 12

Agenda

- Google authentication
  - The code
  - Running locally
  - Running on Render
  - How it works
  - **Assessment**

Assessment

- **Pros**
  - Users need not remember (yet another) password
  - Application need not manage usernames or passwords
  - Application ***cannot*** access passwords
  - Application can access profile info that user provided to Google
    - Given name, family name, picture, …

Assessment

- **Cons**
  - Complex
  - Adds overhead, but mostly only during first user visit per browser session
  - Application is constrained to users who have Google accounts
  - Must use HTTPS with local server
  - If attacker learns user's password for **Google**, then attacker learns user's password for **your app**

Assessment

- For more information...
- https://realpython.com/flask-google-login/