

## PennyAdmin13Https/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.before_request
17: def before_request():
18:     if (not app.debug) and (not flask.request.is_secure):
19:         url = flask.request.url.replace('http://', 'https://', 1)
20:         return flask.redirect(url, code=301)
21:     return None
22:
23: #-----
24:
25: @app.route('/', methods=['GET'])
26: @app.route('/index', methods=['GET'])
27: def index():
28:
29:     username = auth.authenticate()
30:     is_authorized = database.is_authorized(username)
31:
32:     html_code = flask.render_template('index.html', username=username,
33:                                     is_authorized=is_authorized)
34:     response = flask.make_response(html_code)
35:     return response
36:
37: #-----
38:
39: @app.route('/show', methods=['GET'])
40: def show():
41:
42:     username = auth.authenticate()
43:
44:     books = database.get_books()
45:     html_code = flask.render_template('show.html',
46:                                     username=username, books=books)
47:     response = flask.make_response(html_code)
48:     return response
49:
50: #-----
51:
52: def report_results(username, message1, message2):
53:
54:     html_code = flask.render_template('reportresults.html',
55:                                     username=username, message1=message1, message2=message2)
56:     response = flask.make_response(html_code)
57:     return response
58:
59: #-----
60:
61: @app.route('/add', methods=['GET'])
62: def add():
63:
64:     username = auth.authenticate()
65:     if not database.is_authorized(username):

```

## PennyAdmin13Https/penny.py (Page 2 of 3)

```

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

```

**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)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # top.py
5: # Author: Bob Dondero
6: #-----
7:
8: import os
9: import flask
10: import flask_wtf.csrf
11: import flask_talisman
12: import dotenv
13:
14: app = flask.Flask('penny', template_folder='.')
15:
16: dotenv.load_dotenv()
17: app.secret_key = os.environ['APP_SECRET_KEY']
18:
19: flask_wtf.csrf.CSRFProtect(app)
20:
21: flask_talisman.Talisman(app)

```

## PennyAdmin14Https/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:     username = auth.authenticate()
21:     is_authorized = database.is_authorized(username)
22:
23:     html_code = flask.render_template('index.html', username=username,
24:                                     is_authorized=is_authorized)
25:     response = flask.make_response(html_code)
26:     return response
27:
28: #-----
29:
30: @app.route('/show', methods=['GET'])
31: def show():
32:
33:     username = auth.authenticate()
34:
35:     books = database.get_books()
36:     html_code = flask.render_template('show.html',
37:                                     username=username, books=books)
38:     response = flask.make_response(html_code)
39:     return response
40:
41: #-----
42:
43: def report_results(username, message1, message2):
44:
45:     html_code = flask.render_template('reportresults.html',
46:                                     username=username, message1=message1, message2=message2)
47:     response = flask.make_response(html_code)
48:     return response
49:
50: #-----
51:
52: @app.route('/add', methods=['GET'])
53: def add():
54:
55:     username = auth.authenticate()
56:     if not database.is_authorized(username):
57:         html_code = 'You are not authorized to add books.'
58:         response = flask.make_response(html_code)
59:         return response
60:
61:     html_code = flask.render_template('add.html', username=username)
62:
63:     response = flask.make_response(html_code)
64:     return response
65:

```

## 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)

```
1: #!/usr/bin/env python
2:
3: #-----
4: # runserver.py
5: # Author: Bob Dondero
6: #-----
7:
8: import sys
9: import penny
10:
11: def main():
12:
13:     if len(sys.argv) != 2:
14:         print('Usage: ' + sys.argv[0] + ' port', file=sys.stderr)
15:         sys.exit(1)
16:
17:     try:
18:         port = int(sys.argv[1])
19:     except Exception:
20:         print('Port must be an integer.', file=sys.stderr)
21:         sys.exit(1)
22:
23:     try:
24:         penny.app.run(host='0.0.0.0', port=port,
25:                       ssl_context=('cert.pem', 'key.pem'))
26:     except Exception as ex:
27:         print(ex, file=sys.stderr)
28:         sys.exit(1)
29:
30: if __name__ == '__main__':
31:     main()
```

## blank (Page 1 of 1)

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 ('rdontero');
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/~rdontero">
6: Bob Dondero</a>
7: <hr>
```

## PennyAdmin16Cas/database.py (Page 1 of 3)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # database.py
5: # Author: Bob Dondero
6: #-----
7:
8: import os
9: import sqlalchemy
10: import sqlalchemy.orm
11: import dotenv
12:
13: #-----
14:
15: dotenv.load_dotenv()
16: _database_url = os.getenv('DATABASE_URL', 'sqlite:///penny.sqlite')
17: _database_url = _database_url.replace('postgres://', 'postgresql://')
18:
19: #-----
20:
21: Base = sqlalchemy.orm.declarative_base()
22:
23: class Book (Base):
24:     __tablename__ = 'books'
25:     isbn = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
26:     author = sqlalchemy.Column(sqlalchemy.String)
27:     title = sqlalchemy.Column(sqlalchemy.String)
28:
29: class AuthorizedUser (Base):
30:     __tablename__ = 'authorizedusers'
31:     username = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
32:
33: _engine = sqlalchemy.create_engine(_database_url)
34:
35: #-----
36:
37: def get_books():
38:
39:     books = []
40:
41:     with sqlalchemy.orm.Session(_engine) as session:
42:         query = session.query(Book)
43:         table = query.all()
44:         for row in table:
45:             book = {'isbn': row.isbn, 'author': row.author,
46:                   'title': row.title}
47:             books.append(book)
48:
49:     return books
50:
51: #-----
52:
53: def add_book(isbn, author, title):
54:
55:     with sqlalchemy.orm.Session(_engine) as session:
56:         row = Book(isbn=isbn, author=author, title=title)
57:         session.add(row)
58:         try:
59:             session.commit()
60:             return True
61:         except sqlalchemy.exc.IntegrityError:
62:             return False
63:
64: #-----
65:

```

## PennyAdmin16Cas/database.py (Page 2 of 3)

```

66: def delete_book(isbn):
67:
68:     with sqlalchemy.orm.Session(_engine) as session:
69:         session.query(Book).filter(Book.isbn==isbn).delete()
70:         session.commit()
71:
72: #-----
73:
74: def is_authorized(username):
75:
76:     with sqlalchemy.orm.Session(_engine) as session:
77:         query = session.query(AuthorizedUser) \
78:             .filter(AuthorizedUser.username==username)
79:         try:
80:             query.one()
81:             return True
82:         except sqlalchemy.exc.NoResultFound:
83:             return False
84:
85: #-----
86:
87: # For testing:
88:
89: def _write_books(books):
90:     for book in books:
91:         print('%s | %s | %s' % (book['isbn'], book['author'],
92:                               book['title']))
93:
94: def _test():
95:     print('-----')
96:     print('Testing get_books()')
97:     print('-----')
98:     print()
99:     books = get_books()
100:    _write_books(books)
101:    print()
102:
103:    print('-----')
104:    print('Testing add_book()')
105:    print('-----')
106:    print()
107:    successful = add_book('456', 'Kernighan', 'New Book')
108:    if successful:
109:        print('Add was successful')
110:        print()
111:        books = get_books()
112:        _write_books(books)
113:        print()
114:    else:
115:        print('Add was unsuccessful')
116:        print()
117:        _write_books(books)
118:        print()
119:    successful = add_book('456', 'Kernighan', 'New Book')
120:    if successful:
121:        print('Add was successful')
122:        print()
123:        books = get_books()
124:        _write_books(books)
125:        print()
126:    else:
127:        print('Add was unsuccessful')
128:        print()
129:        _write_books(books)
130:        print()

```

## PennyAdmin16Cas/database.py (Page 3 of 3)

```
131:
132:     print('-----')
133:     print('Testing delete_book()')
134:     print('-----')
135:     print()
136:     delete_book('456')
137:     books = get_books()
138:     _write_books(books)
139:     print()
140:     delete_book('456')
141:     books = get_books()
142:     _write_books(books)
143:     print()
144:
145:     print('-----')
146:     print('Testing is_authorized()')
147:     print('-----')
148:     print()
149:     print(is_authorized('rdontero'))
150:     print(is_authorized('rdontero2'))
151:
152: if __name__ == '__main__':
153:     _test()
```

## blank (Page 1 of 1)

1: This page is intentionally blank.



## PennyAdmin16Cas/auth.py (Page 1 of 2)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # auth.py
5: # Authors: Alex Halderman, Scott Karlin, Brian Kernighan, Bob Dondero
6: #-----
7:
8: import urllib.request
9: import urllib.parse
10: import re
11: import flask
12:
13: from top import app
14:
15: #-----
16:
17: _CAS_URL = 'https://fed.princeton.edu/cas/'
18:
19: #-----
20:
21: # Return url after stripping out the "ticket" parameter that was
22: # added by the CAS server.
23:
24: def strip_ticket(url):
25:     if url is None:
26:         return "something is badly wrong"
27:     url = re.sub(r'ticket=[^&]*&?', '', url)
28:     url = re.sub(r'\?&?&|&$', '', url)
29:     return url
30:
31: #-----
32:
33: # Validate a login ticket by contacting the CAS server. If
34: # valid, return the user's username; otherwise, return None.
35:
36: def validate(ticket):
37:     val_url = (_CAS_URL + "validate" + '?service='
38:               + urllib.parse.quote(strip_ticket(flask.request.url))
39:               + '&ticket=' + urllib.parse.quote(ticket))
40:     lines = []
41:     with urllib.request.urlopen(val_url) as flo:
42:         lines = flo.readlines() # Should return 2 lines.
43:     if len(lines) != 2:
44:         return None
45:     first_line = lines[0].decode('utf-8')
46:     second_line = lines[1].decode('utf-8')
47:     if not first_line.startswith('yes'):
48:         return None
49:     return second_line
50:
51: #-----
52:
53: # Authenticate the remote user, and return the user's username.
54: # Do not return unless the user is successfully authenticated.
55:
56: def authenticate():
57:
58:     # If the username is in the session, then the user was
59:     # authenticated previously. So return the username.
60:     if 'username' in flask.session:
61:         return flask.session.get('username')
62:
63:     # If the request does not contain a login ticket, then redirect
64:     # the browser to the login page to get one.
65:     ticket = flask.request.args.get('ticket')

```

## PennyAdmin16Cas/auth.py (Page 2 of 2)

```

66:     if ticket is None:
67:         login_url = (_CAS_URL + 'login?service=' +
68:                     urllib.parse.quote(flask.request.url))
69:         flask.abort(flask.redirect(login_url))
70:
71:     # If the login ticket is invalid, then redirect the browser
72:     # to the login page to get a new one.
73:     username = validate(ticket)
74:     if username is None:
75:         login_url = (_CAS_URL + 'login?service='
76:                     + urllib.parse.quote(strip_ticket(flask.request.url)))
77:         flask.abort(flask.redirect(login_url))
78:
79:     # The user is authenticated, so store the username in
80:     # the session.
81:     username = username.strip().lower()
82:     flask.session['username'] = username
83:     return username
84:
85: #-----
86:
87: @app.route('/logoutapp', methods=['GET'])
88: def logoutapp():
89:
90:     # Log out of the application.
91:     flask.session.clear()
92:     html_code = flask.render_template('loggedout.html')
93:     response = flask.make_response(html_code)
94:     return response
95:
96: #-----
97:
98: @app.route('/logoutcas', methods=['GET'])
99: def logoutcas():
100:
101:     # Log out of the CAS session, and then the application.
102:     logout_url = (_CAS_URL + 'logout?service='
103:                  + urllib.parse.quote(
104:                      re.sub('logoutcas', 'logoutapp', flask.request.url)))
105:     flask.abort(flask.redirect(logout_url))

```

## PennyAdmin17Google/runserver.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
3: #-----
4: # runserver.py
5: # Author: Bob Dondero
6: #-----
7:
8: import sys
9: import penny
10:
11: # Google expects the application to run on port 5000.
12: PORT = 5000
13:
14: def main():
15:
16:     if len(sys.argv) != 1:
17:         print('Usage: ' + sys.argv[0], file=sys.stderr)
18:         sys.exit(1)
19:
20:     try:
21:         penny.app.run(host='0.0.0.0', port=PORT, debug=True,
22:                       ssl_context=('cert.pem', 'key.pem'))
23:     except Exception as ex:
24:         print(ex, file=sys.stderr)
25:         sys.exit(1)
26:
27: if __name__ == '__main__':
28:     main()
```

## blank (Page 1 of 1)

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

## PennyAdmin17Google/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)
13:   VALUES ('donderorbert@gmail.com');
14: INSERT INTO authorizedusers (username)
15:   VALUES ('bwk@gmail.com');
```

## PennyAdmin17Google/footer.html (Page 1 of 1)

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

## PennyAdmin17Google/database.py (Page 1 of 3)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # database.py
5: # Author: Bob Dondero
6: #-----
7:
8: import os
9: import sqlalchemy
10: import sqlalchemy.orm
11: import dotenv
12:
13: #-----
14:
15: dotenv.load_dotenv()
16: _database_url = os.getenv('DATABASE_URL', 'sqlite:///penny.sqlite')
17: _database_url = _database_url.replace('postgres://', 'postgresql://')
18:
19: #-----
20:
21: Base = sqlalchemy.orm.declarative_base()
22:
23: class Book (Base):
24:     __tablename__ = 'books'
25:     isbn = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
26:     author = sqlalchemy.Column(sqlalchemy.String)
27:     title = sqlalchemy.Column(sqlalchemy.String)
28:
29: class AuthorizedUser (Base):
30:     __tablename__ = 'authorizedusers'
31:     username = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
32:
33: _engine = sqlalchemy.create_engine(_database_url)
34:
35: #-----
36:
37: def get_books():
38:
39:     books = []
40:
41:     with sqlalchemy.orm.Session(_engine) as session:
42:         query = session.query(Book)
43:         table = query.all()
44:         for row in table:
45:             book = {'isbn': row.isbn, 'author': row.author,
46:                   'title': row.title}
47:             books.append(book)
48:
49:     return books
50:
51: #-----
52:
53: def add_book(isbn, author, title):
54:
55:     with sqlalchemy.orm.Session(_engine) as session:
56:         row = Book(isbn=isbn, author=author, title=title)
57:         session.add(row)
58:         try:
59:             session.commit()
60:             return True
61:         except sqlalchemy.exc.IntegrityError:
62:             return False
63:
64: #-----
65:

```

## PennyAdmin17Google/database.py (Page 2 of 3)

```

66: def delete_book(isbn):
67:
68:     with sqlalchemy.orm.Session(_engine) as session:
69:         session.query(Book).filter(Book.isbn==isbn).delete()
70:         session.commit()
71:
72: #-----
73:
74: def is_authorized(username):
75:
76:     with sqlalchemy.orm.Session(_engine) as session:
77:         query = session.query(AuthorizedUser) \
78:             .filter(AuthorizedUser.username==username)
79:         try:
80:             query.one()
81:             return True
82:         except sqlalchemy.exc.NoResultFound:
83:             return False
84:
85: #-----
86:
87: # For testing:
88:
89: def _write_books(books):
90:     for book in books:
91:         print('%s | %s | %s' % (book['isbn'], book['author'],
92:                               book['title']))
93:
94: def _test():
95:     print('-----')
96:     print('Testing get_books()')
97:     print('-----')
98:     print()
99:     books = get_books()
100:    _write_books(books)
101:    print()
102:
103:    print('-----')
104:    print('Testing add_book()')
105:    print('-----')
106:    print()
107:    successful = add_book('456', 'Kernighan', 'New Book')
108:    if successful:
109:        print('Add was successful')
110:        print()
111:        books = get_books()
112:        _write_books(books)
113:        print()
114:    else:
115:        print('Add was unsuccessful')
116:        print()
117:        _write_books(books)
118:        print()
119:    successful = add_book('456', 'Kernighan', 'New Book')
120:    if successful:
121:        print('Add was successful')
122:        print()
123:        books = get_books()
124:        _write_books(books)
125:        print()
126:    else:
127:        print('Add was unsuccessful')
128:        print()
129:        _write_books(books)
130:        print()

```

## PennyAdmin17Google/database.py (Page 3 of 3)

```

131:
132:     print('-----')
133:     print('Testing delete_book()')
134:     print('-----')
135:     print()
136:     delete_book('456')
137:     books = get_books()
138:     _write_books(books)
139:     print()
140:     delete_book('456')
141:     books = get_books()
142:     _write_books(books)
143:     print()
144:
145:     print('-----')
146:     print('Testing is_authorized()')
147:     print('-----')
148:     print()
149:     print(is_authorized('rdontero'))
150:     print(is_authorized('rdontero2'))
151:
152: if __name__ == '__main__':
153:     _test()

```

## PennyAdmin17Google/auth.py (Page 1 of 3)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # auth.py
5: # Author: Bob Dondero
6: # With lots of help from https://realpython.com/flask-google-login/
7: #-----
8:
9: import os
10: import json
11: import requests
12: import flask
13: import oauthlib.oauth2
14: import dotenv
15:
16: from top import app
17:
18: #-----
19:
20: GOOGLE_DISCOVERY_URL = (
21:     'https://accounts.google.com/.well-known/openid-configuration')
22:
23: dotenv.load_dotenv()
24: GOOGLE_CLIENT_ID = os.environ['GOOGLE_CLIENT_ID']
25: GOOGLE_CLIENT_SECRET = os.environ['GOOGLE_CLIENT_SECRET']
26:
27: client = oauthlib.oauth2.WebApplicationClient(GOOGLE_CLIENT_ID)
28:
29: #-----
30:
31: @app.route('/login', methods=['GET'])
32: def login():
33:
34:     # Determine the URL for Google login.
35:     google_provider_cfg = requests.get(
36:         GOOGLE_DISCOVERY_URL, timeout=2).json()
37:     authorization_endpoint = (
38:         google_provider_cfg['authorization_endpoint'])
39:
40:     # Construct the request URL for Google login, providing scopes
41:     # to fetch the user's profile data.
42:     request_uri = client.prepare_request_uri(
43:         authorization_endpoint,
44:         redirect_uri = flask.request.base_url + '/callback',
45:         scope=['openid', 'email', 'profile'],
46:     )
47:
48:     #-----
49:     # For learning:
50:     # print('request_uri:', request_uri, file=sys.stderr)
51:     #-----
52:
53:     # Redirect to the request URL.
54:     return flask.redirect(request_uri)
55:
56: #-----
57:
58: @app.route('/login/callback', methods=['GET'])
59: def callback():
60:
61:     # Get the authorization code that Google sent.
62:     code = flask.request.args.get('code')
63:
64:     #-----
65:     # For learning:

```

## PennyAdmin17Google/auth.py (Page 2 of 3)

```

66:     # print('code:', code, file=sys.stderr)
67:     #-----
68:
69:     # Determine the URL to fetch tokens that allow the application to
70:     # ask for the user's profile data.
71:     google_provider_cfg = requests.get(
72:         GOOGLE_DISCOVERY_URL, timeout=2).json()
73:     token_endpoint = google_provider_cfg['token_endpoint']
74:
75:     # Construct a request to fetch the tokens.
76:     token_url, headers, body = client.prepare_token_request(
77:         token_endpoint,
78:         authorization_response=flask.request.url,
79:         redirect_url=flask.request.base_url,
80:         code=code
81:     )
82:
83:     #-----
84:     # For learning:
85:     # print('token_url:', token_url, file=sys.stderr)
86:     # print('headers:', headers, file=sys.stderr)
87:     # print('body:', body, file=sys.stderr)
88:     #-----
89:
90:     # Fetch the tokens.
91:     token_response = requests.post(
92:         token_url,
93:         headers=headers,
94:         data=body,
95:         auth=(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET),
96:         timeout=2
97:     )
98:
99:     #-----
100:    # For learning:
101:    # print('token_response.json():', token_response.json(),
102:         #   file=sys.stderr)
103:    #-----
104:
105:    # Parse the tokens.
106:    client.parse_request_body_response(
107:        json.dumps(token_response.json()))
108:
109:    # Using the tokens, fetch the user's profile data,
110:    # including the user's Google profile image and email address.
111:    userinfo_endpoint = google_provider_cfg['userinfo_endpoint']
112:    uri, headers, body = client.add_token(userinfo_endpoint)
113:
114:    #-----
115:    # For learning:
116:    # print('uri:', uri, file=sys.stderr)
117:    # print('headers:', headers, file=sys.stderr)
118:    # print('body:', body, file=sys.stderr)
119:    #-----
120:
121:    userinfo_response = requests.get(uri, headers=headers, data=body,
122:        timeout=2)
123:
124:    #-----
125:    # For learning:
126:    # print('userinfo_response.json():', userinfo_response.json(),
127:         #   file=sys.stderr)
128:    #-----
129:
130:    # 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:
137:     flask.session['email'] = userinfo_response.json()['email']
138:     #flask.session['sub'] = userinfo_response.json()['sub']
139:     #flask.session['name'] = userinfo_response.json()['name']
140:     #flask.session['given_name'] = (
141:         userinfo_response.json()['given_name'])
142:     #flask.session['family_name'] = (
143:         userinfo_response.json()['family_name'])
144:     #flask.session['picture'] = userinfo_response.json()['picture']
145:     #flask.session['email_verified'] = (
146:         userinfo_response.json()['email_verified'])
147:     #flask.session['locale'] = userinfo_response.json()['locale']
148:
149:     return flask.redirect(flask.url_for('index'))
150:
151: #-----
152:
153: @app.route('/logoutapp', methods=['GET'])
154: def logoutapp():
155:
156:     # Log out of the application.
157:     flask.session.clear()
158:     html_code = flask.render_template('loggedout.html')
159:     response = flask.make_response(html_code)
160:     return response
161:
162: #-----
163:
164: @app.route('/logoutgoogle', methods=['GET'])
165: def logoutgoogle():
166:
167:     # Log out of the application.
168:     flask.session.clear()
169:
170:     # Log out of Google.
171:     flask.abort(flask.redirect(
172:         'https://mail.google.com/mail/u/0/?logout&hl=en'))
173:
174: #-----
175:
176: def authenticate():
177:
178:     if 'email' not in flask.session:
179:         flask.abort(flask.redirect(flask.url_for('login')))
180:
181:     return flask.session.get('email')

```

## PennyAdmin18Auth0/runserver.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
3: #-----
4: # runserver.py
5: # Author: Bob Dondero
6: #-----
7:
8: import sys
9: import penny
10:
11: PORT = 3000
12:
13: def main():
14:
15:     if len(sys.argv) != 1:
16:         print('Usage: ' + sys.argv[0], file=sys.stderr)
17:         sys.exit(1)
18:
19:     try:
20:         penny.app.run(host='0.0.0.0', port=PORT, debug=True)
21:     except Exception as ex:
22:         print(ex, file=sys.stderr)
23:         sys.exit(1)
24:
25: if __name__ == '__main__':
26:     main()
```

## blank (Page 1 of 1)

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

**PennyAdmin18Auth0/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
13:   ('rdonero@princeton.edu');
14: INSERT INTO authorizedusers (username) VALUES
15:   ('bwk@princeton.edu');
```

**PennyAdmin18Auth0/footer.html (Page 1 of 1)**

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



## PennyAdmin18Auth0/database.py (Page 1 of 3)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # database.py
5: # Author: Bob Dondero
6: #-----
7:
8: import os
9: import sqlalchemy
10: import sqlalchemy.orm
11: import dotenv
12:
13: #-----
14:
15: dotenv.load_dotenv()
16: _database_url = os.getenv('DATABASE_URL', 'sqlite:///penny.sqlite')
17: _database_url = _database_url.replace('postgres://', 'postgresql://')
18:
19: #-----
20:
21: Base = sqlalchemy.orm.declarative_base()
22:
23: class Book (Base):
24:     __tablename__ = 'books'
25:     isbn = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
26:     author = sqlalchemy.Column(sqlalchemy.String)
27:     title = sqlalchemy.Column(sqlalchemy.String)
28:
29: class AuthorizedUser (Base):
30:     __tablename__ = 'authorizedusers'
31:     username = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
32:
33: _engine = sqlalchemy.create_engine(_database_url)
34:
35: #-----
36:
37: def get_books():
38:
39:     books = []
40:
41:     with sqlalchemy.orm.Session(_engine) as session:
42:         query = session.query(Book)
43:         table = query.all()
44:         for row in table:
45:             book = {'isbn': row.isbn, 'author': row.author,
46:                   'title': row.title}
47:             books.append(book)
48:
49:     return books
50:
51: #-----
52:
53: def add_book(isbn, author, title):
54:
55:     with sqlalchemy.orm.Session(_engine) as session:
56:         row = Book(isbn=isbn, author=author, title=title)
57:         session.add(row)
58:         try:
59:             session.commit()
60:             return True
61:         except sqlalchemy.exc.IntegrityError:
62:             return False
63:
64: #-----
65:

```

## PennyAdmin18Auth0/database.py (Page 2 of 3)

```

66: def delete_book(isbn):
67:
68:     with sqlalchemy.orm.Session(_engine) as session:
69:         session.query(Book).filter(Book.isbn==isbn).delete()
70:         session.commit()
71:
72: #-----
73:
74: def is_authorized(username):
75:
76:     with sqlalchemy.orm.Session(_engine) as session:
77:         query = session.query(AuthorizedUser) \
78:             .filter(AuthorizedUser.username==username)
79:         try:
80:             query.one()
81:             return True
82:         except sqlalchemy.exc.NoResultFound:
83:             return False
84:
85: #-----
86:
87: # For testing:
88:
89: def _write_books(books):
90:     for book in books:
91:         print('%s | %s | %s' % (book['isbn'], book['author'],
92:                               book['title']))
93:
94: def _test():
95:     print('-----')
96:     print('Testing get_books()')
97:     print('-----')
98:     print()
99:     books = get_books()
100:    _write_books(books)
101:    print()
102:
103:    print('-----')
104:    print('Testing add_book()')
105:    print('-----')
106:    print()
107:    successful = add_book('456', 'Kernighan', 'New Book')
108:    if successful:
109:        print('Add was successful')
110:        print()
111:        books = get_books()
112:        _write_books(books)
113:        print()
114:    else:
115:        print('Add was unsuccessful')
116:        print()
117:        _write_books(books)
118:        print()
119:    successful = add_book('456', 'Kernighan', 'New Book')
120:    if successful:
121:        print('Add was successful')
122:        print()
123:        books = get_books()
124:        _write_books(books)
125:        print()
126:    else:
127:        print('Add was unsuccessful')
128:        print()
129:        _write_books(books)
130:        print()

```

## PennyAdmin18Auth0/database.py (Page 3 of 3)

```

131:
132:     print('-----')
133:     print('Testing delete_book()')
134:     print('-----')
135:     print()
136:     delete_book('456')
137:     books = get_books()
138:     _write_books(books)
139:     print()
140:     delete_book('456')
141:     books = get_books()
142:     _write_books(books)
143:     print()
144:
145:     print('-----')
146:     print('Testing is_authorized()')
147:     print('-----')
148:     print()
149:     print(is_authorized('rdontero'))
150:     print(is_authorized('rdontero2'))
151:
152: if __name__ == '__main__':
153:     _test()

```

## PennyAdmin18Auth0/auth.py (Page 1 of 2)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # auth.py
5: # Author: Bob Dondero
6: #-----
7:
8: import os
9: import urllib.parse
10: import flask
11: import dotenv
12: import authlib.integrations.flask_client
13:
14: from top import app
15:
16: #-----
17:
18: dotenv.load_dotenv()
19: AUTH0_CLIENT_ID = os.environ.get('AUTH0_CLIENT_ID')
20: AUTH0_CLIENT_SECRET = os.environ.get('AUTH0_CLIENT_SECRET')
21: AUTH0_DOMAIN = os.environ.get("AUTH0_DOMAIN")
22:
23: oauth = authlib.integrations.flask_client.OAuth(app)
24:
25: oauth.register(
26:     'auth0',
27:     client_id=AUTH0_CLIENT_ID,
28:     client_secret=AUTH0_CLIENT_SECRET,
29:     client_kwargs={'scope': 'openid profile email'},
30:     server_metadata_url='https://' + AUTH0_DOMAIN +
31:         '/.well-known/openid-configuration')
32:
33: #-----
34:
35: @app.route('/callback', methods=["GET", "POST"])
36: def callback():
37:     try:
38:         token = oauth.auth0.authorize_access_token()
39:         flask.session['user'] = token
40:     except Exception:
41:         pass
42:     return flask.redirect('/')
43:
44: #-----
45:
46: @app.route('/login', methods=['GET'])
47: def login():
48:     return oauth.auth0.authorize_redirect(
49:         redirect_uri=flask.url_for('callback', _external=True))
50:
51: #-----
52:
53: @app.route('/logout', methods=['GET'])
54: def logout():
55:     flask.session.clear()
56:     return flask.redirect(
57:         'https://' + AUTH0_DOMAIN + '/v2/logout?'
58:         + urllib.parse.urlencode(
59:             {
60:                 'returnTo': flask.url_for('loggedout', _external=True),
61:                 'client_id': AUTH0_CLIENT_ID,
62:             },
63:             quote_via=urllib.parse.quote_plus))
64:
65: #-----

```

## PennyAdmin18Auth0/auth.py (Page 2 of 2)

```
66:
67: @app.route('/loggedout', methods=['GET'])
68: def loggedout():
69:     html_code = flask.render_template('loggedout.html')
70:     response = flask.make_response(html_code)
71:     return response
72:
73: #-----
74:
75: # Authenticate the remote user, and return the user's username.
76: # Do not return unless the user is successfully authenticated.
77:
78: def authenticate():
79:
80:     # If the user is in the session, then the user was
81:     # authenticated previously. So return the username.
82:     if 'user' in flask.session:
83:         return flask.session.get('user')['userinfo']['name']
84:
85:     flask.abort(flask.redirect('/login'))
```