PennyAdmin13Https/penny.py (Page 1 of 3)

```
#!/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)

```
        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)

```
131: @app.route('/handledelete', methods=['POST'])
132: def handle_delete():
133:
134:     username = auth.authenticate()
135:     if not database.is_authorized(username):
136:         html_code = 'You are not authorized to delete books.'
137:         response = flask.make_response(html_code)
138:         return response
139:
140:     isbn = flask.request.form.get('isbn')
141:     if (isbn is None) or (isbn.strip() == ''):
142:         return report_results(username, 'Missing ISBN', '')
143:
144:     isbn = isbn.strip()
145:
146:     database.delete_book(isbn)
147:
148:     message1 = 'The deletion was successful'
149:     message2 = 'The database now does not contain a book with ISBN '
150:     message2 += isbn
151:
152:     return report_results(username, message1, message2)
```

blank (Page 1 of 1)

```
1: This page is intentionally blank.
```

PennyAdmin14Https/top.py (Page 1 of 1)

```
#!/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)

```
#!/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)

```
 66: #-----------------------------------------------------------------------
 67:
 68: @app.route('/handleadd', methods=['POST'])
 69: def handle_add():
 70:
 71:     username = auth.authenticate()
 72:     if not database.is_authorized(username):
 73:         html_code = 'You are not authorized to add books.'
 74:         response = flask.make_response(html_code)
 75:         return response
 76:
 77:     isbn = flask.request.form.get('isbn')
 78:     if (isbn is None) or (isbn.strip() == ''):
 79:         return report_results(username, 'Missing ISBN', '')
 80:
 81:     author = flask.request.form.get('author')
 82:     if (author is None) or (author.strip() == ''):
 83:         return report_results(username, 'Missing author', '')
 84:
 85:     title = flask.request.form.get('title')
 86:     if (title is None) or (title.strip() == ''):
 87:         return report_results(username, 'Missing title', '')
 88:
 89:     isbn = isbn.strip()
 90:     author = author.strip()
 91:     title = title.strip()
 92:
 93:     successful = database.add_book(isbn, author, title)
 94:     if successful:
 95:         message1 = 'The addition was successful'
 96:         message2 = 'The database now contains a book with isbn ' + isbn
 97:         message2 += ' author ' + author + ' and title ' + title
 98:     else:
 99:         message1 = 'The addition was unsuccessful'
100:         message2 = 'A book with ISBN ' + isbn + ' already exists'
101:
102:     return report_results(username, message1, message2)
103:
104: #-----------------------------------------------------------------------
105:
106: @app.route('/delete', methods=['GET'])
107: def delete():
108:
109:     username = auth.authenticate()
110:     if not database.is_authorized(username):
111:         html_code = 'You are not authorized to delete books.'
112:         response = flask.make_response(html_code)
113:         return response
114:
115:     html_code = flask.render_template('delete.html', username=username)
116:
117:     response = flask.make_response(html_code)
118:     return response
119:
120: #-----------------------------------------------------------------------
121:
122: @app.route('/handledelete', methods=['POST'])
123: def handle_delete():
124:
125:     username = auth.authenticate()
126:     if not database.is_authorized(username):
127:         html_code = 'You are not authorized to delete books.'
128:         response = flask.make_response(html_code)
129:         return response
130:
```

PennyAdmin14Https/penny.py (Page 3 of 3)

```
131:     isbn = flask.request.form.get('isbn')
132:     if (isbn is None) or (isbn.strip() == ''):
133:         return report_results(username, 'Missing ISBN', '')
134:
135:     isbn = isbn.strip()
136:
137:     database.delete_book(isbn)
138:
139:     message1 = 'The deletion was successful'
140:     message2 = 'The database now does not contain a book with ISBN '
141:     message2 += isbn
142:
143:     return report_results(username, message1, message2)
```

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.
```

PennyAdmin16Cas/penny.sql (Page 1 of 1)

```
 1: DROP TABLE IF EXISTS books;
 2: CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);
 3: INSERT INTO books (isbn, author, title)
 4:    VALUES ('123', 'Kernighan','The Practice of Programming');
 5: INSERT INTO books (isbn, author, title)
 6:    VALUES ('234', 'Kernighan','The C Programming Language');
 7: INSERT INTO books (isbn, author, title)
 8:    VALUES ('345', 'Sedgewick','Algorithms in C');
 9:
10: DROP TABLE IF EXISTS authorizedusers;
11: CREATE TABLE authorizedusers (username TEXT);
12: INSERT INTO authorizedusers (username) VALUES ('rdondero');
13: INSERT INTO authorizedusers (username) VALUES ('bwk');
```

PennyAdmin16Cas/footer.html (Page 1 of 1)

```
1: <hr>
2: <a href="logoutapp">Log out of the app</a></br>
3: <a href="logoutcas">Log out of the app and end the CAS session</a></br>
4: <hr>
5: Created by <a href="https://www.cs.princeton.edu/~rdondero">
6: Bob Dondero</a>
7: <hr>
```

PennyAdmin16Cas/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://')

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

class Base(sqlalchemy.orm.DeclarativeBase):
    pass

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)

```

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)
```

PennyAdmin16Cas/database.py (Page 3 of 3)

```
131:         print()
132:
133:     print('----------------------------------')
134:     print('Testing delete_book()')
135:     print('----------------------------------')
136:     print()
137:     delete_book('456')
138:     books = get_books()
139:     _write_books(books)
140:     print()
141:     delete_book('456')
142:     books = get_books()
143:     _write_books(books)
144:     print()
145:
146:     print('----------------------------------')
147:     print('Testing is_authorized()')
148:     print('----------------------------------')
149:     print()
150:     print(is_authorized('rdondero'))
151:     print(is_authorized('rdondero2'))
152:
153: if __name__ == '__main__':
154:     _test()
```

PennyAdmin16Cas/penny.py (Page 1 of 3)

```
  1: #!/usr/bin/env python
  2:
  3: #-----------------------------------------------------------------------
  4: # penny.py
  5: # Author: Bob Dondero
  6: #-----------------------------------------------------------------------
  7:
  8: import flask
  9: import database
 10: import auth
 11:
 12: from top import app
 13:
 14: #-----------------------------------------------------------------------
 15:
 16: @app.route('/', methods=['GET'])
 17: @app.route('/index', methods=['GET'])
 18: def index():
 19:
 20:     user_info = auth.authenticate()
 21:     # print(user_info)
 22:     username = user_info['user']
 23:
 24:     is_authorized = database.is_authorized(username)
 25:
 26:     html_code = flask.render_template('index.html', username=username,
 27:         is_authorized=is_authorized)
 28:     response = flask.make_response(html_code)
 29:     return response
 30:
 31: #-----------------------------------------------------------------------
 32:
 33: @app.route('/show', methods=['GET'])
 34: def show():
 35:
 36:     user_info = auth.authenticate()
 37:     # print(user_info)
 38:     username = user_info['user']
 39:
 40:     books = database.get_books()
 41:     html_code = flask.render_template('show.html',
 42:         username=username, books=books)
 43:     response = flask.make_response(html_code)
 44:     return response
 45:
 46: #-----------------------------------------------------------------------
 47:
 48: def report_results(username, message1, message2):
 49:
 50:     html_code = flask.render_template('reportresults.html',
 51:         username=username, message1=message1, message2=message2)
 52:     response = flask.make_response(html_code)
 53:     return response
 54:
 55: #-----------------------------------------------------------------------
 56:
 57: @app.route('/add', methods=['GET'])
 58: def add():
 59:
 60:     user_info = auth.authenticate()
 61:     # print(user_info)
 62:     username = user_info['user']
 63:
 64:     if not database.is_authorized(username):
 65:         html_code = 'You are not authorized to add books.'
```

PennyAdmin16Cas/penny.py (Page 2 of 3)

```
66:         response = flask.make_response(html_code)
67:         return response
68:
69:     html_code = flask.render_template('add.html', username=username)
70:
71:     response = flask.make_response(html_code)
72:     return response
73:
74: #-----------------------------------------------------------------------
75:
76: @app.route('/handleadd', methods=['POST'])
77: def handle_add():
78:
79:     user_info = auth.authenticate()
80:     # print(user_info)
81:     username = user_info['user']
82:
83:     if not database.is_authorized(username):
84:         html_code = 'You are not authorized to add books.'
85:         response = flask.make_response(html_code)
86:         return response
87:
88:     isbn = flask.request.form.get('isbn')
89:     if (isbn is None) or (isbn.strip() == ''):
90:         return report_results(username, 'Missing ISBN', '')
91:
92:     author = flask.request.form.get('author')
93:     if (author is None) or (author.strip() == ''):
94:         return report_results(username, 'Missing author', '')
95:
96:     title = flask.request.form.get('title')
97:     if (title is None) or (title.strip() == ''):
98:         return report_results(username, 'Missing title', '')
99:
100:    isbn = isbn.strip()
101:    author = author.strip()
102:    title = title.strip()
103:
104:    successful = database.add_book(isbn, author, title)
105:    if successful:
106:        message1 = 'The addition was successful'
107:        message2 = 'The database now contains a book with isbn ' + isbn
108:        message2 += ' author ' + author + ' and title ' + title
109:    else:
110:        message1 = 'The addition was unsuccessful'
111:        message2 = 'A book with ISBN ' + isbn + ' already exists'
112:
113:    return report_results(username, message1, message2)
114:
115: #-----------------------------------------------------------------------
116:
117: @app.route('/delete', methods=['GET'])
118: def delete():
119:
120:    user_info = auth.authenticate()
121:    # print(user_info)
122:    username = user_info['user']
123:
124:    if not database.is_authorized(username):
125:        html_code = 'You are not authorized to delete books.'
126:        response = flask.make_response(html_code)
127:        return response
128:
129:    html_code = flask.render_template('delete.html', username=username)
130:
```

PennyAdmin16Cas/penny.py (Page 3 of 3)

```
131:    response = flask.make_response(html_code)
132:    return response
133:
134: #-----------------------------------------------------------------------
135:
136: @app.route('/handledelete', methods=['POST'])
137: def handle_delete():
138:
139:    user_info = auth.authenticate()
140:    # print(user_info)
141:    username = user_info['user']
142:
143:    if not database.is_authorized(username):
144:        html_code = 'You are not authorized to delete books.'
145:        response = flask.make_response(html_code)
146:        return response
147:
148:    isbn = flask.request.form.get('isbn')
149:    if (isbn is None) or (isbn.strip() == ''):
150:        return report_results(username, 'Missing ISBN', '')
151:
152:    isbn = isbn.strip()
153:
154:    database.delete_book(isbn)
155:
156:    message1 = 'The deletion was successful'
157:    message2 = 'The database now does not contain a book with ISBN '
158:    message2 += isbn
159:
160:    return report_results(username, message1, message2)
```

PennyAdmin16Cas/auth.py (Page 1 of 2)

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

#-----------------------------------------------------------------------
# auth.py
# Authors: Alex Halderman, Scott Karlin, Brian Kernighan, Bob Dondero,
#          and Joshua Lau '26
#-----------------------------------------------------------------------

import urllib.request
import urllib.parse
import re
import json
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 user_info; 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)
        + '&format=json')
    with urllib.request.urlopen(val_url) as flo:
        result = json.loads(flo.read().decode('utf-8'))

    if (not result) or ('serviceResponse' not in result):
        return None

    service_response = result['serviceResponse']

    if 'authenticationSuccess' in service_response:
        user_info = service_response['authenticationSuccess']
        return user_info

    if 'authenticationFailure' in service_response:
        print('CAS authentication failure:', service_response)
        return None

    print('Unexpected CAS response:', service_response)
    return None

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

```

PennyAdmin16Cas/auth.py (Page 2 of 2)

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

def authenticate():

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

    # 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')
    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.
    user_info = validate(ticket)
    if user_info 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 user_info in
    # the session and return the username.
    flask.session['user_info'] = user_info
    return user_info

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

@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))
```

PennyAdmin17Google/runserver.py (Page 1 of 1)

```
#!/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://')

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

class Base(sqlalchemy.orm.DeclarativeBase):
    pass

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)
```

PennyAdmin17Google/database.py (Page 3 of 3)

```
131:         print()
132:
133:     print('----------------------------------')
134:     print('Testing delete_book()')
135:     print('----------------------------------')
136:     print()
137:     delete_book('456')
138:     books = get_books()
139:     _write_books(books)
140:     print()
141:     delete_book('456')
142:     books = get_books()
143:     _write_books(books)
144:     print()
145:
146:     print('----------------------------------')
147:     print('Testing is_authorized()')
148:     print('----------------------------------')
149:     print()
150:     print(is_authorized('rdondero'))
151:     print(is_authorized('rdondero2'))
152:
153: if __name__ == '__main__':
154:     _test()
```

PennyAdmin17Google/penny.py (Page 1 of 3)

```
 1: #!/usr/bin/env python
 2:
 3: #-----------------------------------------------------------------------
 4: # penny.py
 5: # Author: Bob Dondero
 6: #-----------------------------------------------------------------------
 7:
 8: import flask
 9: import database
10: import auth
11:
12: from top import app
13:
14: #-----------------------------------------------------------------------
15:
16: @app.route('/', methods=['GET'])
17: @app.route('/index', methods=['GET'])
18: def index():
19:
20:     user_info = auth.authenticate()
21:     # print(user_info)
22:     username = user_info['email']
23:
24:     is_authorized = database.is_authorized(username)
25:
26:     html_code = flask.render_template('index.html', username=username,
27:         is_authorized=is_authorized)
28:     response = flask.make_response(html_code)
29:     return response
30:
31: #-----------------------------------------------------------------------
32:
33: @app.route('/show', methods=['GET'])
34: def show():
35:
36:     user_info = auth.authenticate()
37:     # print(user_info)
38:     username = user_info['email']
39:
40:     books = database.get_books()
41:     html_code = flask.render_template('show.html',
42:         username=username, books=books)
43:     response = flask.make_response(html_code)
44:     return response
45:
46: #-----------------------------------------------------------------------
47:
48: def report_results(username, message1, message2):
49:
50:     html_code = flask.render_template('reportresults.html',
51:         username=username, message1=message1, message2=message2)
52:     response = flask.make_response(html_code)
53:     return response
54:
55: #-----------------------------------------------------------------------
56:
57: @app.route('/add', methods=['GET'])
58: def add():
59:
60:     user_info = auth.authenticate()
61:     # print(user_info)
62:     username = user_info['email']
63:
64:     if not database.is_authorized(username):
65:         html_code = 'You are not authorized to add books.'
```

PennyAdmin17Google/penny.py (Page 2 of 3)

```
66:         response = flask.make_response(html_code)
67:         return response
68:
69:     html_code = flask.render_template('add.html', username=username)
70:
71:     response = flask.make_response(html_code)
72:     return response
73:
74: #-----------------------------------------------------------------------
75:
76: @app.route('/handleadd', methods=['POST'])
77: def handle_add():
78:
79:     user_info = auth.authenticate()
80:     # print(user_info)
81:     username = user_info['email']
82:
83:     if not database.is_authorized(username):
84:         html_code = 'You are not authorized to add books.'
85:         response = flask.make_response(html_code)
86:         return response
87:
88:     isbn = flask.request.form.get('isbn')
89:     if (isbn is None) or (isbn.strip() == ''):
90:         return report_results(username, 'Missing ISBN', '')
91:
92:     author = flask.request.form.get('author')
93:     if (author is None) or (author.strip() == ''):
94:         return report_results(username, 'Missing author', '')
95:
96:     title = flask.request.form.get('title')
97:     if (title is None) or (title.strip() == ''):
98:         return report_results(username, 'Missing title', '')
99:
100:    isbn = isbn.strip()
101:    author = author.strip()
102:    title = title.strip()
103:
104:    successful = database.add_book(isbn, author, title)
105:    if successful:
106:        message1 = 'The addition was successful'
107:        message2 = 'The database now contains a book with isbn ' + isbn
108:        message2 += ' author ' + author + ' and title ' + title
109:    else:
110:        message1 = 'The addition was unsuccessful'
111:        message2 = 'A book with ISBN ' + isbn + ' already exists'
112:
113:    return report_results(username, message1, message2)
114:
115: #-----------------------------------------------------------------------
116:
117: @app.route('/delete', methods=['GET'])
118: def delete():
119:
120:    user_info = auth.authenticate()
121:    # print(user_info)
122:    username = user_info['email']
123:
124:    if not database.is_authorized(username):
125:        html_code = 'You are not authorized to delete books.'
126:        response = flask.make_response(html_code)
127:        return response
128:
129:    html_code = flask.render_template('delete.html', username=username)
130:
```

PennyAdmin17Google/penny.py (Page 3 of 3)

```
131:    response = flask.make_response(html_code)
132:    return response
133:
134: #-----------------------------------------------------------------------
135:
136: @app.route('/handledelete', methods=['POST'])
137: def handle_delete():
138:
139:    user_info = auth.authenticate()
140:    # print(user_info)
141:    username = user_info['email']
142:
143:    if not database.is_authorized(username):
144:        html_code = 'You are not authorized to delete books.'
145:        response = flask.make_response(html_code)
146:        return response
147:
148:    isbn = flask.request.form.get('isbn')
149:    if (isbn is None) or (isbn.strip() == ''):
150:        return report_results(username, 'Missing ISBN', '')
151:
152:    isbn = isbn.strip()
153:
154:    database.delete_book(isbn)
155:
156:    message1 = 'The deletion was successful'
157:    message2 = 'The database now does not contain a book with ISBN '
158:    message2 += isbn
159:
160:    return report_results(username, message1, message2)
```

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:
```

PennyAdmin17Google/auth.py (Page 2 of 3)

```
    # print('code:', code, file=sys.stderr)
    #-------------------------------------------------------------------

    # Determine the URL to fetch tokens that allow the application to
    # ask for the user's profile data.
    google_provider_cfg = requests.get(
        GOOGLE_DISCOVERY_URL, timeout=2).json()
    token_endpoint = google_provider_cfg['token_endpoint']

    # Construct a request to fetch the tokens.
    token_url, headers, body = client.prepare_token_request(
        token_endpoint,
        authorization_response=flask.request.url,
        redirect_url=flask.request.base_url,
        code=code
    )

    #-------------------------------------------------------------------
    # For learning:
    # print('token_url:', token_url, file=sys.stderr)
    # print('headers:', headers, file=sys.stderr)
    # print('body:', body, file=sys.stderr)
    #-------------------------------------------------------------------

    # Fetch the tokens.
    token_response = requests.post(
        token_url,
        headers=headers,
        data=body,
        auth=(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET),
        timeout=2
    )

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

    # Parse the tokens.
    client.parse_request_body_response(
        json.dumps(token_response.json()))

    # Using the tokens, fetch the user's profile data,
    # including the user's Google profile image and email address.
    userinfo_endpoint = google_provider_cfg['userinfo_endpoint']
    uri, headers, body = client.add_token(userinfo_endpoint)

    #-------------------------------------------------------------------
    # For learning:
    # print('uri:', uri, file=sys.stderr)
    # print('headers:', headers, file=sys.stderr)
    # print('body:', body, file=sys.stderr)
    #-------------------------------------------------------------------

    userinfo_response = requests.get(uri, headers=headers, data=body,
        timeout=2)

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

    # Optional: Make sure the user's email address is verified.
```

PennyAdmin17Google/auth.py (Page 3 of 3)

```
131:     if not userinfo_response.json().get('email_verified'):
132:         message = 'User email not available or not verified by Google.'
133:         return message, 400
134:
135:     # Save the user profile data in the session.
136:     flask.session['user_info'] = userinfo_response.json()
137:
138:     return flask.redirect(flask.url_for('index'))
139:
140: #-----------------------------------------------------------------------
141:
142: @app.route('/logoutapp', methods=['GET'])
143: def logoutapp():
144:
145:     # Log out of the application.
146:     flask.session.clear()
147:     html_code = flask.render_template('loggedout.html')
148:     response = flask.make_response(html_code)
149:     return response
150:
151: #-----------------------------------------------------------------------
152:
153: @app.route('/logoutgoogle', methods=['GET'])
154: def logoutgoogle():
155:
156:     # Log out of the application.
157:     flask.session.clear()
158:
159:     # Log out of Google.
160:     flask.abort(flask.redirect(
161:         'https://mail.google.com/mail/u/0/?logout&hl=en'))
162:
163: #-----------------------------------------------------------------------
164:
165: def authenticate():
166:
167:     if 'user_info' not in flask.session:
168:         flask.abort(flask.redirect(flask.url_for('login')))
169:
170:     return flask.session.get('user_info')
```