Security Issues in Web Programming (Part 4)

Robert M. Dondero, Ph.D.

Princeton University

Objectives

- We will cover:
  - Data comm attacks
  - Third-party authentication (briefly):
    - CAS
    - Google authentication
    - Auth0 authentication

Agenda

- **Data comm attacks**
- Third-party authentication (briefly)
  - CAS
  - Google authentication
  - Auth0 authentication

Data Comm Attacks

- **Problem:**
  - Attacker may access data during comm between PennyAdmin app and browser
- **Solution:**
  - ***Hypertext Transfer Protocol Secure (HTTPS)***

Data Comm Attacks

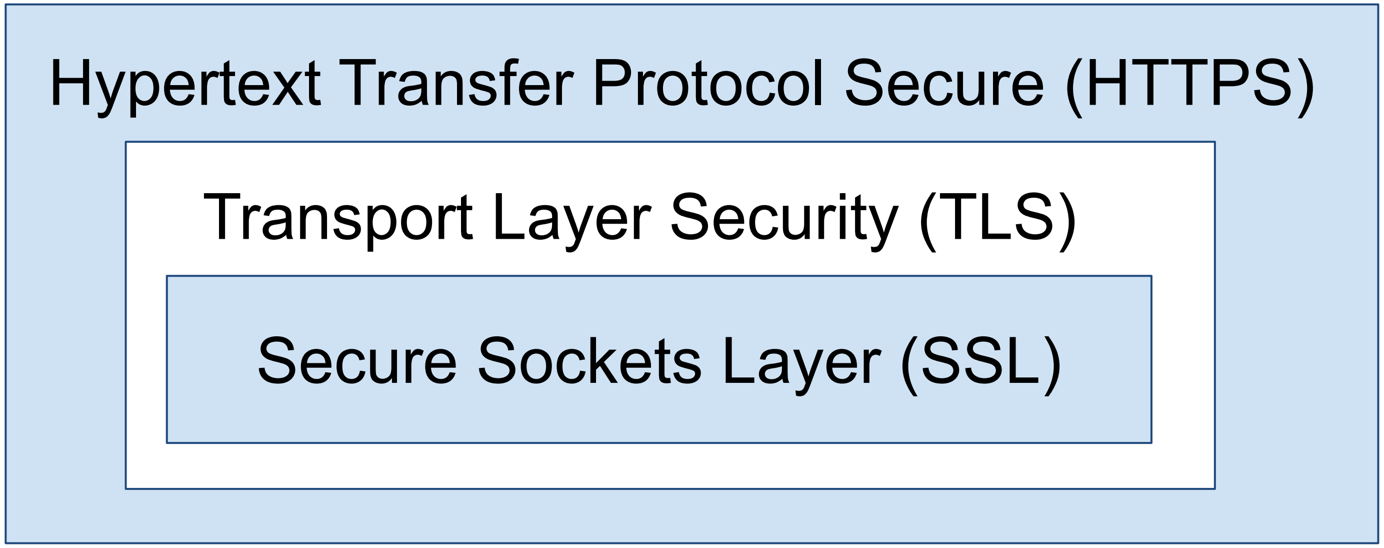
- **Technical** advantages of using HTTPS
  - Confidentiality
    - Prohibits ***message eavesdropping attacks***
  - Integrity
    - Prohibits ***message tampering attacks***
  - Authentication
    - Prohibits ***message forgery attacks***

Data Comm Attacks

- **Business** advantages of using HTTPS
  - Increases user confidence/trust in website
  - Increases Google search rank of website

Data Comm Attacks

- How HTTPS works:


Hypertext Transfer Protocol Secure (HTTPS)
Transport Layer Security (TLS)
Secure Sockets Layer (SSL)

Data Comm Attacks

- How to use HTTPS:
  - Configure server & app to use (& require use of) HTTPS
  - Command browser to send request specifying HTTPS as protocol
    - `https://…`

Data Comm Attacks

- How to configure server & app to use (& require use of) HTTPS:
  - Depends upon server…

Data Comm Attacks

- **Render server**
  - Already configured to use (& require use of) HTTPS
    - When server receives **`http:`**`//`*`something`* request, it sends redirect for **`https:`**`//`*`something`* request
  - So:
    - **Server**: Do nothing!
    - **App**: Do nothing!

Data Comm Attacks

- **Heroku server**
  - Already configured to use (but not require use of) HTTPS
    - When server receives **`https:`**`//`*`something`* request, it uses HTTPS
    - When server receives **`http:`**`//`*`something`* request, it uses HTTP
  - So
    - **Server**: (Regrettably) Do nothing!
    - **App**: Small change…

Data Comm Attacks

- **Solution 1:**
  - App explicitly performs redirects

Data Comm Attacks

- See **<u>PennyAdmin13Https</u>** app
  - runserver.py
  - penny.sql, penny.sqlite
  - database.py
  - header.html, footer.html
  - index.html, show.html,
  - add.html, delete.html, reportresults.html
  - login.html, signup.html, loggedout.html
  - top.py, **penny.py**, auth.py

PennyAdmin13Https/penny.py (Page 1 of 3)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# penny.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import flask
import database
import auth

from top import app

#-----------------------------------------------------------------------

@app.before_request
def before_request():
    if (not app.debug) and (not flask.request.is_secure):
        url = flask.request.url.replace('http://', 'https://', 1)
        return flask.redirect(url, code=301)
    return None

#-----------------------------------------------------------------------

@app.route('/', methods=['GET'])
@app.route('/index', methods=['GET'])
def index():

    username = auth.authenticate()
    is_authorized = database.is_authorized(username)

    html_code = flask.render_template('index.html', username=username,
        is_authorized=is_authorized)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/show', methods=['GET'])
def show():

    username = auth.authenticate()

    books = database.get_books()
    html_code = flask.render_template('show.html',
        username=username, books=books)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

def report_results(username, message1, message2):

    html_code = flask.render_template('reportresults.html',
        username=username, message1=message1, message2=message2)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/add', methods=['GET'])
def add():

    username = auth.authenticate()
    if not database.is_authorized(username):
```

PennyAdmin13Https/penny.py (Page 2 of 3)

```python
        html_code = 'You are not authorized to add books.'
        response = flask.make_response(html_code)
        return response

    html_code = flask.render_template('add.html', username=username)

    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/handleadd', methods=['POST'])
def handle_add():

    username = auth.authenticate()
    if not database.is_authorized(username):
        html_code = 'You are not authorized to add books.'
        response = flask.make_response(html_code)
        return response

    isbn = flask.request.form.get('isbn')
    if (isbn is None) or (isbn.strip() == ''):
        return report_results(username, 'Missing ISBN', '')

    author = flask.request.form.get('author')
    if (author is None) or (author.strip() == ''):
        return report_results(username, 'Missing author', '')

    title = flask.request.form.get('title')
    if (title is None) or (title.strip() == ''):
        return report_results(username, 'Missing title', '')

    isbn = isbn.strip()
    author = author.strip()
    title = title.strip()

    successful = database.add_book(isbn, author, title)
    if successful:
        message1 = 'The addition was successful'
        message2 = 'The database now contains a book with isbn ' + isbn
        message2 += ' author ' + author + ' and title ' + title
    else:
        message1 = 'The addition was unsuccessful'
        message2 = 'A book with ISBN ' + isbn + ' already exists'

    return report_results(username, message1, message2)

#-----------------------------------------------------------------------

@app.route('/delete', methods=['GET'])
def delete():

    username = auth.authenticate()
    if not database.is_authorized(username):
        html_code = 'You are not authorized to delete books.'
        response = flask.make_response(html_code)
        return response

    html_code = flask.render_template('delete.html', username=username)

    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

```

PennyAdmin13Https/penny.py (Page 3 of 3)

```python
@app.route('/handledelete', methods=['POST'])
def handle_delete():

    username = auth.authenticate()
    if not database.is_authorized(username):
        html_code = 'You are not authorized to delete books.'
        response = flask.make_response(html_code)
        return response

    isbn = flask.request.form.get('isbn')
    if (isbn is None) or (isbn.strip() == ''):
        return report_results(username, 'Missing ISBN', '')

    isbn = isbn.strip()

    database.delete_book(isbn)

    message1 = 'The deletion was successful'
    message2 = 'The database now does not contain a book with ISBN '
    message2 += isbn

    return report_results(username, message1, message2)
```

blank (Page 1 of 1)

```
This page is intentionally blank.
```

Data Comm Attacks

- **Solution 2:**
  - ***`flask_talisman`*** module

Data Comm Attacks

- See **PennyAdmin14Https** app
  - runserver.py
  - penny.sql, penny.sqlite
  - database.py
  - header.html, footer.html
  - index.html, show.html,
  - add.html, delete.html, reportresults.html
  - login.html, signup.html, loggedout.html
  - **top.py**, **penny.py**, auth.py

PennyAdmin14Https/top.py (Page 1 of 1)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# top.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import flask
import flask_wtf.csrf
import flask_talisman
import dotenv

app = flask.Flask('penny', template_folder='.')

dotenv.load_dotenv()
app.secret_key = os.environ['APP_SECRET_KEY']

flask_wtf.csrf.CSRFProtect(app)

flask_talisman.Talisman(app)
```

PennyAdmin14Https/penny.py (Page 1 of 3)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# penny.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import flask
import database
import auth

from top import app

#-----------------------------------------------------------------------

@app.route('/', methods=['GET'])
@app.route('/index', methods=['GET'])
def index():

    username = auth.authenticate()
    is_authorized = database.is_authorized(username)

    html_code = flask.render_template('index.html', username=username,
        is_authorized=is_authorized)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/show', methods=['GET'])
def show():

    username = auth.authenticate()

    books = database.get_books()
    html_code = flask.render_template('show.html',
        username=username, books=books)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

def report_results(username, message1, message2):

    html_code = flask.render_template('reportresults.html',
        username=username, message1=message1, message2=message2)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/add', methods=['GET'])
def add():

    username = auth.authenticate()
    if not database.is_authorized(username):
        html_code = 'You are not authorized to add books.'
        response = flask.make_response(html_code)
        return response

    html_code = flask.render_template('add.html', username=username)

    response = flask.make_response(html_code)
    return response

```

PennyAdmin14Https/penny.py (Page 2 of 3)

```python
#-----------------------------------------------------------------------

@app.route('/handleadd', methods=['POST'])
def handle_add():

    username = auth.authenticate()
    if not database.is_authorized(username):
        html_code = 'You are not authorized to add books.'
        response = flask.make_response(html_code)
        return response

    isbn = flask.request.form.get('isbn')
    if (isbn is None) or (isbn.strip() == ''):
        return report_results(username, 'Missing ISBN', '')

    author = flask.request.form.get('author')
    if (author is None) or (author.strip() == ''):
        return report_results(username, 'Missing author', '')

    title = flask.request.form.get('title')
    if (title is None) or (title.strip() == ''):
        return report_results(username, 'Missing title', '')

    isbn = isbn.strip()
    author = author.strip()
    title = title.strip()

    successful = database.add_book(isbn, author, title)
    if successful:
        message1 = 'The addition was successful'
        message2 = 'The database now contains a book with isbn ' + isbn
        message2 += ' author ' + author + ' and title ' + title
    else:
        message1 = 'The addition was unsuccessful'
        message2 = 'A book with ISBN ' + isbn + ' already exists'

    return report_results(username, message1, message2)

#-----------------------------------------------------------------------

@app.route('/delete', methods=['GET'])
def delete():

    username = auth.authenticate()
    if not database.is_authorized(username):
        html_code = 'You are not authorized to delete books.'
        response = flask.make_response(html_code)
        return response

    html_code = flask.render_template('delete.html', username=username)

    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/handledelete', methods=['POST'])
def handle_delete():

    username = auth.authenticate()
    if not database.is_authorized(username):
        html_code = 'You are not authorized to delete books.'
        response = flask.make_response(html_code)
        return response

```

PennyAdmin14Https/penny.py (Page 3 of 3)

```python
    isbn = flask.request.form.get('isbn')
    if (isbn is None) or (isbn.strip() == ''):
        return report_results(username, 'Missing ISBN', '')

    isbn = isbn.strip()

    database.delete_book(isbn)

    message1 = 'The deletion was successful'
    message2 = 'The database now does not contain a book with ISBN '
    message2 += isbn

    return report_results(username, message1, message2)
```

Data Comm Attacks

- Notes:
  - Good to design your app to require use of HTTPS even when the app server already forces use of HTTPS
  - flask_talisman implements some additional security measures
  - Need not configure Flask test server to use (or require use of) HTTPS
    - But if you want to…
    - Or if you're using Google authentication…

Data Comm Attacks

- How to configure Flask test server & app to use (& require use of) HTTPS:

Data Comm Attacks

- **Preliminary step**: Get a ***certificate*** for your app
- **Option 1**: Get a certificate that is signed by a ***certificate authority***

Data Comm Attacks

Certificate authorities:

Rank	Authority	Market Share
1	IdenTrust	49%
2	DigiCert	19%
3	Sectigo	16%
4	Let’s Encrypt	8%
5	GoDaddy	6%
6	GlobalSign	3%

https://en.wikipedia.org/wiki/Certificate_authority#Providers
(as of Aug 2022)

Data Comm Attacks

- **Preliminary step**: Get a certificate for your app

- **Option 1**: Buy a certificate that is signed by a certificate authority
- **Option 2**: Create a ***self-signed certificate***

Data Comm Attacks

```
$ openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem -days 365
Generating a RSA private key
..........................................................++++
.......++++
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
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 []:
$
```

Output: cert.pem, key.pem

Data Comm Attacks

- Self-signed certificate
  - Confidentiality: yes
  - Integrity: yes
  - Authentication: no

Data Comm Attacks

- See **<u>PennyAdmin15HttpsLocal</u>** app
  - **runserver.py**
  - penny.sql, penny.sqlite
  - database.py
  - header.html, footer.html
  - index.html, show.html,
  - add.html, delete.html, reportresults.html
  - login.html, signup.html, loggedout.html
  - top.py, penny.py, auth.py

PennyAdmin15HttpsLocal/runserver.py (Page 1 of 1)

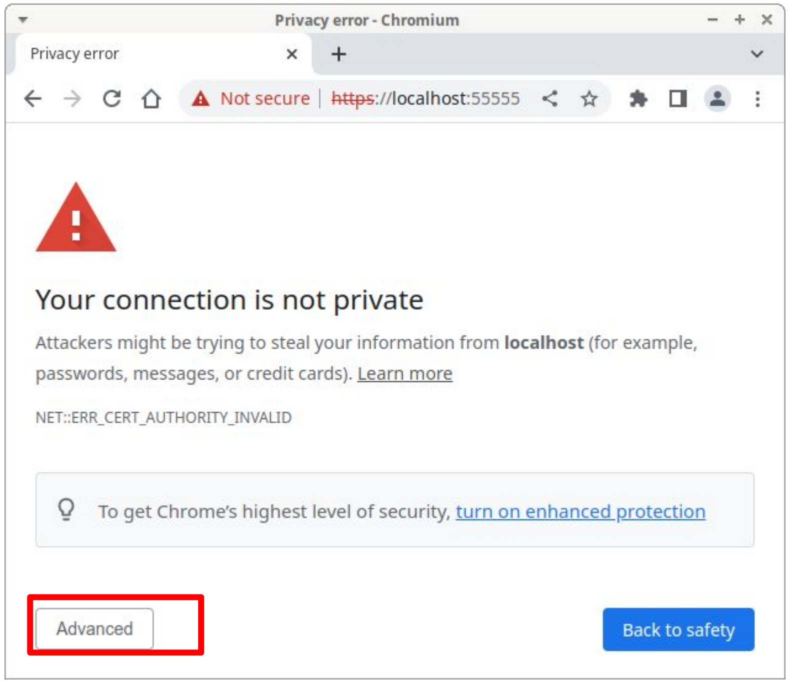
```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# runserver.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import sys
import penny

def main():

    if len(sys.argv) != 2:
        print('Usage: ' + sys.argv[0] + ' port', file=sys.stderr)
        sys.exit(1)

    try:
        port = int(sys.argv[1])
    except Exception:
        print('Port must be an integer.', file=sys.stderr)
        sys.exit(1)

    try:
        penny.app.run(host='0.0.0.0', port=port,
            ssl_context=('cert.pem', 'key.pem'))
    except Exception as ex:
        print(ex, file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

blank (Page 1 of 1)

```
This page is intentionally blank.
```

Data Comm Attacks

- See **PennyAdmin15HttpsLocal** app
  - On local computer with Flask test server (using self-signed certif)



Privacy error - Chromium

Privacy error

Not secure | https://localhost:55555

Your connection is not private

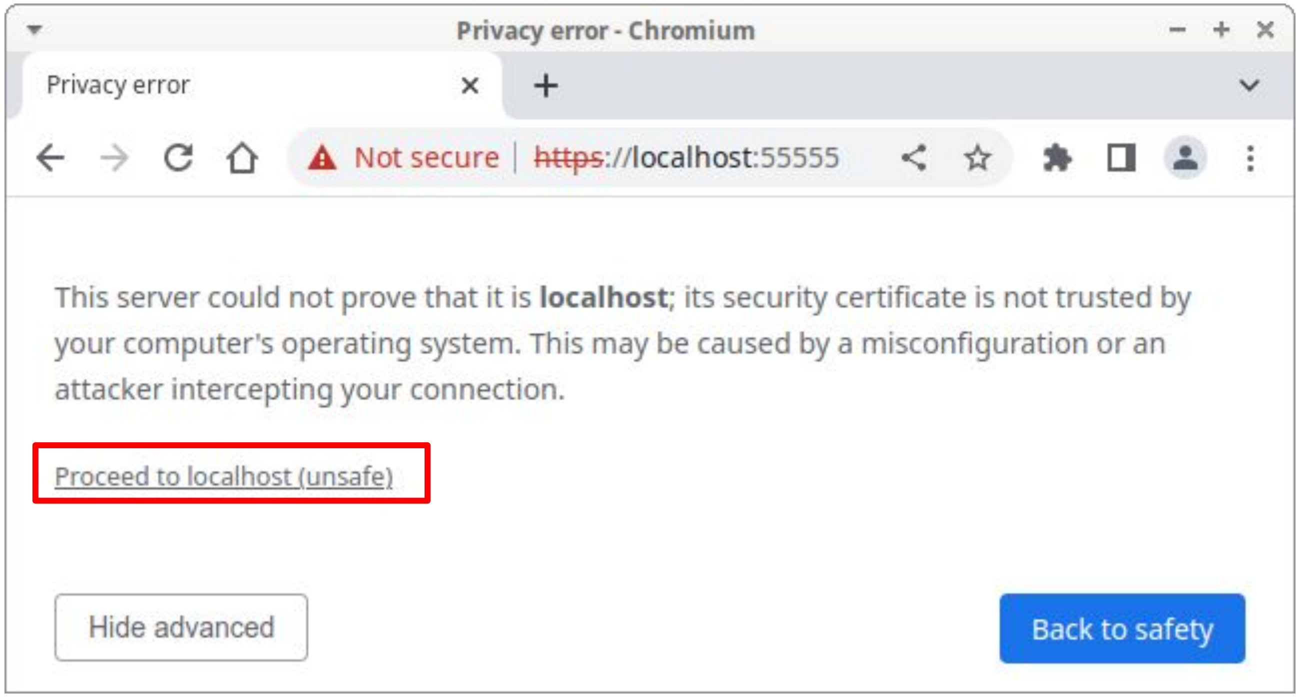
Attackers might be trying to steal your information from **localhost** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_AUTHORITY_INVALID

To get Chrome's highest level of security, turn on enhanced protection

Data Comm Attacks

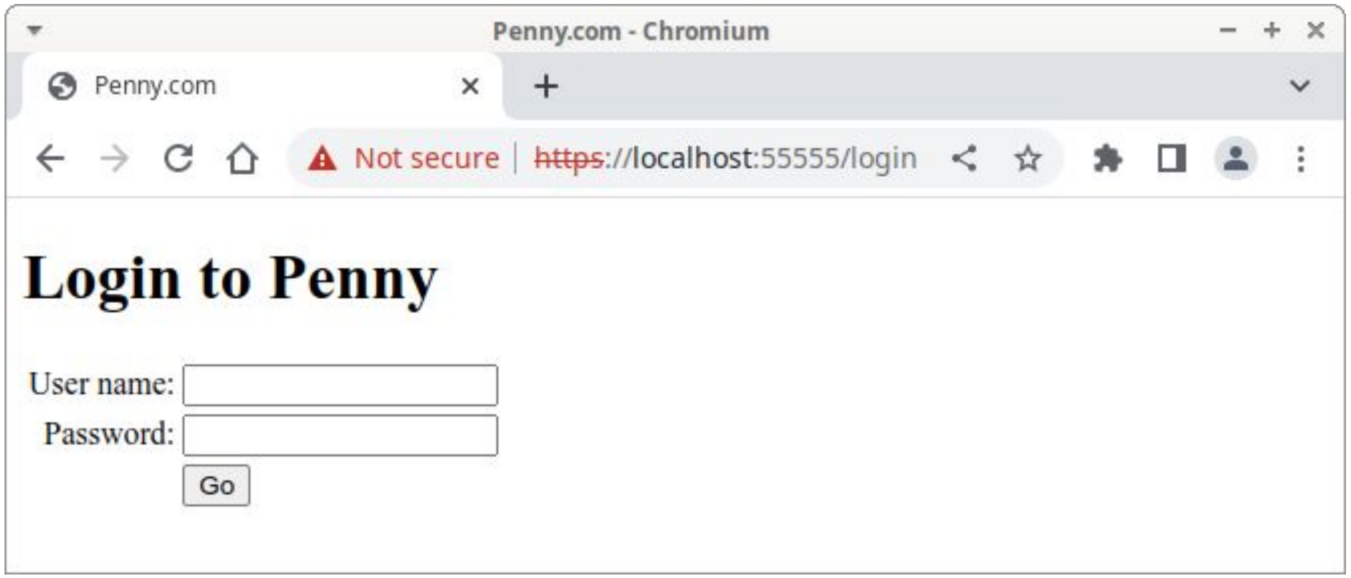
- See **PennyAdmin15HttpsLocal** app
  - On local computer with Flask test server (using self-signed certif)



This server could not prove that it is **localhost**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Data Comm Attacks

- See **PennyAdmin15HttpsLocal** app
  - On local computer with Flask test server (using self-signed certif)



Login to Penny

User name:

Password:

Go

Data Comm Attacks

- Q: Project concern?
- A: **Yes**

Agenda

- Data comm attacks
- **Third-party authentication (briefly)**
  - CAS
  - Google authentication
  - Auth0 authentication

Agenda

- Data comm attacks
- Third-party authentication (briefly)
  - **CAS**
  - Google authentication
  - Auth0 authentication

CAS

- ***Central Authentication Service (CAS)***

"The **Central Authentication Service (CAS)** is a single sign-on protocol for the web. **Its purpose is to permit a user to access multiple applications while providing their credentials (such as userid and password) only once. It also allows web applications to authenticate users without gaining access to a user's security credentials, such as a password.**"

– https://en.wikipedia.org/wiki/Central_Authentication_Service

CAS

- See **<u>PennyAdmin16Cas</u>** app (cont.)
  - **Part 1**: User logs into CAS server
    - Unnecessary if user is already logged into CAS server
    - User must provide credentials
  - **Part 2**: User logs into PennyAdmin
    - User need not provide credentials

CAS

- See **PennyAdmin16Cas** app (cont.)
  - How to run it on your local computer…

CAS

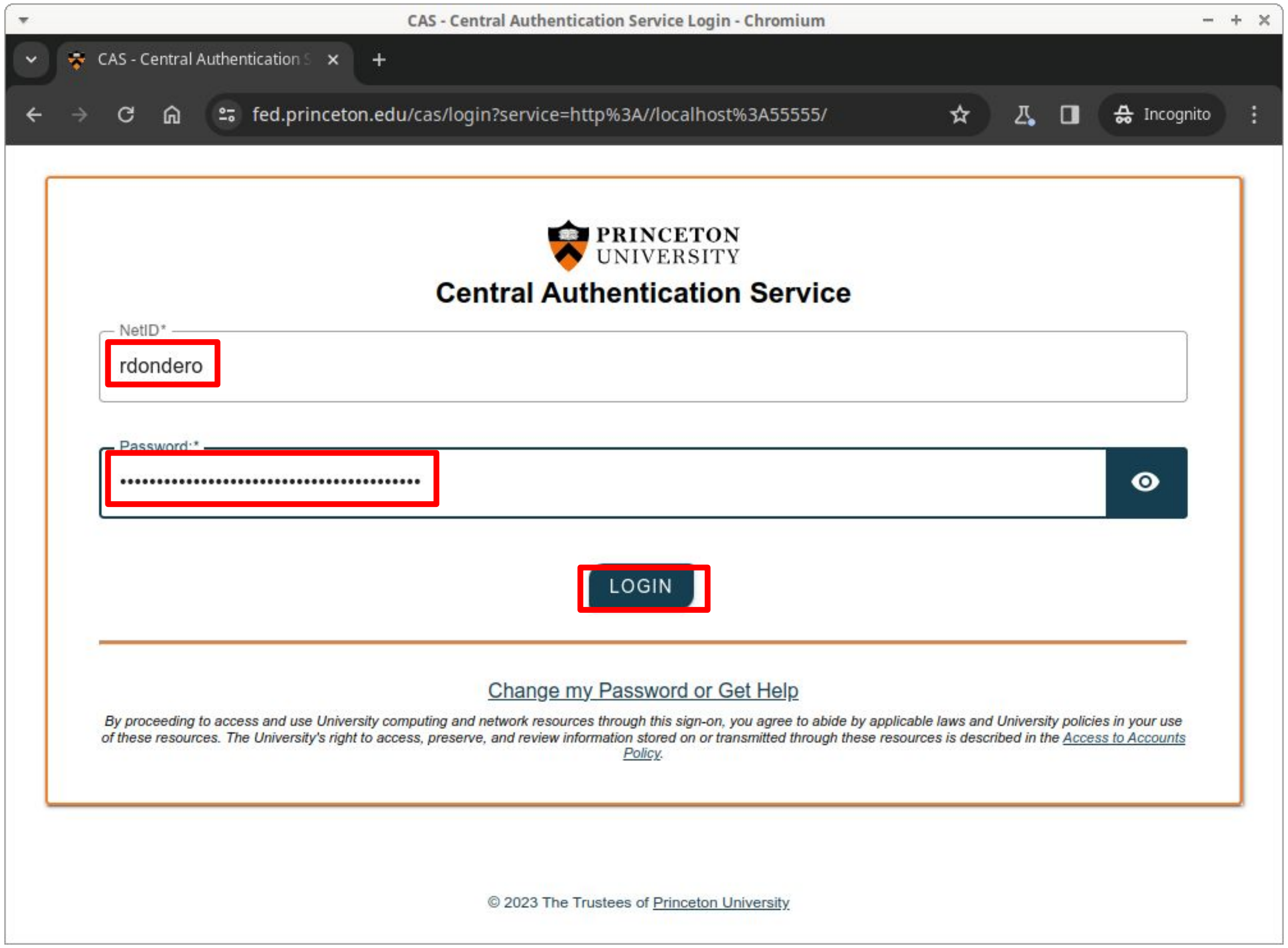
- See **PennyAdmin16Cas** app (cont.)
  - In terminal, enter this command:

```
$ python runserver.py 55555
```

  - In browser, enter URL:
    - http://localhost:55555
      - Must use `localhost` (and not `127.0.0.1`, and not the real IP address of your computer)

CAS

- See **PennyAdmin16Cas** app (cont.)



CAS - Central Authentication Service Login - Chromium

CAS - Central Authentication S

fed.princeton.edu/cas/login?service=http%3A//localhost%3A55555/

Incognito

PRINCETON UNIVERSITY

Central Authentication Service

NetID*

rdondero

Password:*

••

LOGIN

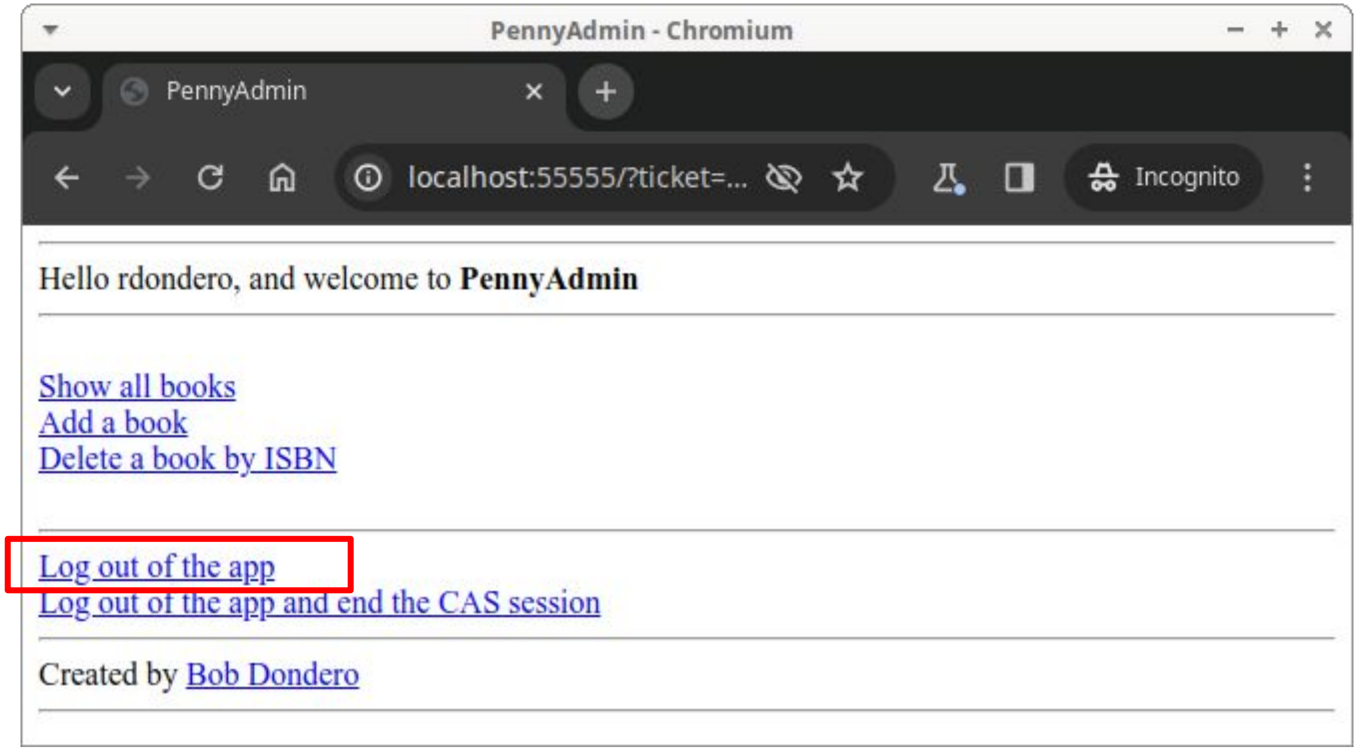
Change my Password or Get Help

By proceeding to access and use University computing and network resources through this sign-on, you agree to abide by applicable laws and University policies in your use of these resources. The University's right to access, preserve, and review information stored on or transmitted through these resources is described in the Access to Accounts Policy.

© 2023 The Trustees of Princeton University

CAS

- See **PennyAdmin16Cas** app (cont.)



Hello rdondero, and welcome to **PennyAdmin**

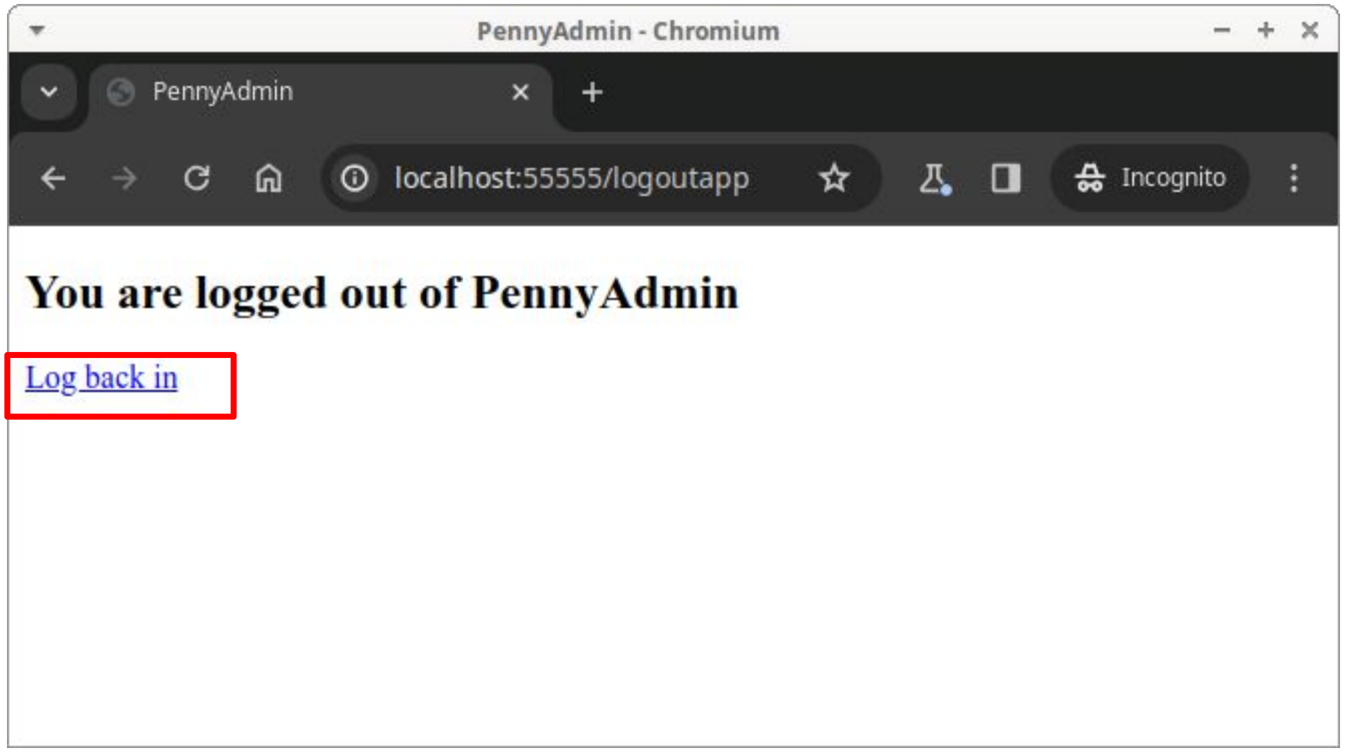
Show all books
Add a book
Delete a book by ISBN

Log out of the app
Log out of the app and end the CAS session

Created by Bob Dondero

CAS

- See **PennyAdmin16Cas** app (cont.)

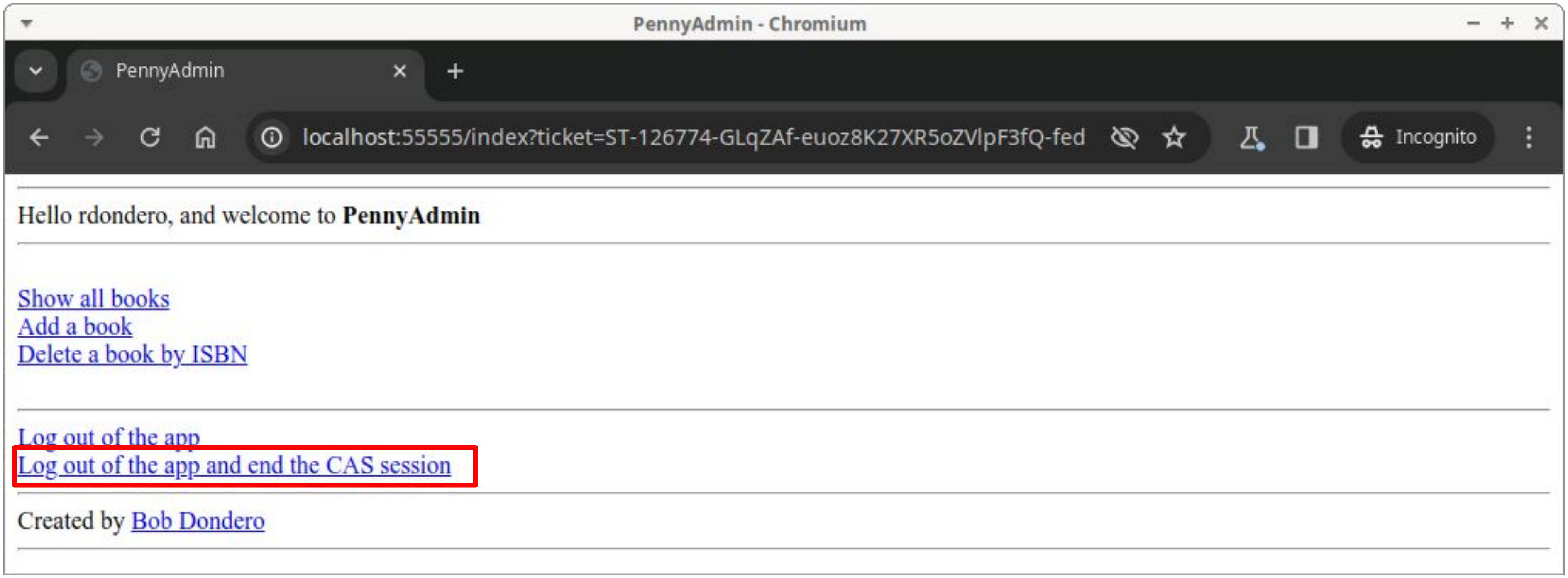


You are logged out of PennyAdmin

Log back in

CAS

- See **PennyAdmin16Cas** app (cont.)


PennyAdmin - Chromium
PennyAdmin
localhost:55555/index?ticket=ST-126774-GLqZAf-euoz8K27XR5oZVlpF3fQ-fed
Incognito

Hello rdondero, and welcome to **PennyAdmin**

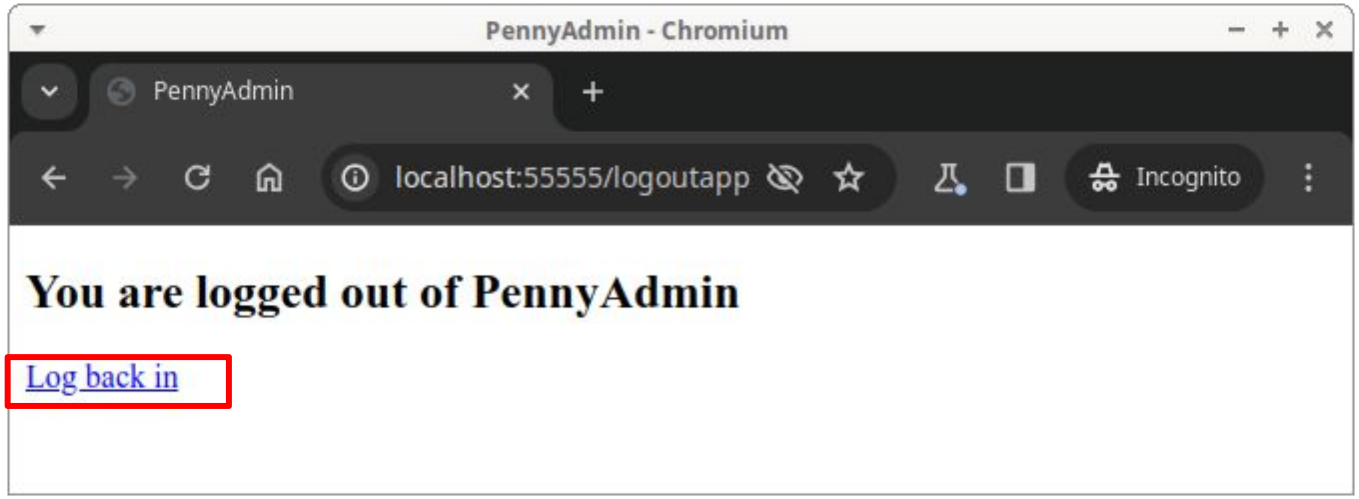
Show all books
Add a book
Delete a book by ISBN

Log out of the app
Log out of the app and end the CAS session

Created by Bob Dondero

CAS

- See **PennyAdmin16Cas** app (cont.)

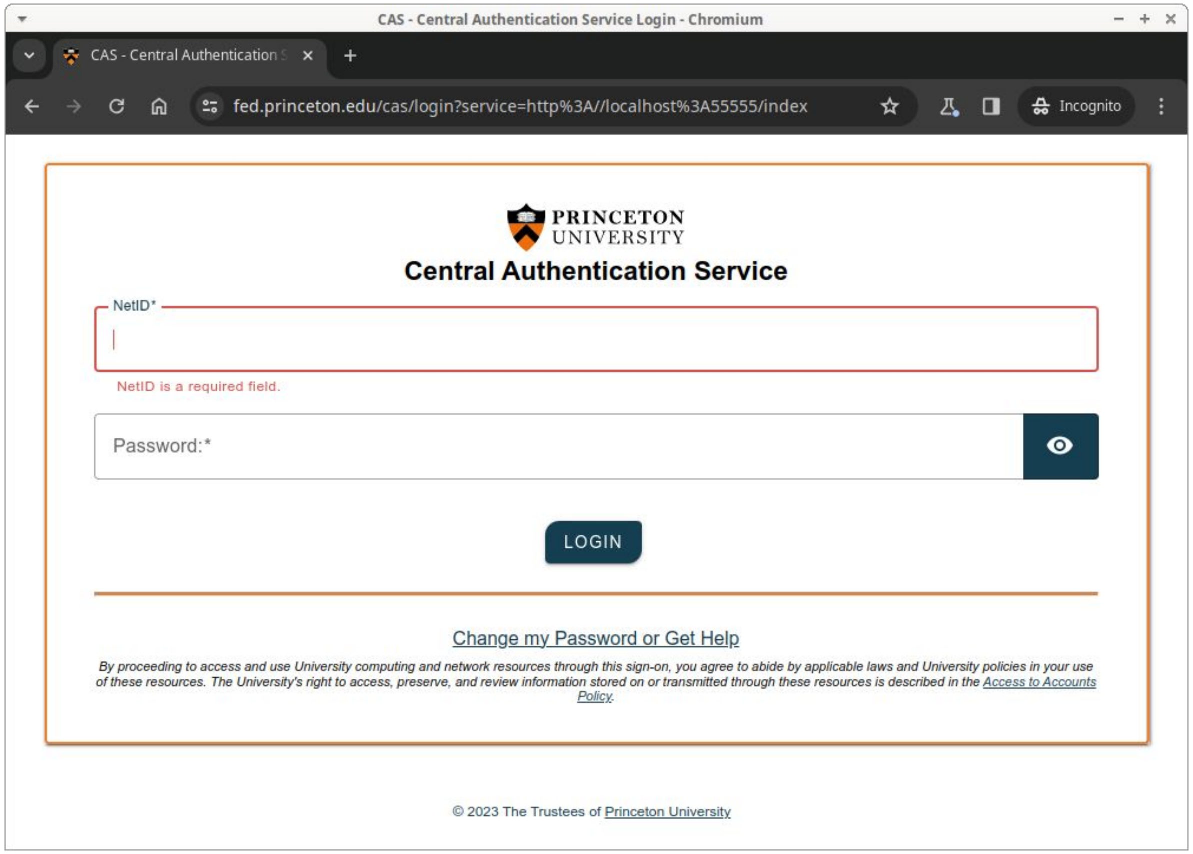


You are logged out of PennyAdmin

Log back in

CAS

- See **PennyAdmin16Cas** app (cont.)


CAS - Central Authentication Service Login - Chromium

fed.princeton.edu/cas/login?service=http%3A//localhost%3A55555/index

PRINCETON UNIVERSITY

Central Authentication Service

NetID*

NetID is a required field.

Password:*

LOGIN

Change my Password or Get Help

By proceeding to access and use University computing and network resources through this sign-on, you agree to abide by applicable laws and University policies in your use of these resources. The University's right to access, preserve, and review information stored on or transmitted through these resources is described in the Access to Accounts Policy.

© 2023 The Trustees of Princeton University

CAS

- See **<u>PennyAdmin16Cas</u>** app (cont.)
  - How to run it on Render (or Heroku, or any cloud service) …

CAS

- See **PennyAdmin16Cas** app (cont.)
  - Ask OIT to place the URL of the app on the ***Princeton CAS white list***
    - Instructions are provided in the COS 333 *Princeton Data Sources* web page
  - In browser, enter URL:
    - https://*ipaddress*

CAS

- See **PennyAdmin16Cas** app (cont.)
  - runserver.py
  - **penny.sql**, penny.sqlite
  - **database.py**
  - header.html, **footer.html**
  - index.html, show.html,
  - add.html, delete.html, reportresults.html
  - loggedout.html
  - top.py, penny.py, **auth.py**

PennyAdmin16Cas/penny.sql (Page 1 of 1)

```
DROP TABLE IF EXISTS books;
CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);
INSERT INTO books (isbn, author, title)
   VALUES ('123', 'Kernighan','The Practice of Programming');
INSERT INTO books (isbn, author, title)
   VALUES ('234', 'Kernighan','The C Programming Language');
INSERT INTO books (isbn, author, title)
   VALUES ('345', 'Sedgewick','Algorithms in C');

DROP TABLE IF EXISTS authorizedusers;
CREATE TABLE authorizedusers (username TEXT);
INSERT INTO authorizedusers (username) VALUES ('rdondero');
INSERT INTO authorizedusers (username) VALUES ('bwk');
```

PennyAdmin16Cas/footer.html (Page 1 of 1)

```
<hr>
<a href="logoutapp">Log out of the app</a></br>
<a href="logoutcas">Log out of the app and end the CAS session</a></br>
<hr>
Created by <a href="https://www.cs.princeton.edu/˜rdondero">
Bob Dondero</a>
<hr>
```

PennyAdmin16Cas/database.py (Page 1 of 3)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# database.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import sqlalchemy
import sqlalchemy.orm
import dotenv

#-----------------------------------------------------------------------

dotenv.load_dotenv()
_database_url = os.getenv('DATABASE_URL', 'sqlite:///penny.sqlite')
_database_url = _database_url.replace('postgres://', 'postgresql://')

#-----------------------------------------------------------------------

Base = sqlalchemy.orm.declarative_base()

class Book (Base):
    __tablename__ = 'books'
    isbn = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
    author = sqlalchemy.Column(sqlalchemy.String)
    title = sqlalchemy.Column(sqlalchemy.String)

class AuthorizedUser (Base):
    __tablename__ = 'authorizedusers'
    username = sqlalchemy.Column(sqlalchemy.String, primary_key=True)

_engine = sqlalchemy.create_engine(_database_url)

#-----------------------------------------------------------------------

def get_books():

    books = []

    with sqlalchemy.orm.Session(_engine) as session:
        query = session.query(Book)
        table = query.all()
        for row in table:
            book = {'isbn': row.isbn, 'author': row.author,
                'title': row.title}
            books.append(book)

    return books

#-----------------------------------------------------------------------

def add_book(isbn, author, title):

    with sqlalchemy.orm.Session(_engine) as session:
        row = Book(isbn=isbn, author=author, title=title)
        session.add(row)
        try:
            session.commit()
            return True
        except sqlalchemy.exc.IntegrityError:
            return False

#-----------------------------------------------------------------------

```

PennyAdmin16Cas/database.py (Page 2 of 3)

```python
def delete_book(isbn):

    with sqlalchemy.orm.Session(_engine) as session:
        session.query(Book).filter(Book.isbn==isbn).delete()
        session.commit()

#-----------------------------------------------------------------------

def is_authorized(username):

    with sqlalchemy.orm.Session(_engine) as session:
        query = session.query(AuthorizedUser) \
            .filter(AuthorizedUser.username==username)
        try:
            query.one()
            return True
        except sqlalchemy.exc.NoResultFound:
            return False

#-----------------------------------------------------------------------

# For testing:

def _write_books(books):
    for book in books:
        print('%s | %s | %s' % (book['isbn'], book['author'],
            book['title']))

def _test():
    print('-----------------------------------')
    print('Testing get_books()')
    print('-----------------------------------')
    print()
    books = get_books()
    _write_books(books)
    print()

    print('-----------------------------------')
    print('Testing add_book()')
    print('-----------------------------------')
    print()
    successful = add_book('456', 'Kernighan', 'New Book')
    if successful:
        print('Add was successful')
        print()
        books = get_books()
        _write_books(books)
        print()
    else:
        print('Add was unsuccessful')
        print()
        _write_books(books)
        print()
    successful = add_book('456', 'Kernighan', 'New Book')
    if successful:
        print('Add was successful')
        print()
        books = get_books()
        _write_books(books)
        print()
    else:
        print('Add was unsuccessful')
        print()
        _write_books(books)
        print()
```

PennyAdmin16Cas/database.py (Page 3 of 3)

```

    print('----------------------------------')
    print('Testing delete_book()')
    print('----------------------------------')
    print()
    delete_book('456')
    books = get_books()
    _write_books(books)
    print()
    delete_book('456')
    books = get_books()
    _write_books(books)
    print()

    print('----------------------------------')
    print('Testing is_authorized()')
    print('----------------------------------')
    print()
    print(is_authorized('rdondero'))
    print(is_authorized('rdondero2'))

if __name__ == '__main__':
    _test()
```

blank (Page 1 of 1)

```
This page is intentionally blank.
```

PennyAdmin16Cas/auth.py (Page 1 of 2)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# auth.py
# Authors: Alex Halderman, Scott Karlin, Brian Kernighan, Bob Dondero
#-----------------------------------------------------------------------

import urllib.request
import urllib.parse
import re
import flask

from top import app

#-----------------------------------------------------------------------

_CAS_URL = 'https://fed.princeton.edu/cas/'

#-----------------------------------------------------------------------

# Return url after stripping out the "ticket" parameter that was
# added by the CAS server.

def strip_ticket(url):
    if url is None:
        return "something is badly wrong"
    url = re.sub(r'ticket=[^&]*&?', '', url)
    url = re.sub(r'\?&?$|&$', '', url)
    return url

#-----------------------------------------------------------------------

# Validate a login ticket by contacting the CAS server. If
# valid, return the user's username; otherwise, return None.

def validate(ticket):
    val_url = (_CAS_URL + "validate" + '?service='
        + urllib.parse.quote(strip_ticket(flask.request.url))
        + '&ticket=' + urllib.parse.quote(ticket))
    lines = []
    with urllib.request.urlopen(val_url) as flo:
        lines = flo.readlines()   # Should return 2 lines.
    if len(lines) != 2:
        return None
    first_line = lines[0].decode('utf-8')
    second_line = lines[1].decode('utf-8')
    if not first_line.startswith('yes'):
        return None
    return second_line

#-----------------------------------------------------------------------

# Authenticate the remote user, and return the user's username.
# Do not return unless the user is successfully authenticated.

def authenticate():

    # If the username is in the session, then the user was
    # authenticated previously.  So return the username.
    if 'username' in flask.session:
        return flask.session.get('username')

    # If the request does not contain a login ticket, then redirect
    # the browser to the login page to get one.
    ticket = flask.request.args.get('ticket')
```

PennyAdmin16Cas/auth.py (Page 2 of 2)

```
    if ticket is None:
        login_url = (_CAS_URL + 'login?service=' +
            urllib.parse.quote(flask.request.url))
        flask.abort(flask.redirect(login_url))

    # If the login ticket is invalid, then redirect the browser
    # to the login page to get a new one.
    username = validate(ticket)
    if username is None:
        login_url = (_CAS_URL + 'login?service='
            + urllib.parse.quote(strip_ticket(flask.request.url)))
        flask.abort(flask.redirect(login_url))

    # The user is authenticated, so store the username in
    # the session.
    username = username.strip().lower()
    flask.session['username'] = username
    return username

#-----------------------------------------------------------------------

@app.route('/logoutapp', methods=['GET'])
def logoutapp():

    # Log out of the application.
    flask.session.clear()
    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/logoutcas', methods=['GET'])
def logoutcas():

    # Log out of the CAS session, and then the application.
    logout_url = (_CAS_URL + 'logout?service='
        + urllib.parse.quote(
            re.sub('logoutcas', 'logoutapp', flask.request.url)))
    flask.abort(flask.redirect(logout_url))
```

CAS

- See **<u>PennyAdmin16Cas</u>** app (cont.)
  - How it works…
  - See **Appendix 1**

CAS

- **Pros**
  - Application need not manage usernames or passwords
  - Application ***cannot*** access passwords!
  - Application is constrained to one user community

CAS

- **Cons**
  - Complex
  - Adds overhead, but only during user’s first visit to the app per browser session
  - Application is constrained to one user community!

Agenda

- Data comm attacks
- Third-party authentication (briefly)
  - CAS
  - **Google authentication**
  - Auth0 authentication

Google Authentication

- See **PennyAdmin17Google** app
  - **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 **PennyAdmin17Google** app (cont.)
  - How to run it on your local computer…

Google Authentication

- **Preliminary**
  - Make sure these packages are installed (via `pip`) in your Python virtual environment

```
Flask
python-dotenv
oauthlib
requests
```

Google Authentication

- **Preliminary**
  - Create a self-signed certificate (as described previously in this lecture)

```
$ 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 []:
$
```

Google Authentication

- **Preliminary**
  - Strongly suggested…
  - Create a ***project Google account*** (i.e., a gmail address) for your project team
    - Use your project Google account exclusively for Google authentication setup and subsequent app testing

Google Authentication

- **Preliminary**
  - Register app (https://localhost:5000) as a client of Google
    - Log into Google using your project Google account
    - Browse to https://console.developers.google.com/apis/credentials
    - Click *CREATE PROJECT*
    - For *Project name* enter `Penny`
    - Click *CREATE*

Google Authentication

- **Preliminary**
  - Register app (https://localhost:5000) as a client of Google (cont.)
    - Click CONFIGURE CONSENT SCREEN
    - For User Type choose *External*
    - Click *CREATE*
    - For *App name* enter `Penny`
    - For *User support email* enter your your project gmail address
    - For *Developer contact information* enter your project gmail address
    - Click *SAVE AND CONTINUE* a few times to finish the consent

Google Authentication

- **Preliminary**
  - Register app (https://localhost:5000) as a client of Google (cont.)
    - Click Credentials
    - Click *Create Credentials*, *OAuth client ID*, *Web Application*
    - In Authorized JavaScript origins:
      - Click ADD URI
      - Enter https://localhost:5000
    - In Authorized redirect URIs:
      - Click ADD URI
      - Add Authorized Redirect URI: https://localhost:5000/login/callback

Google Authentication

- **Preliminary**
  - Register app (https://localhost:5000) as a client of Google (cont.)
    - Google provides **`GOOGLE_CLIENT_ID`** and **`GOOGLE_CLIENT_SECRET`**
      - Take note of them!

Google Authentication

Create environment variables:

```
APP_SECRET_KEY=yourappsecretkey
GOOGLE_CLIENT_ID=yourgoogleclientid
GOOGLE_CLIENT_SECRET=yourgoogleclientsecret
```

Google Authentication

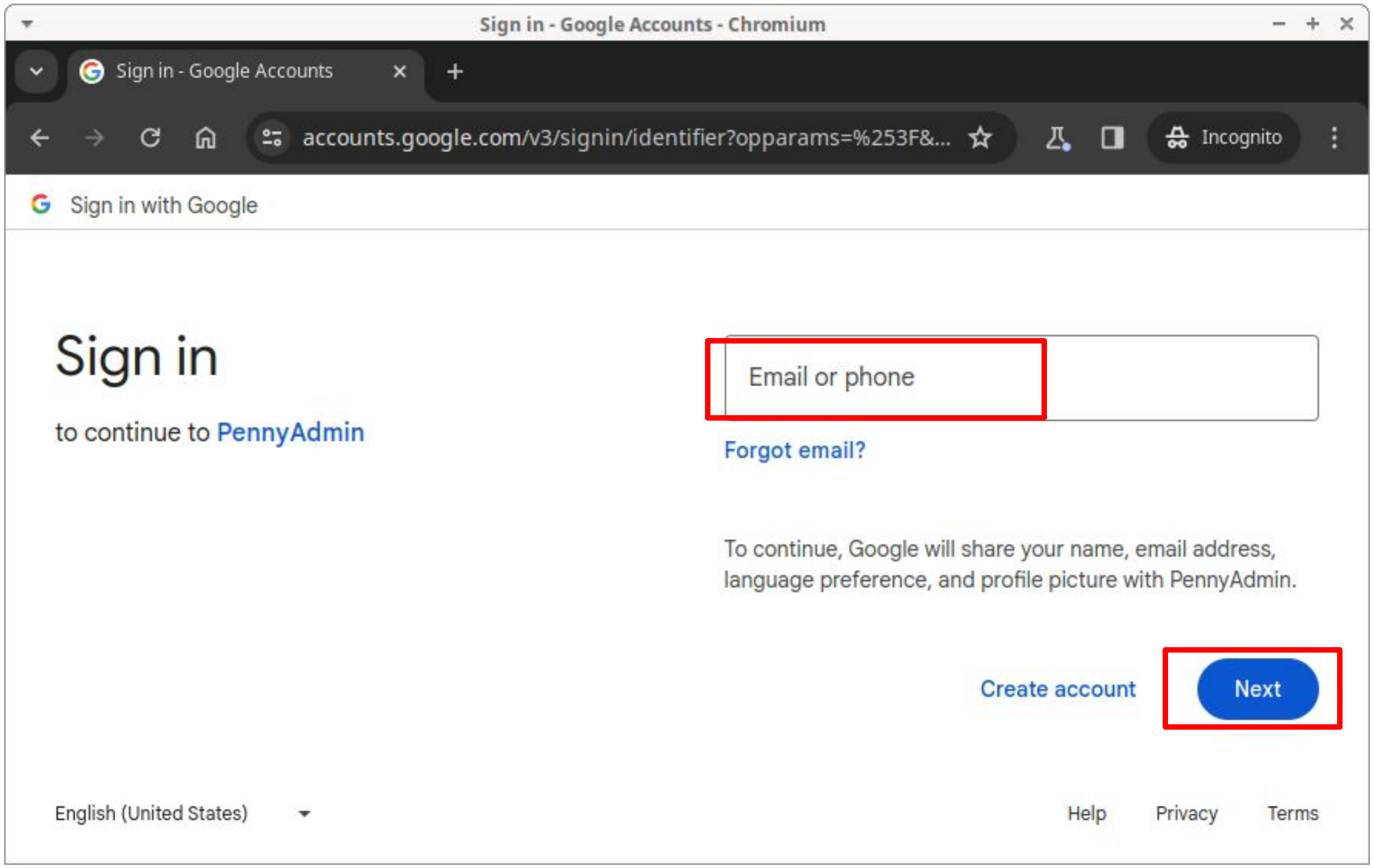
- See **PennyAdmin17Google** app (cont.)
  - In terminal, enter this command:

```
$ python runserver.py
```

    - Runs Flask test server on port 5000
    - Runs Flask test server using HTTPS
  - In browser, enter URL:
    - https://localhost:5000

Google Authentication

- See **<u>PennyAdmin17Google</u>** app (cont.)


Sign in - Google Accounts - Chromium

Sign in - Google Accounts

accounts.google.com/v3/signin/identifier?opparams=%253F&...

Incognito

Sign in with Google

Sign in

to continue to PennyAdmin

Email or phone

Forgot email?

To continue, Google will share your name, email address, language preference, and profile picture with PennyAdmin.

Create account

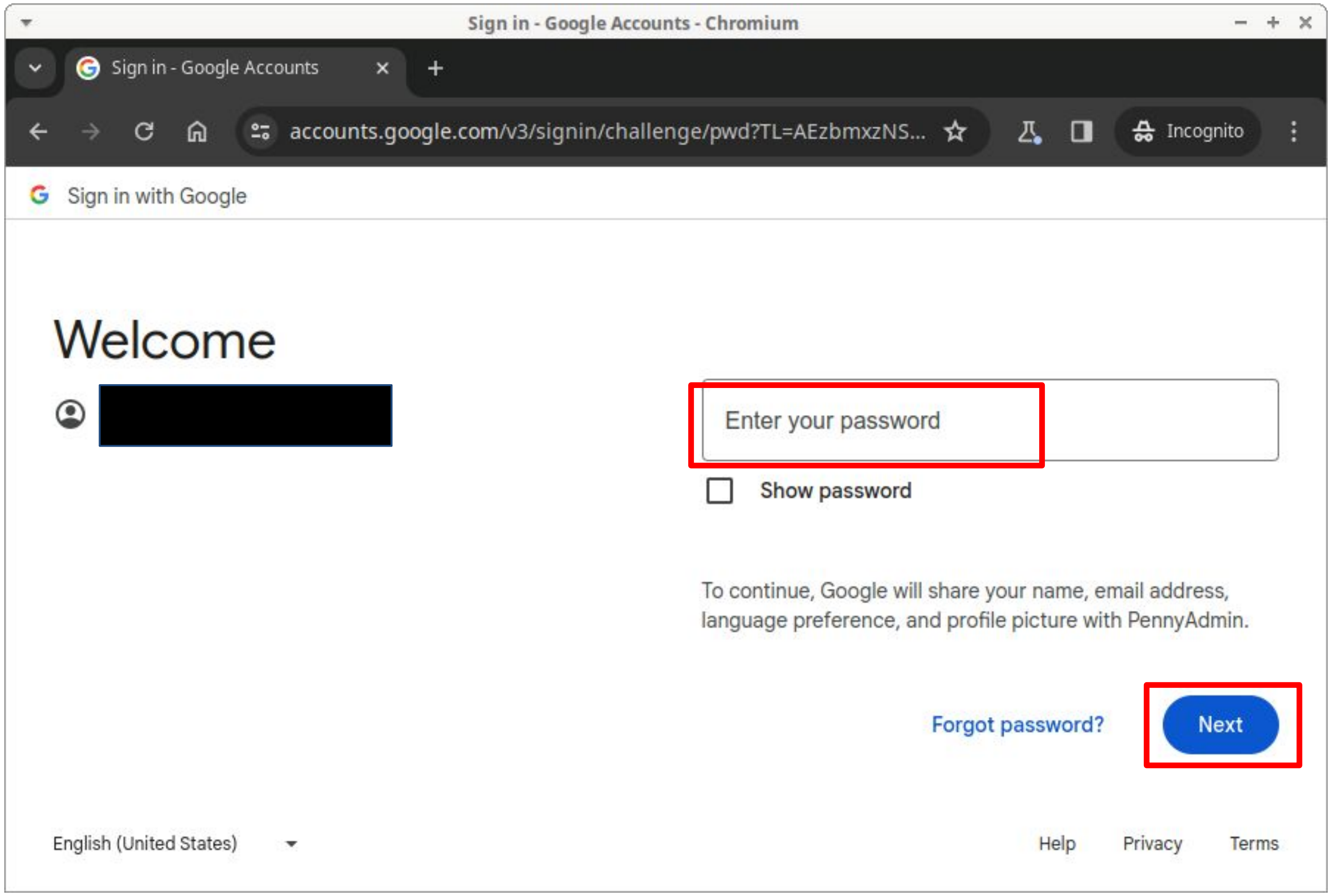
Next

English (United States)

Help Privacy Terms

Google Authentication

- See **PennyAdmin17Google** app (cont.)



Sign in - Google Accounts - Chromium

Sign in - Google Accounts

accounts.google.com/v3/signin/challenge/pwd?TL=AEzbmxzNS...

Sign in with Google

Welcome

[redacted]

Enter your password

Show password

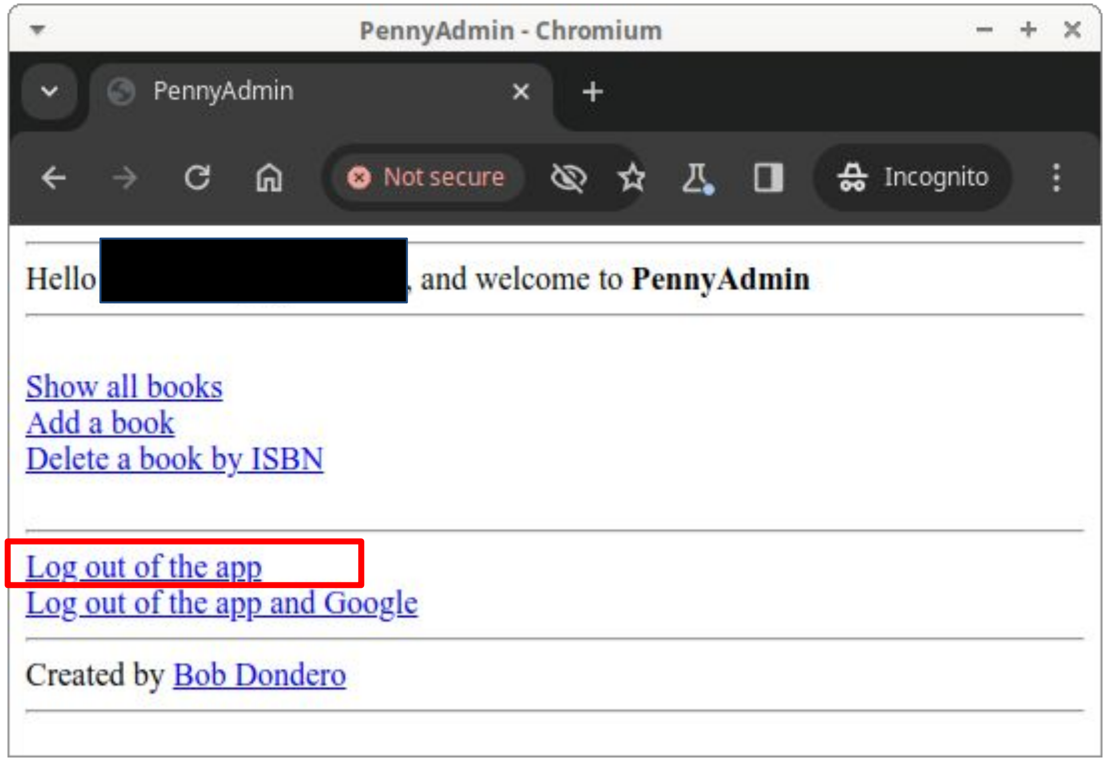
To continue, Google will share your name, email address, language preference, and profile picture with PennyAdmin.

Forgot password? Next

English (United States) Help Privacy Terms

Google Authentication

- See **PennyAdmin17Google** app (cont.)



Hello [redacted], and welcome to **PennyAdmin**

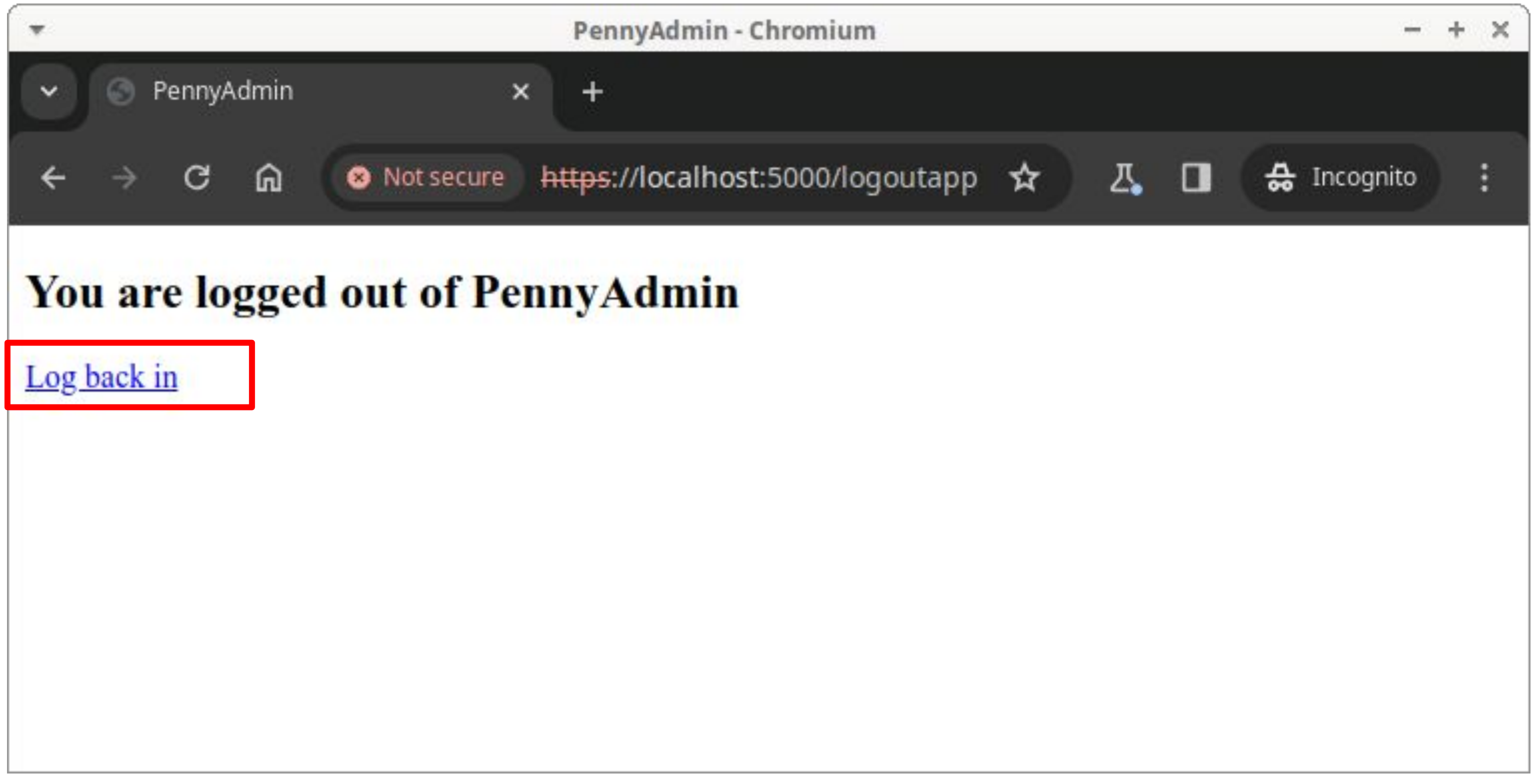
Show all books
Add a book
Delete a book by ISBN

Log out of the app
Log out of the app and Google

Created by Bob Dondero

Google Authentication

- See **PennyAdmin17Google** app (cont.)

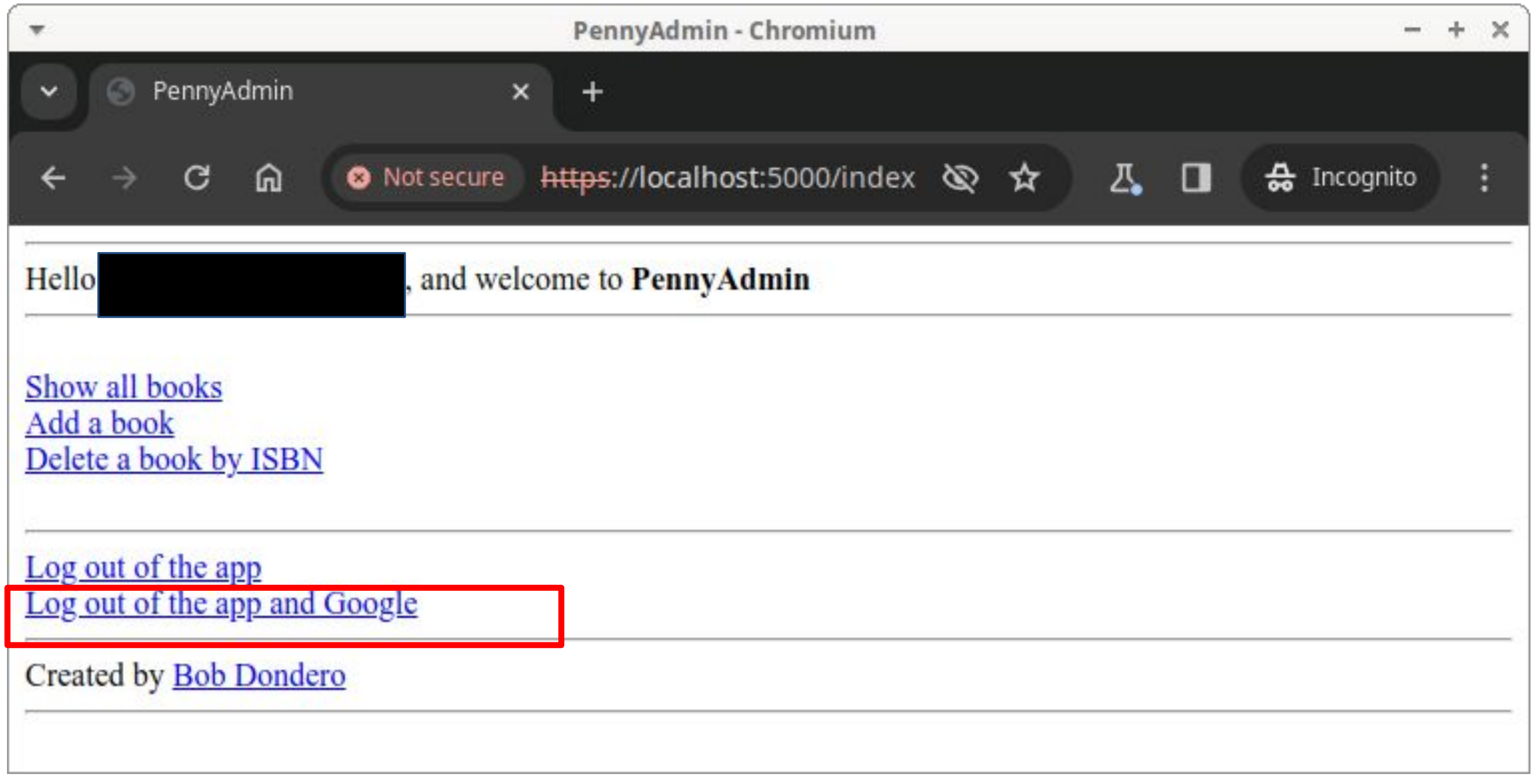


You are logged out of PennyAdmin

Log back in

Google Authentication

- See **PennyAdmin17Google** app (cont.)



Hello [redacted], and welcome to **PennyAdmin**

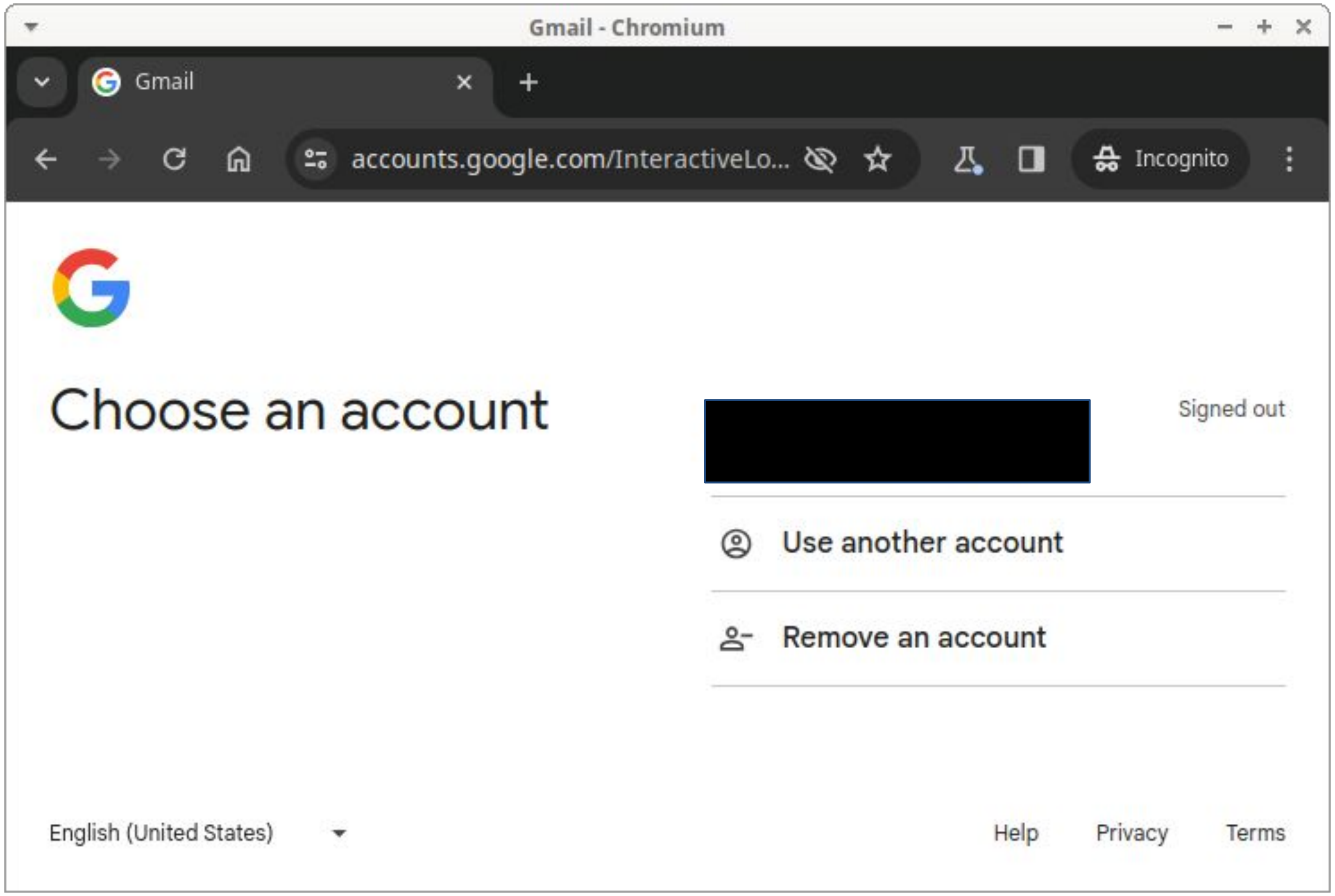
Show all books
Add a book
Delete a book by ISBN

Log out of the app
Log out of the app and Google

Created by Bob Dondero

Google Authentication

- See **PennyAdmin17Google** app (cont.)



Gmail - Chromium

accounts.google.com/InteractiveLo...

Choose an account

[redacted] Signed out

Use another account

Remove an account

English (United States) Help Privacy Terms

How to show loggedout page?

Google Authentication

- See **PennyAdmin17Google** (cont.)
  - How to run it on Render (or Heroku, or any cloud service)…

Google Authentication

- Preliminary
  - Deploy the app to Render
    - Push the app to a GitHub repo
    - Create a new Render app linked to the GitHub repo
    - Deploy the application from GitHub to Render
  - Configure the Render app
    - Create env vars APP_SECRET_KEY, GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET

Google Authentication

- Preliminary (cont.)
  - All preliminaries are the same, except:
    - For *Authorized JavaScript origins* enter the URL of your deployed application
    - For *Authorized redirect URIs* enter the callback URL of your deployed application
- In browser, enter URL:
    - https://*ipaddressofrenderapp*

Google Authentication

- See <u>**PennyAdmin17Google**</u> app (cont.)
  - How it works…
  - See **Appendix 2**

Google Authentication

- See **PennyAdmin17Google** app (cont.)
  - **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**

PennyAdmin17Google/runserver.py (Page 1 of 1)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# runserver.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import sys
import penny

# Google expects the application to run on port 5000.
PORT = 5000

def main():

    if len(sys.argv) != 1:
        print('Usage: ' + sys.argv[0], file=sys.stderr)
        sys.exit(1)

    try:
        penny.app.run(host='0.0.0.0', port=PORT, debug=True,
            ssl_context=('cert.pem', 'key.pem'))
    except Exception as ex:
        print(ex, file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

blank (Page 1 of 1)

```
This page is intentionally blank.
```

PennyAdmin17Google/penny.sql (Page 1 of 1)

```
DROP TABLE IF EXISTS books;
CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);
INSERT INTO books (isbn, author, title)
   VALUES ('123', 'Kernighan','The Practice of Programming');
INSERT INTO books (isbn, author, title)
   VALUES ('234', 'Kernighan','The C Programming Language');
INSERT INTO books (isbn, author, title)
   VALUES ('345', 'Sedgewick','Algorithms in C');

DROP TABLE IF EXISTS authorizedusers;
CREATE TABLE authorizedusers (username TEXT);
INSERT INTO authorizedusers (username)
   VALUES ('donderorobert@gmail.com');
INSERT INTO authorizedusers (username)
   VALUES ('bwk@gmail.com');
```

PennyAdmin17Google/footer.html (Page 1 of 1)

```
<hr>
<a href="logoutapp">Log out of the app</a></br>
<a href="logoutgoogle">Log out of the app and Google</a></br>
<hr>
Created by <a href="https://www.cs.princeton.edu/~rdondero">
Bob Dondero</a>
<hr>
```

PennyAdmin17Google/database.py (Page 1 of 3)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# database.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import sqlalchemy
import sqlalchemy.orm
import dotenv

#-----------------------------------------------------------------------

dotenv.load_dotenv()
_database_url = os.getenv('DATABASE_URL', 'sqlite:///penny.sqlite')
_database_url = _database_url.replace('postgres://', 'postgresql://')

#-----------------------------------------------------------------------

Base = sqlalchemy.orm.declarative_base()

class Book (Base):
    __tablename__ = 'books'
    isbn = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
    author = sqlalchemy.Column(sqlalchemy.String)
    title = sqlalchemy.Column(sqlalchemy.String)

class AuthorizedUser (Base):
    __tablename__ = 'authorizedusers'
    username = sqlalchemy.Column(sqlalchemy.String, primary_key=True)

_engine = sqlalchemy.create_engine(_database_url)

#-----------------------------------------------------------------------

def get_books():

    books = []

    with sqlalchemy.orm.Session(_engine) as session:
        query = session.query(Book)
        table = query.all()
        for row in table:
            book = {'isbn': row.isbn, 'author': row.author,
                'title': row.title}
            books.append(book)

    return books

#-----------------------------------------------------------------------

def add_book(isbn, author, title):

    with sqlalchemy.orm.Session(_engine) as session:
        row = Book(isbn=isbn, author=author, title=title)
        session.add(row)
        try:
            session.commit()
            return True
        except sqlalchemy.exc.IntegrityError:
            return False

#-----------------------------------------------------------------------

```

PennyAdmin17Google/database.py (Page 2 of 3)

```
def delete_book(isbn):

    with sqlalchemy.orm.Session(_engine) as session:
        session.query(Book).filter(Book.isbn==isbn).delete()
        session.commit()

#-----------------------------------------------------------------------

def is_authorized(username):

    with sqlalchemy.orm.Session(_engine) as session:
        query = session.query(AuthorizedUser) \
            .filter(AuthorizedUser.username==username)
        try:
            query.one()
            return True
        except sqlalchemy.exc.NoResultFound:
            return False

#-----------------------------------------------------------------------

# For testing:

def _write_books(books):
    for book in books:
        print('%s | %s | %s' % (book['isbn'], book['author'],
            book['title']))

def _test():
    print('-----------------------------------')
    print('Testing get_books()')
    print('-----------------------------------')
    print()
    books = get_books()
    _write_books(books)
    print()

    print('-----------------------------------')
    print('Testing add_book()')
    print('-----------------------------------')
    print()
    successful = add_book('456', 'Kernighan', 'New Book')
    if successful:
        print('Add was successful')
        print()
        books = get_books()
        _write_books(books)
        print()
    else:
        print('Add was unsuccessful')
        print()
        _write_books(books)
        print()
    successful = add_book('456', 'Kernighan', 'New Book')
    if successful:
        print('Add was successful')
        print()
        books = get_books()
        _write_books(books)
        print()
    else:
        print('Add was unsuccessful')
        print()
        _write_books(books)
        print()
```

PennyAdmin17Google/database.py (Page 3 of 3)

```

    print('----------------------------------')
    print('Testing delete_book()')
    print('----------------------------------')
    print()
    delete_book('456')
    books = get_books()
    _write_books(books)
    print()
    delete_book('456')
    books = get_books()
    _write_books(books)
    print()

    print('----------------------------------')
    print('Testing is_authorized()')
    print('----------------------------------')
    print()
    print(is_authorized('rdondero'))
    print(is_authorized('rdondero2'))

if __name__ == '__main__':
    _test()
```

PennyAdmin17Google/auth.py (Page 1 of 3)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#   With lots of help from https://realpython.com/flask-google-login/
#-----------------------------------------------------------------------

import os
import json
import requests
import flask
import oauthlib.oauth2
import dotenv

from top import app

#-----------------------------------------------------------------------

GOOGLE_DISCOVERY_URL = (
    'https://accounts.google.com/.well-known/openid-configuration')

dotenv.load_dotenv()
GOOGLE_CLIENT_ID = os.environ['GOOGLE_CLIENT_ID']
GOOGLE_CLIENT_SECRET = os.environ['GOOGLE_CLIENT_SECRET']

client = oauthlib.oauth2.WebApplicationClient(GOOGLE_CLIENT_ID)

#-----------------------------------------------------------------------

@app.route('/login', methods=['GET'])
def login():

    # Determine the URL for Google login.
    google_provider_cfg = requests.get(
        GOOGLE_DISCOVERY_URL, timeout=2).json()
    authorization_endpoint = (
        google_provider_cfg['authorization_endpoint'])

    # Construct the request URL for Google login, providing scopes
    # to fetch the user's profile data.
    request_uri = client.prepare_request_uri(
        authorization_endpoint,
        redirect_uri = flask.request.base_url + '/callback',
        scope=['openid', 'email', 'profile'],
    )

    #-------------------------------------------------------------------
    # For learning:
    # print('request_uri:', request_uri, file=sys.stderr)
    #-------------------------------------------------------------------

    # Redirect to the request URL.
    return flask.redirect(request_uri)

#-----------------------------------------------------------------------

@app.route('/login/callback', methods=['GET'])
def callback():

    # Get the authorization code that Google sent.
    code = flask.request.args.get('code')

    #-------------------------------------------------------------------
    # For learning:
```

Google Authentication

- **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, …

Google Authentication

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

Google Authentication

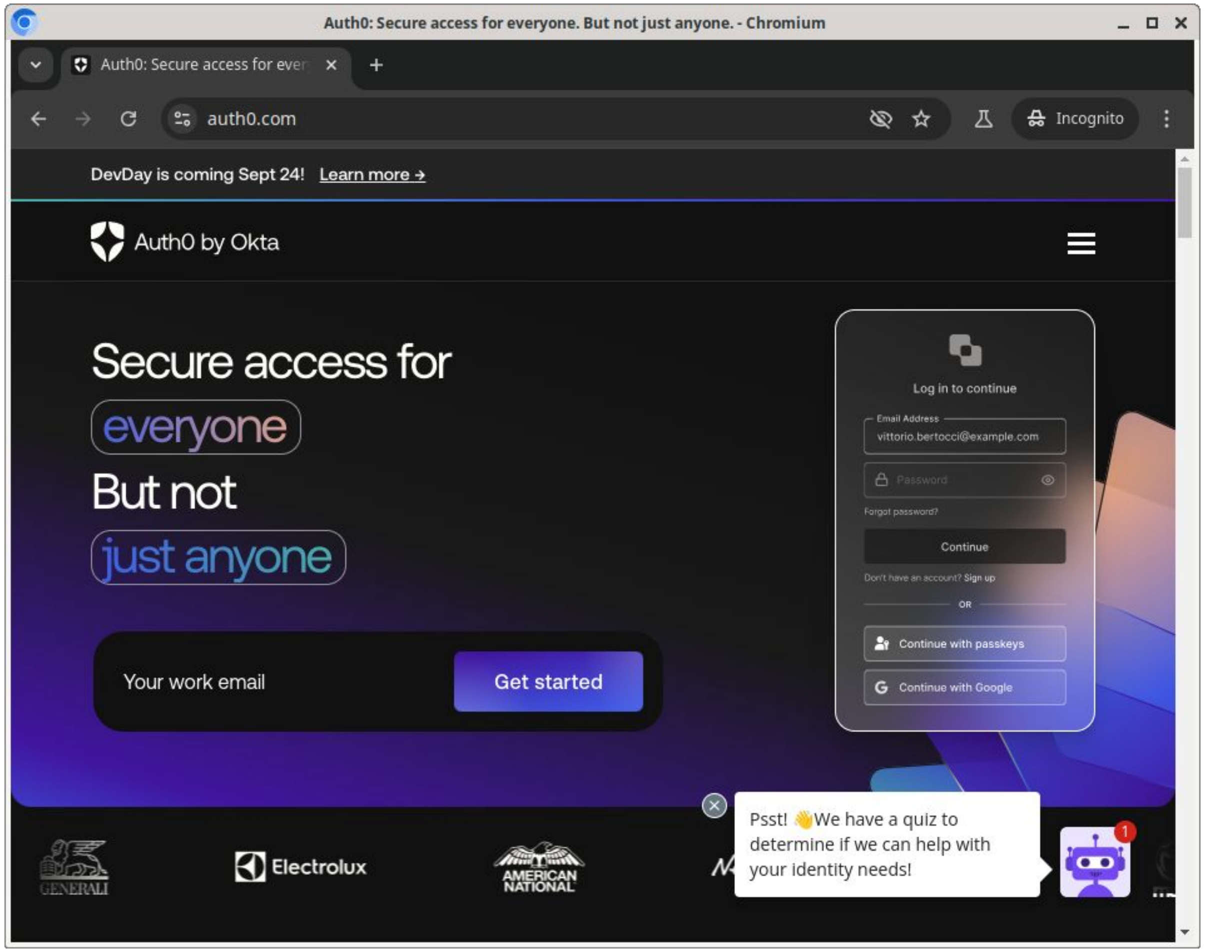
- For more information...
- https://realpython.com/flask-google-login/

Agenda

- Data comm attacks
- Third-party authentication (briefly)
  - CAS
  - Google authentication
  - **Auth0 authentication**

Auth0 Authentication

- ***Auth0*** https://auth0.com


Auth0: Secure access for everyone. But not just anyone. - Chromium
Auth0: Secure access for ever
auth0.com
Incognito
DevDay is coming Sept 24! Learn more →
Auth0 by Okta
Secure access for everyone But not just anyone
Your work email
Get started
Log in to continue
Email Address
vittorio.bertocci@example.com
Password
Forgot password?
Continue
Don't have an account? Sign up
OR
Continue with passkeys
Continue with Google
Psst! 👋We have a quiz to determine if we can help with your identity needs!
GENERALI
Electrolux
AMERICAN NATIONAL

Auth0 Authentication

- ***Auth0*** (cont.)
  - A commercial software product from Okta
  - Free tier
    - 7500 monthly active users
    - Password authentication with email, username, or phone number
    - Social authentication (Google, Facebook, …)
  - Paid tiers
    - SMS authentication
    - Role-based access control (authorization)
    - Multi-factor authentication
    - …

Auth0 Authentication

- See **PennyAdmin18Auth0** app
  - How to run it on your computer

Auth0 Authentication

- **Preliminary**
  - Make sure these packages are installed (via `pip`) in your Python virtual environment

```
Flask
python-dotenv
Authlib
requests
```

Auth0 Authentication

- **Preliminary**
  - Browse to this address: https://auth0.com
  - Sign up for a new account, or login to your existing account
    - Signing up for a new account requires you to provide an email address, but not a credit card number

Auth0 Authentication

- **Preliminary (cont.)**
  - Click on *Applications* -> *Applications*
  - For *Name* enter `Penny`
  - Note the *Domain*, *ClientID*, and *Client Secret*
    - You'll need them later
  - For *Allowed Callback URLs* enter `http://localhost:3000/callback`
  - For *Allowed Logout URLs* enter `http://localhost:3000/loggedout`
  - Click *Save Changes*
  - *Note these*
  - Leave the Autho0 website, if you want

Auth0 Authentication

- **Preliminary (cont.)**
  - Note these values generated by Auth0:
    - Auth0 client id
    - Auth0 client secret
    - Auth0 domain
  - Leave the Auth0 website, if you want

Auth0 Authentication

- **Preliminary (cont.)**
  - Create environment variables:

```
APP_SECRET_KEY=<any secret key you want>
AUTH0_CLIENT_ID=<the Auth0 ClientID>
AUTH0_CLIENT_SECRET=<the Auth0 Client Secret>
AUTH0_DOMAIN=<the Auth0 Domain>
```

Auth0 Authentication

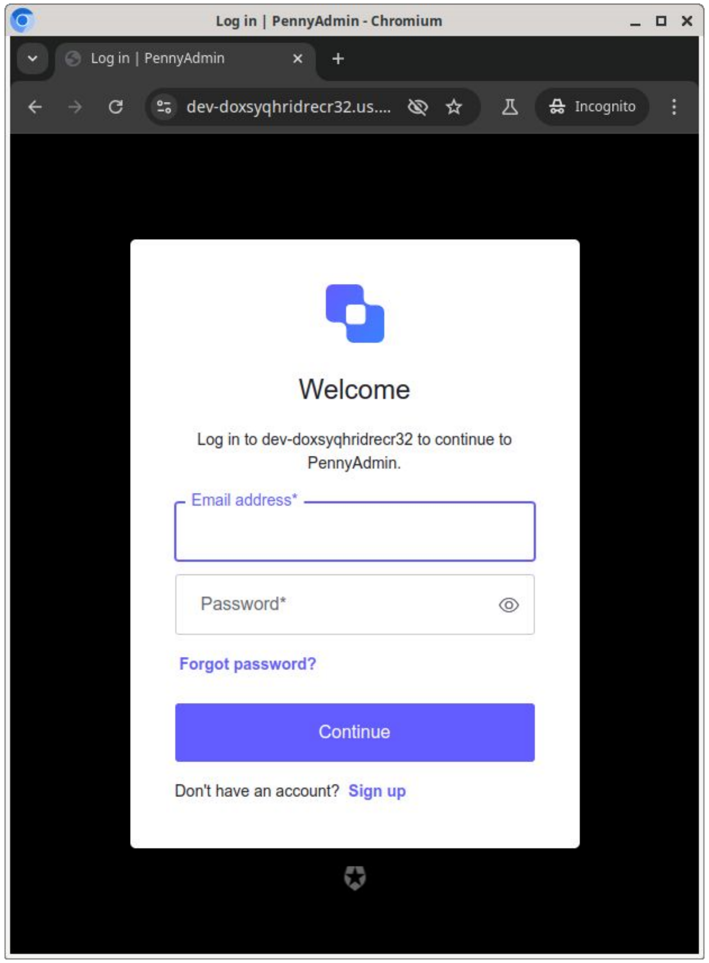
- See **PennyAdmin18Auth0** app (cont.)
  - In terminal, enter this command:

```
$ python runserver.py
```

    - Runs Flask test server on port 3000
  - In browser, enter URL:
    - https://localhost:3000

Auth0 Authentication

- See **PennyAdmin18Auth0** app (cont.)



Welcome

Log in to dev-doxsyqhridrecr32 to continue to PennyAdmin.

Email address*

Password*

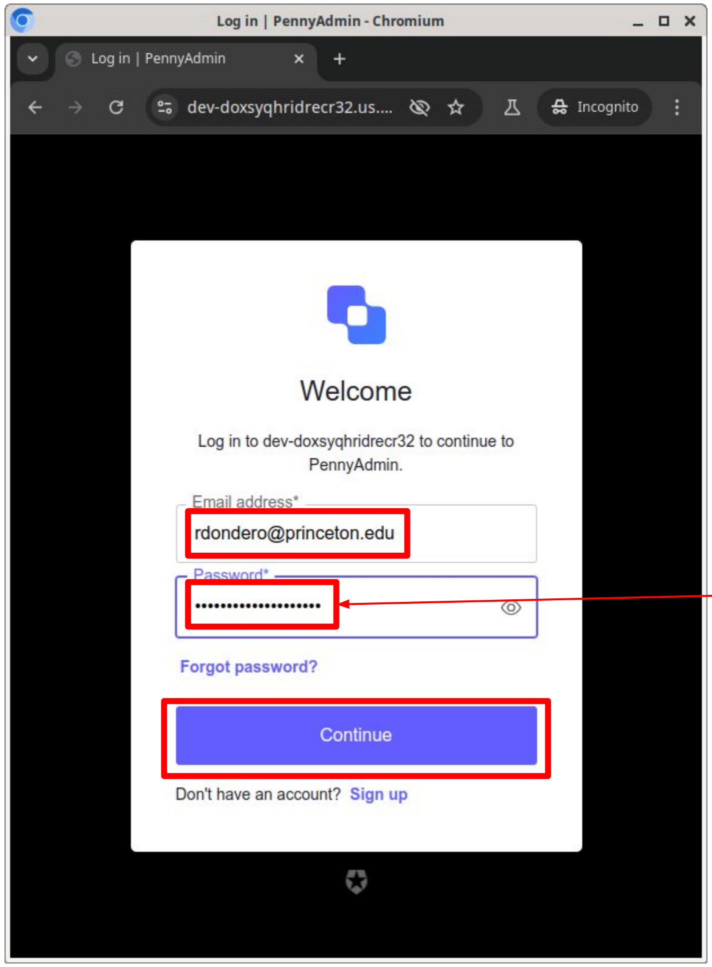
Forgot password?

Continue

Don't have an account? Sign up

Auth0 Authentication

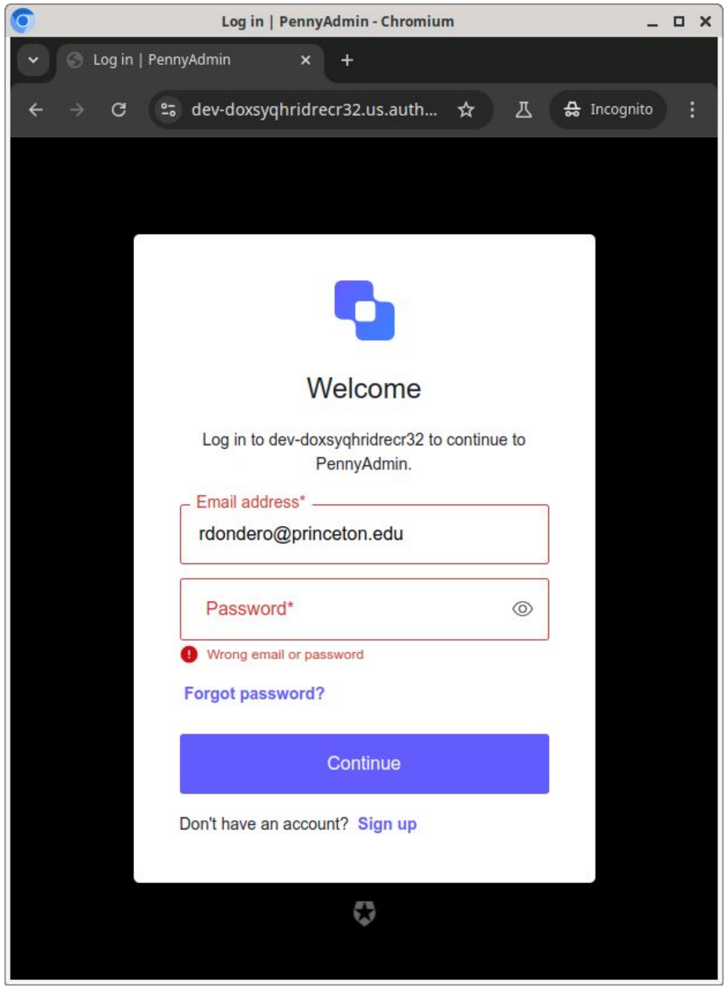
- See **PennyAdmin18Auth0** app (cont.)


Log in | PennyAdmin - Chromium
Log in | PennyAdmin
dev-doxsyqhridrecr32.us....
Incognito
Welcome
Log in to dev-doxsyqhridrecr32 to continue to PennyAdmin.
Email address*
rdondero@princeton.edu
Password*
Forgot password?
Continue
Don't have an account? Sign up

Wrong password

Auth0 Authentication

- See **<u>PennyAdmin18Auth0</u>** app (cont.)



Log in | PennyAdmin - Chromium

Log in | PennyAdmin

dev-doxsyqhridrecr32.us.auth...

Incognito

Welcome

Log in to dev-doxsyqhridrecr32 to continue to PennyAdmin.

Email address*

rdondero@princeton.edu

Password*

Wrong email or password

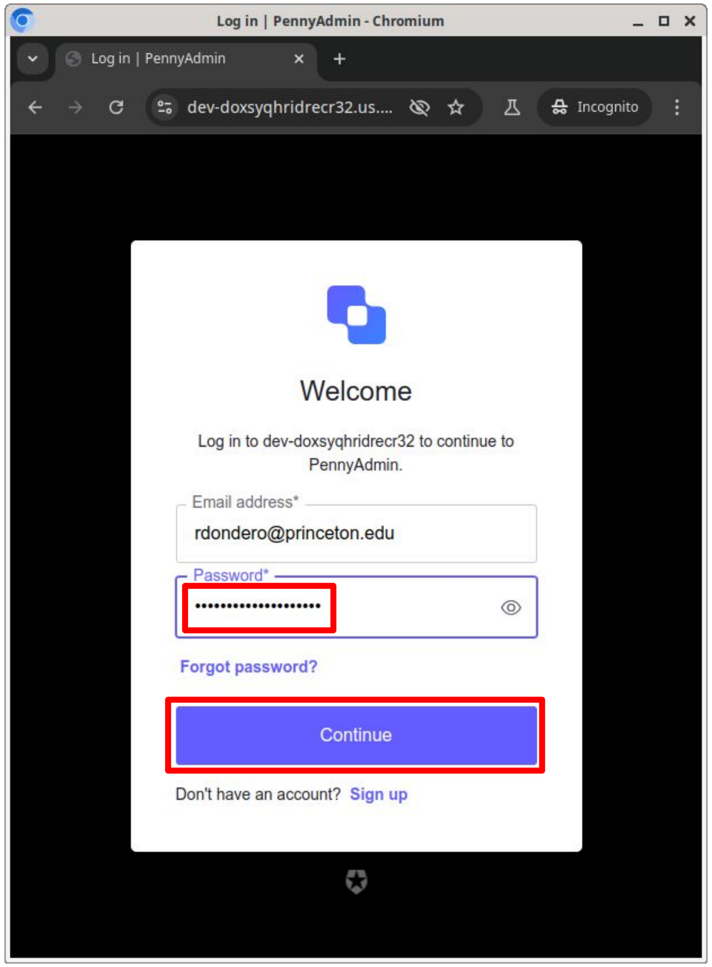
Forgot password?

Continue

Don't have an account? Sign up

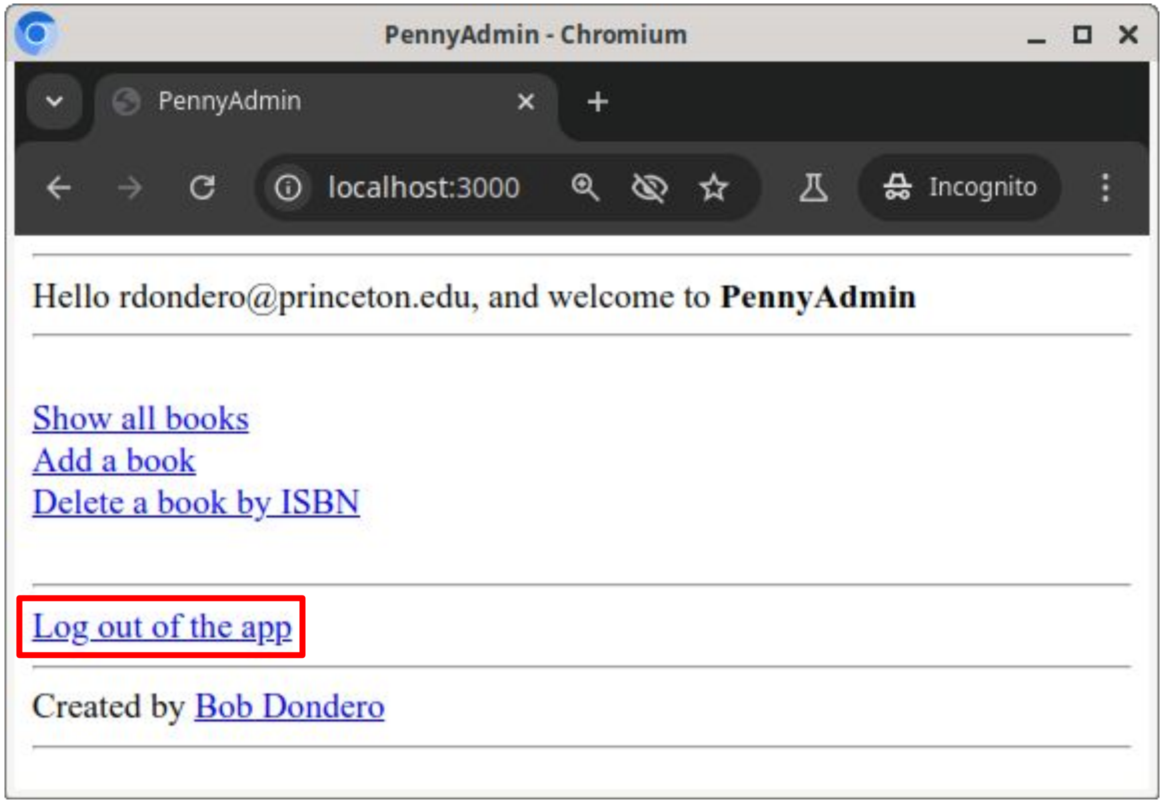
Auth0 Authentication

- See **<u>PennyAdmin18Auth0</u>** app (cont.)


Log in | PennyAdmin - Chromium
Log in | PennyAdmin
dev-doxsyqhridrecr32.us....
Incognito
Welcome
Log in to dev-doxsyqhridrecr32 to continue to PennyAdmin.
Email address*
rdondero@princeton.edu
Password*
Forgot password?
Continue
Don't have an account? Sign up

Auth0 Authentication

- See **PennyAdmin18Auth0** app (cont.)



Hello rdondero@princeton.edu, and welcome to **PennyAdmin**

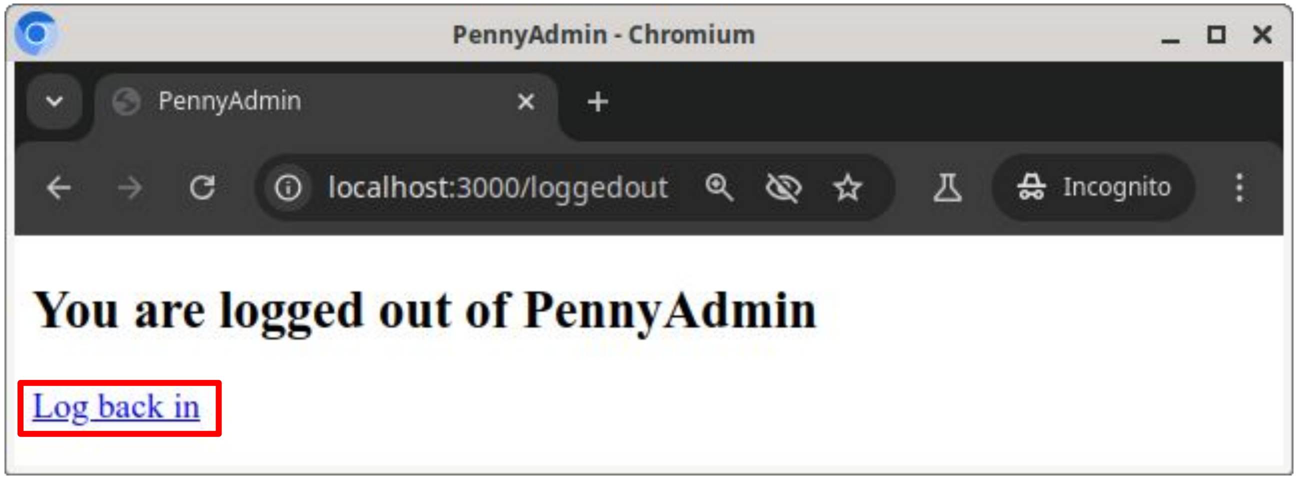
Show all books
Add a book
Delete a book by ISBN

Log out of the app

Created by Bob Dondero

Auth0 Authentication

- See **PennyAdmin18Auth0** app (cont.)

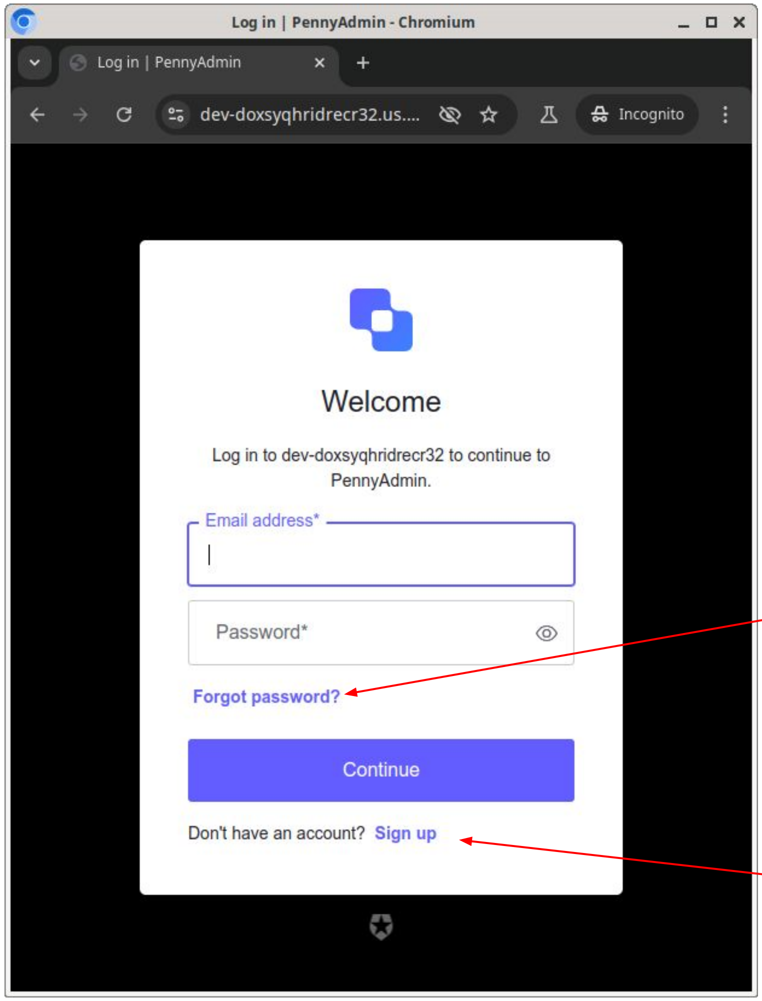


You are logged out of PennyAdmin

Log back in

Auth0 Authentication

- See **PennyAdmin18Auth0** app (cont.)



Log in | PennyAdmin - Chromium

Log in | PennyAdmin

dev-doxsyqhridrecr32.us....

Incognito

Welcome

Log in to dev-doxsyqhridrecr32 to continue to PennyAdmin.

Email address*

Password*

Forgot password?

Continue

Don't have an account? Sign up

Sends email

Verifies email (but doesn't block sign-up)

Auth0 Authentication

- See **PennyAdmin18Auth0** app
  - How to run it on Render.com

Auth0 Authentication

- **Preliminary**
  - Deploy the app to Render
    - Push the app to a GitHub repo
    - Create a new Render app linked to the GitHub repo
    - Deploy the application from GitHub to Render
  - Configure the Render app
    - Create env vars APP_SECRET_KEY, AUTH0_CLIENT_ID, AUTH0_CLIENT_SECRET, and AUTH0_DOMAIN

Auth0 Authentication

- **Preliminary (cont.)**
  - Log into the Auth0 website
  - Add the appropriate Render URLs to *Allowed Callback URLs* and *Allowed Logout URLs*

- Browse to the application as usual

Auth0 Authentication

- See **PennyAdmin18Auth0** app (cont.)
  - **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**

PennyAdmin18Auth0/runserver.py (Page 1 of 1)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# runserver.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import sys
import penny

PORT = 3000

def main():

    if len(sys.argv) != 1:
        print('Usage: ' + sys.argv[0], file=sys.stderr)
        sys.exit(1)

    try:
        penny.app.run(host='0.0.0.0', port=PORT, debug=True)
    except Exception as ex:
        print(ex, file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

blank (Page 1 of 1)

```
This page is intentionally blank.
```

PennyAdmin18Auth0/penny.sql (Page 1 of 1)

```
DROP TABLE IF EXISTS books;
CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);
INSERT INTO books (isbn, author, title)
   VALUES ('123', 'Kernighan','The Practice of Programming');
INSERT INTO books (isbn, author, title)
   VALUES ('234', 'Kernighan','The C Programming Language');
INSERT INTO books (isbn, author, title)
   VALUES ('345', 'Sedgewick','Algorithms in C');

DROP TABLE IF EXISTS authorizedusers;
CREATE TABLE authorizedusers (username TEXT);
INSERT INTO authorizedusers (username) VALUES
   ('rdondero@princeton.edu');
INSERT INTO authorizedusers (username) VALUES
   ('bwk@princeton.edu');
```

PennyAdmin18Auth0/footer.html (Page 1 of 1)

```
<hr>
<a href="/logout">Log out of the app</a></br>
<hr>
Created by <a href="https://www.cs.princeton.edu/~rdondero">
Bob Dondero</a>
<hr>
```

PennyAdmin18Auth0/database.py (Page 1 of 3)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# database.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import sqlalchemy
import sqlalchemy.orm
import dotenv

#-----------------------------------------------------------------------

dotenv.load_dotenv()
_database_url = os.getenv('DATABASE_URL', 'sqlite:///penny.sqlite')
_database_url = _database_url.replace('postgres://', 'postgresql://')

#-----------------------------------------------------------------------

Base = sqlalchemy.orm.declarative_base()

class Book (Base):
    __tablename__ = 'books'
    isbn = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
    author = sqlalchemy.Column(sqlalchemy.String)
    title = sqlalchemy.Column(sqlalchemy.String)

class AuthorizedUser (Base):
    __tablename__ = 'authorizedusers'
    username = sqlalchemy.Column(sqlalchemy.String, primary_key=True)

_engine = sqlalchemy.create_engine(_database_url)

#-----------------------------------------------------------------------

def get_books():

    books = []

    with sqlalchemy.orm.Session(_engine) as session:
        query = session.query(Book)
        table = query.all()
        for row in table:
            book = {'isbn': row.isbn, 'author': row.author,
                'title': row.title}
            books.append(book)

    return books

#-----------------------------------------------------------------------

def add_book(isbn, author, title):

    with sqlalchemy.orm.Session(_engine) as session:
        row = Book(isbn=isbn, author=author, title=title)
        session.add(row)
        try:
            session.commit()
            return True
        except sqlalchemy.exc.IntegrityError:
            return False

#-----------------------------------------------------------------------

```

PennyAdmin18Auth0/database.py (Page 2 of 3)

```python
def delete_book(isbn):

    with sqlalchemy.orm.Session(_engine) as session:
        session.query(Book).filter(Book.isbn==isbn).delete()
        session.commit()

#-----------------------------------------------------------------------

def is_authorized(username):

    with sqlalchemy.orm.Session(_engine) as session:
        query = session.query(AuthorizedUser) \
            .filter(AuthorizedUser.username==username)
        try:
            query.one()
            return True
        except sqlalchemy.exc.NoResultFound:
            return False

#-----------------------------------------------------------------------

# For testing:

def _write_books(books):
    for book in books:
        print('%s | %s | %s' % (book['isbn'], book['author'],
            book['title']))

def _test():
    print('-----------------------------------')
    print('Testing get_books()')
    print('-----------------------------------')
    print()
    books = get_books()
    _write_books(books)
    print()

    print('-----------------------------------')
    print('Testing add_book()')
    print('-----------------------------------')
    print()
    successful = add_book('456', 'Kernighan', 'New Book')
    if successful:
        print('Add was successful')
        print()
        books = get_books()
        _write_books(books)
        print()
    else:
        print('Add was unsuccessful')
        print()
        _write_books(books)
        print()
    successful = add_book('456', 'Kernighan', 'New Book')
    if successful:
        print('Add was successful')
        print()
        books = get_books()
        _write_books(books)
        print()
    else:
        print('Add was unsuccessful')
        print()
        _write_books(books)
        print()
```

PennyAdmin18Auth0/database.py (Page 3 of 3)

```

    print('-----------------------------------')
    print('Testing delete_book()')
    print('-----------------------------------')
    print()
    delete_book('456')
    books = get_books()
    _write_books(books)
    print()
    delete_book('456')
    books = get_books()
    _write_books(books)
    print()

    print('-----------------------------------')
    print('Testing is_authorized()')
    print('-----------------------------------')
    print()
    print(is_authorized('rdondero'))
    print(is_authorized('rdondero2'))

if __name__ == '__main__':
    _test()
```

PennyAdmin18Auth0/auth.py (Page 1 of 2)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import urllib.parse
import flask
import dotenv
import authlib.integrations.flask_client

from top import app

#-----------------------------------------------------------------------

dotenv.load_dotenv()
AUTH0_CLIENT_ID = os.environ.get('AUTH0_CLIENT_ID')
AUTH0_CLIENT_SECRET = os.environ.get('AUTH0_CLIENT_SECRET')
AUTH0_DOMAIN = os.environ.get("AUTH0_DOMAIN")

oauth = authlib.integrations.flask_client.OAuth(app)

oauth.register(
    'auth0',
    client_id=AUTH0_CLIENT_ID,
    client_secret=AUTH0_CLIENT_SECRET,
    client_kwargs={'scope': 'openid profile email'},
    server_metadata_url='https://' + AUTH0_DOMAIN +
        '/.well-known/openid-configuration')

#-----------------------------------------------------------------------

@app.route('/callback', methods=["GET", "POST"])
def callback():
    try:
        token = oauth.auth0.authorize_access_token()
        flask.session['user'] = token
    except Exception:
        pass
    return flask.redirect('/')

#-----------------------------------------------------------------------

@app.route('/login', methods=['GET'])
def login():
    return oauth.auth0.authorize_redirect(
        redirect_uri=flask.url_for('callback', _external=True))

#-----------------------------------------------------------------------

@app.route('/logout', methods=['GET'])
def logout():
    flask.session.clear()
    return flask.redirect(
        'https://' + AUTH0_DOMAIN + '/v2/logout?'
        + urllib.parse.urlencode(
            {
                'returnTo': flask.url_for('loggedout', _external=True),
                'client_id': AUTH0_CLIENT_ID,
            },
            quote_via=urllib.parse.quote_plus))

#-----------------------------------------------------------------------
```

PennyAdmin18Auth0/auth.py (Page 2 of 2)

```

@app.route('/loggedout', methods=['GET'])
def loggedout():
    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

# Authenticate the remote user, and return the user's username.
# Do not return unless the user is successfully authenticated.

def authenticate():

    # If the user is in the session, then the user was
    # authenticated previously.  So return the username.
    if 'user' in flask.session:
        return flask.session.get('user')['userinfo']['name']

    flask.abort(flask.redirect('/login'))
```

Auth0 Authentication

- See **PennyAdmin18Auth0** app (cont.)
  - How it works…
  - **???**

Auth0 Authentication

- **Pros**
  - Simple code
  - Application need not manage usernames or passwords
  - Application ***cannot*** access passwords
  - Application passwords can be unique to your application
  - Users can use third-party authentication

Auth0 Authentication

- **Cons**
  - Adds overhead, but mostly only during first visit per browser session
  - Users must remember (yet another) password
  - Users can use third-party authentication
    - Can that be disabled?

Summary

- We have covered:
  - Data comm attacks
  - Third-party authentication (briefly)
    - CAS
    - Google authentication
    - Auth0 authentication

Summary

- We have covered:
  - SQL injection attacks
  - Cross-site scripting (XSS) attacks
  - Authentication & authorization
  - Cookie forgery attacks
  - Cross-site request forgery (CSRF) attacks
  - Data storage attacks
  - Data comm attacks
  - Third-party authentication (briefly)

Appendix 1:
How CAS Works

How CAS Works

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

How CAS Works

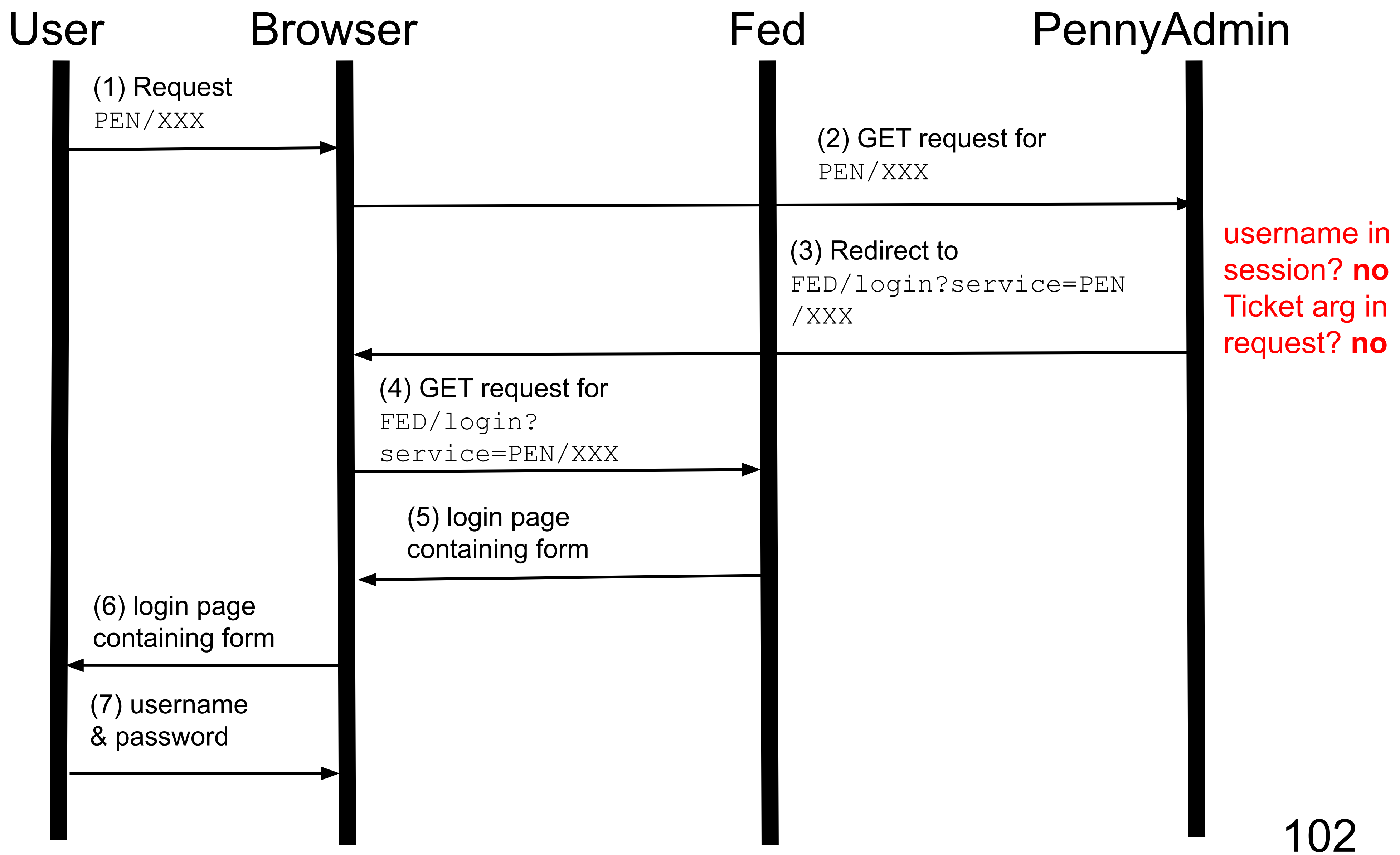
- See **PennyAdmin16Cas** app (cont.)
  - The flow…

Abbreviations:
  FED = https://fed.princeton.edu/cas
  PEN = https://localhost:55555

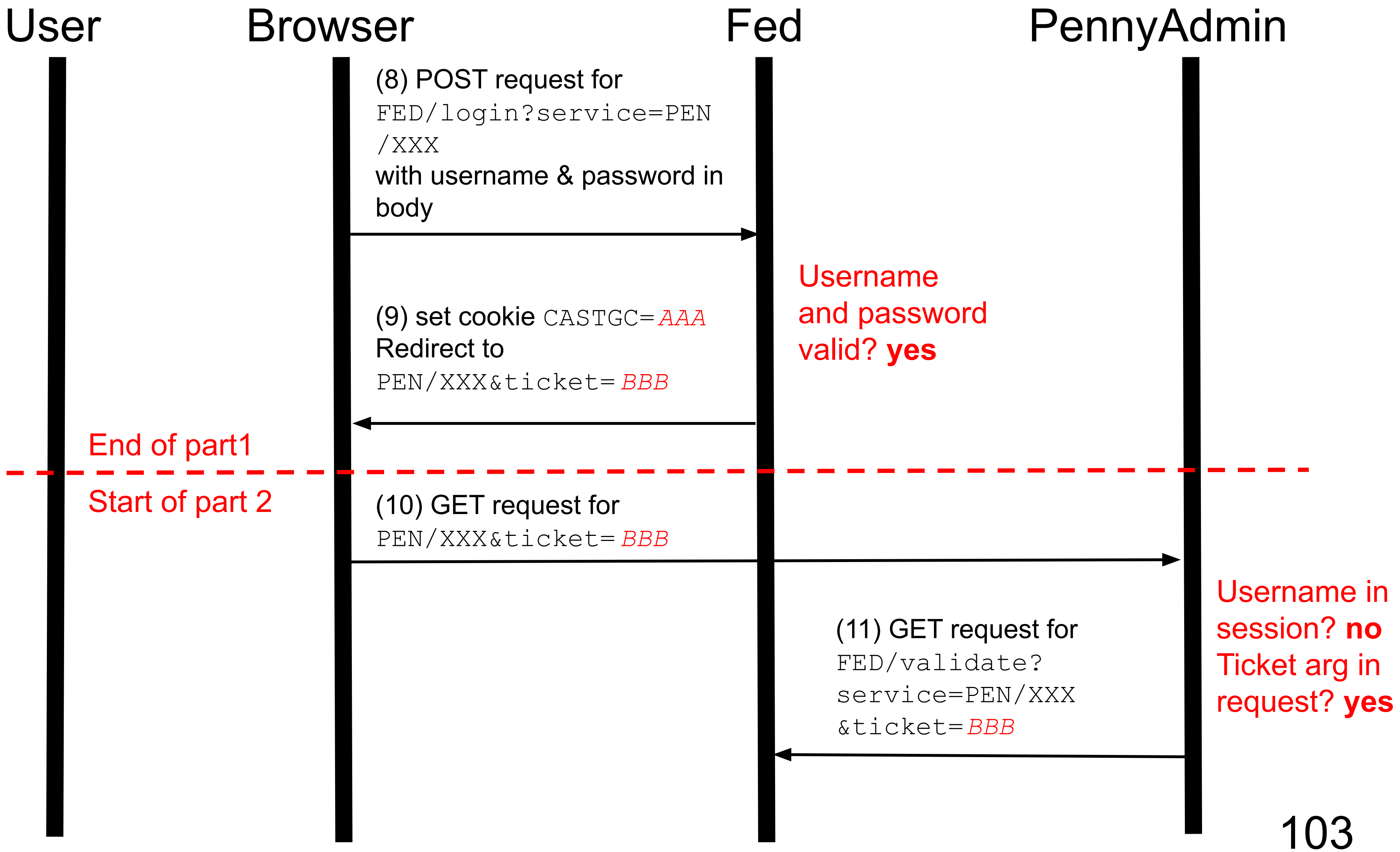
How CAS Works

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

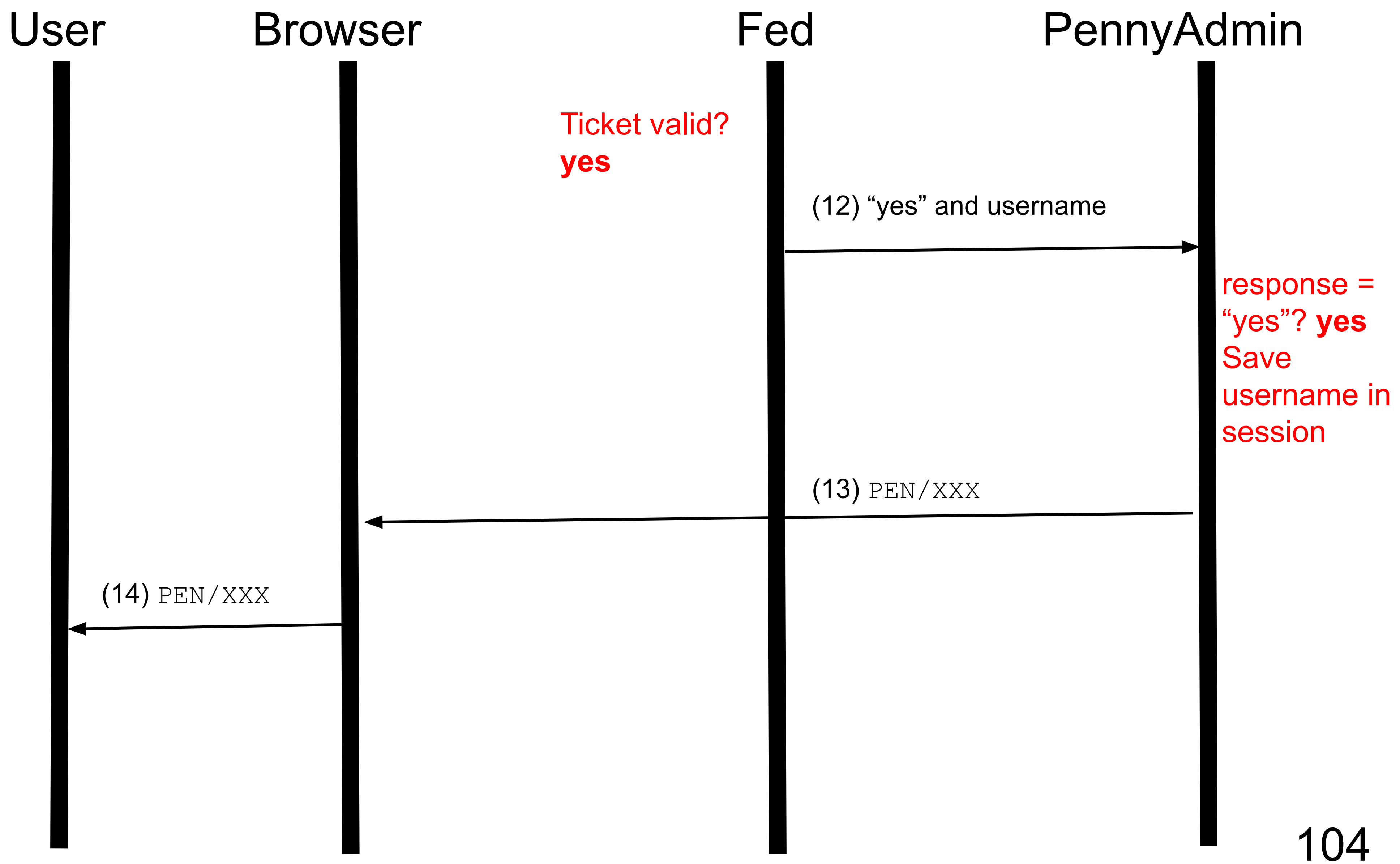
How CAS Works


User
Browser
Fed
PennyAdmin
(1) Request PEN/XXX
(2) GET request for PEN/XXX
(3) Redirect to FED/login?service=PEN/XXX
username in session? **no** Ticket arg in request? **no**
(4) GET request for FED/login?service=PEN/XXX
(5) login page containing form
(6) login page containing form
(7) username & password

How CAS Works


User
Browser
Fed
PennyAdmin
(8) POST request for
FED/login?service=PEN/XXX
with username & password in body
Username and password valid? **yes**
(9) set cookie CASTGC=*AAA*
Redirect to
PEN/XXX&ticket=*BBB*
End of part1
Start of part 2
(10) GET request for
PEN/XXX&ticket=*BBB*
Username in session? **no**
Ticket arg in request? **yes**
(11) GET request for
FED/validate?service=PEN/XXX&ticket=*BBB*

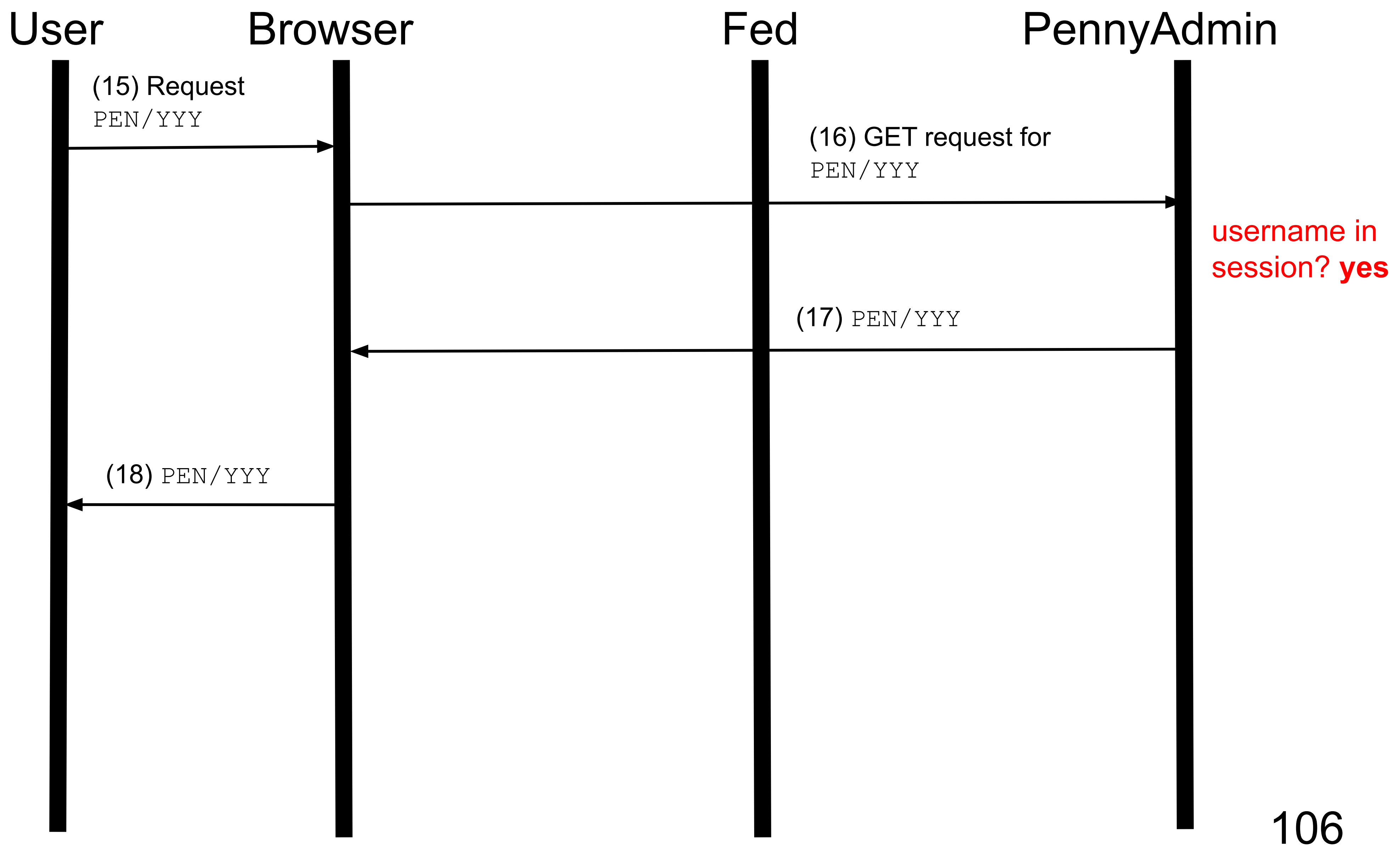
How CAS Works



How CAS Works

Second use of PennyAdmin in same browser session…

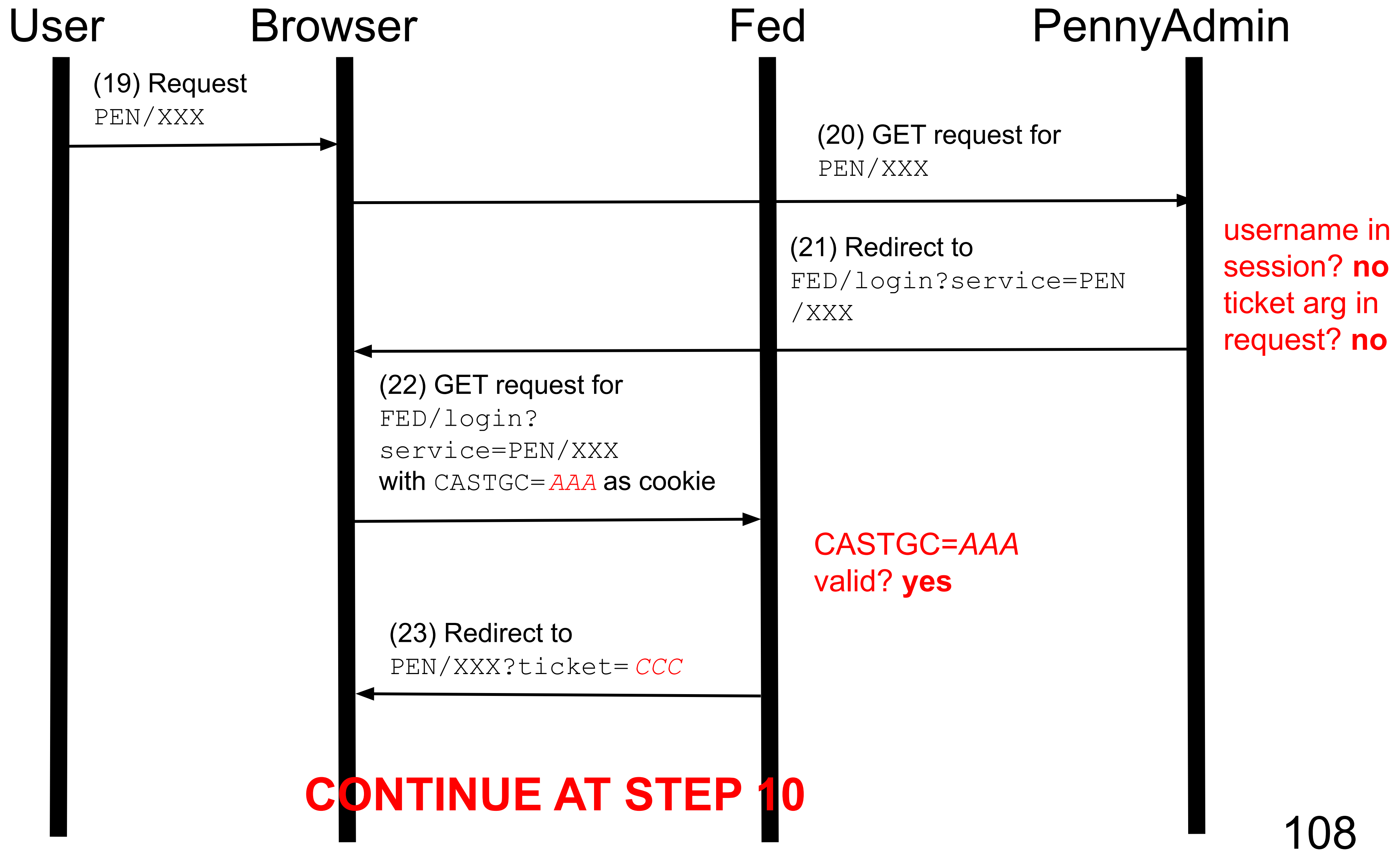
How CAS Works



How CAS Works

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

How CAS Works


User
Browser
Fed
PennyAdmin
(19) Request PEN/XXX
(20) GET request for PEN/XXX
(21) Redirect to FED/login?service=PEN/XXX
username in session? **no** ticket arg in request? **no**
(22) GET request for FED/login?service=PEN/XXX with CASTGC=*AAA* as cookie
CASTGC=*AAA* valid? **yes**
(23) Redirect to PEN/XXX?ticket=*CCC*
CONTINUE AT STEP 10

How CAS Works

- For more information…
- https://apereo.github.io/cas/6.5.x/protocol/CAS-Protocol.html

Appendix 2:
How Google Authentication Works

How Google Auth Works

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

How Google Auth Works

- ***OAuth2***

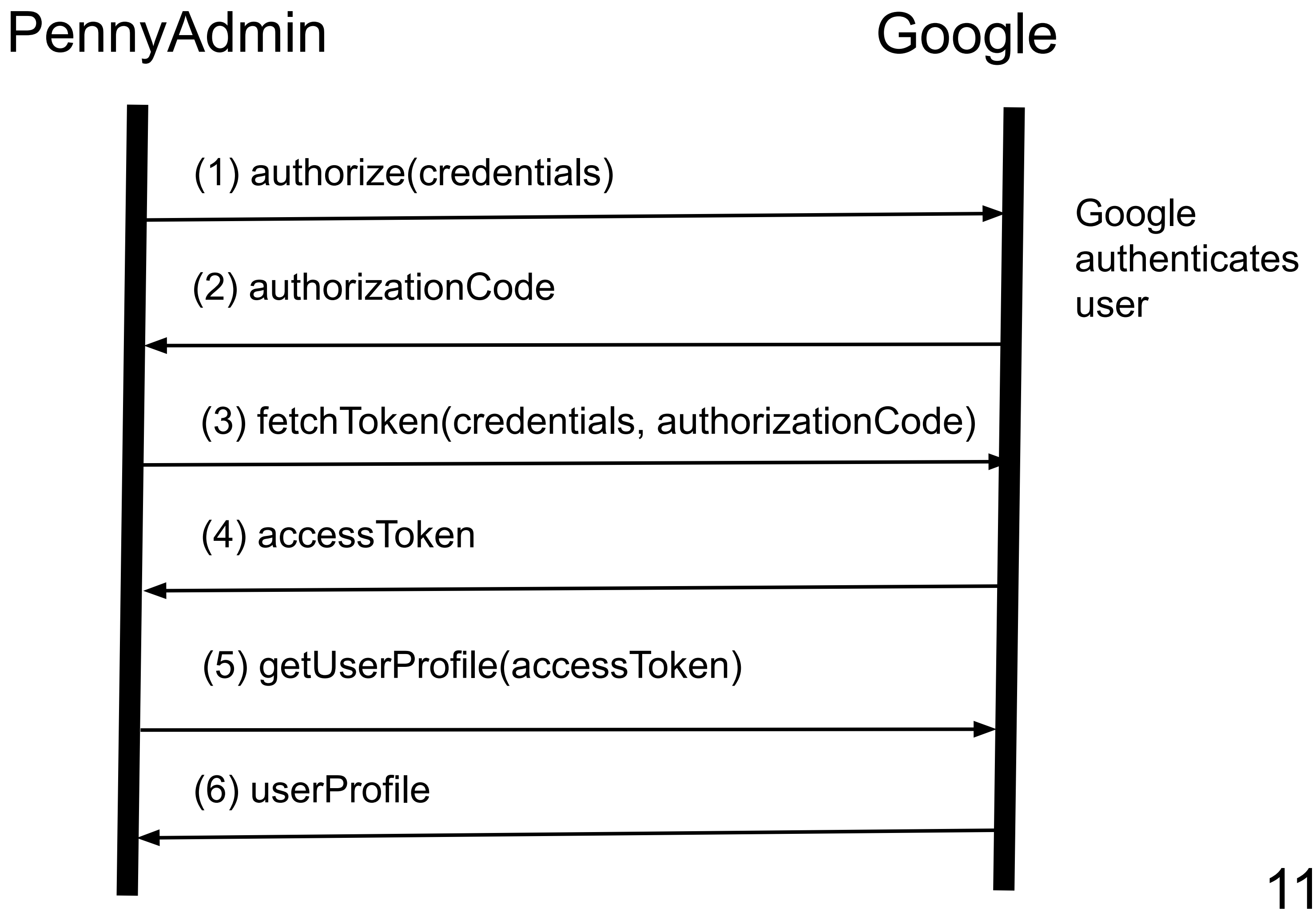
OAuth ("Open Authorization") is an open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information on other websites but without giving them the passwords. This mechanism is used by companies such as Amazon, **Google**, Facebook, 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 Google Auth Works

OAuth2 Flow Overview:

Ahead of time: register PennyAdmin with Google; get credentials


PennyAdmin
Google
(1) authorize(credentials)
Google authenticates user
(2) authorizationCode
(3) fetchToken(credentials, authorizationCode)
(4) accessToken
(5) getUserProfile(accessToken)
(6) userProfile

How Google Auth Works

- See **PennyAdmin17Google** app (cont.)
  - The flow:

How Google Auth Works

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

How Google Auth Works

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

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

(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 Google Auth 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 Google Auth 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/index`

How Google Auth Works

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

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

(20) Browser
Render index page

How Google Auth Works

Second use of PennyAdmin in browser session…

How Google Auth Works

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

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

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

(24) Browser
Render show page

How Google Auth Works

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

How Google Auth Works

(25) User
Type: `https://localhost:5000/index`

(26) Browser
Send GET request: `https://localhost:5000/index`

(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 Google Auth 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