pythonspot.com

Login authentication with Flask – Python Tutorial

10-12 minutes

The Flask Logo

In this tutorial you will learn how to build a login web app with Python using Flask.

Related course

Python Flask: Make Web Apps with Python

Installing Flask

Install **Flask** using the command below:

Create a file called hello.py

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def index():
return "Hello World!"

if __name__ == "__main__":
app.run(host='0.0.0.0', port=4000)
```

Finally run the web app using this command:

Open http://localhost:4000/ in your webbrowser, and “Hello World!” should appear.

Building a Flask login screen

Create this Python file and save it as app.py:

```
from flask import Flask
from flask import Flask, flash, redirect, render_template, request,
session, abort
import os

app = Flask(__name__)
```

```
@app.route('/')
def home():
if not session.get('logged_in'):
return render_template('login.html')
else:
return "Hello Boss!"

@app.route('/login', methods=['POST'])
def do_admin_login():
if request.form['password'] == 'password' and
request.form['username'] == 'admin':
session['logged_in'] = True
else:
flash('wrong password!')
return home()

if __name__ == "__main__":
app.secret_key = os.urandom(12)
app.run(debug=True,host='0.0.0.0', port=4000)
```

There are two routes (paths you can see in your browser URL bar) created here:

```
@app.route('/')
@app.route('/login', methods=['POST'])
```

The first one displays the login screen or the home screen, based on the condition if you are logged in. The second route validates the login variables on login.

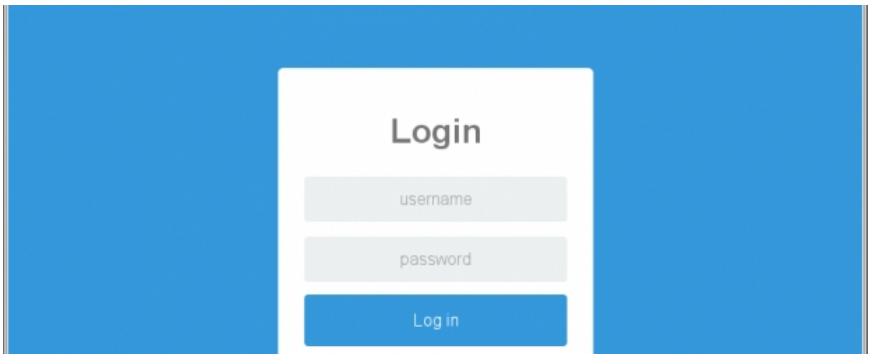
We create the directory /templates/. Create the file /templates/login.html with this code:

```
{% block body %}
{% if session['logged_in'] %}

You're logged in already!

{% else %}
<form action="/login" method="POST">
  <input type="username" name="username" placeholder="Username">
<input type="password" name="password" placeholder="Password">
<input type="submit" value="Log in">
</form>{% endif %}
{% endblock %}
```

Run the web app using this command:

Open http://localhost:4000/ in your webbrowser, and the login screen should appear. The

login credentials are shown in the *do_admin_login()* function.



Pythonspot.com Login Screen Python

Making it look amazing

While functional, the login screen looks like an early 90s User Interface (UI). We picked a random login template from codepen.io. We create the directory /static/ with the file style.css.

```
* {
box-sizing: border-box;
}

*:focus {
outline: none;
}
body {
font-family: Arial;
background-color: #3498DB;
padding: 50px;
}
.login {
margin: 20px auto;
width: 300px;
}
.login-screen {
background-color: #FFF;
padding: 20px;
border-radius: 5px
}

.app-title {
text-align: center;
color: #777;
}
```

```
.login-form {
text-align: center;
}
.control-group {
margin-bottom: 10px;
}

input {
text-align: center;
background-color: #ECF0F1;
border: 2px solid transparent;
border-radius: 3px;
font-size: 16px;
font-weight: 200;
padding: 10px 0;
width: 250px;
transition: border .5s;
}

input:focus {
border: 2px solid #3498DB;
box-shadow: none;
}

.btn {
border: 2px solid transparent;
background: #3498DB;
color: #ffffff;
font-size: 16px;
line-height: 25px;
padding: 10px 0;
text-decoration: none;
text-shadow: none;
border-radius: 3px;
box-shadow: none;
transition: 0.25s;
display: block;
width: 250px;
margin: 0 auto;
}
```

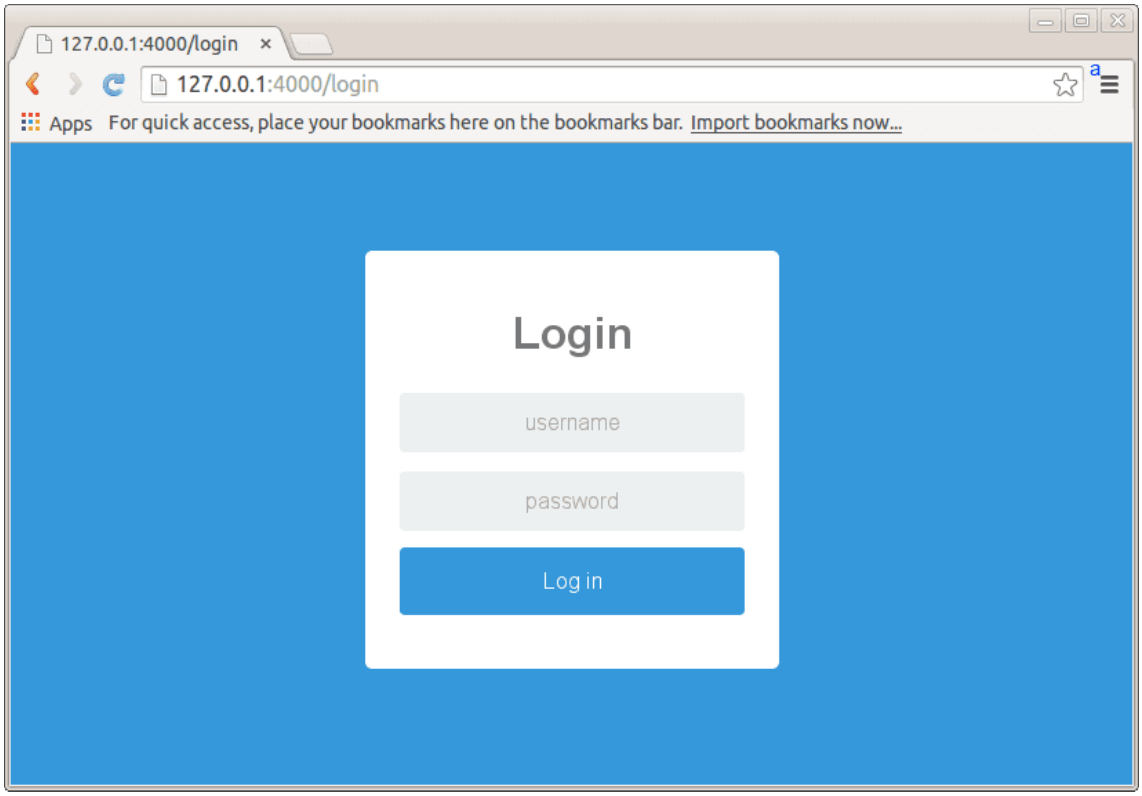
```
.btn:hover {
background-color: #2980B9;
}

.login-link {
font-size: 12px;
color: #444;
display: block;
margin-top: 12px;
}
```

Modify the login.html template as:

```
                        <link rel="stylesheet" href="/static
/style.css" type="text/css">
{% block body %}
{% if session['logged_in'] %}

You're logged in already!

{% else %}

<form action="/login" method="POST">
<div class="login">
<div class="login-screen">
<div class="app-title">
<h1>Login</h1>
</div>
<div class="login-form">
<div class="control-group">
                                <input type="text" class="login-
field" value="" placeholder="username" name="username">
<label class="login-field-icon fui-user" for="login-name"></label>
</div>
<div class="control-group">
                                <input type="password"
class="login-field" value="" placeholder="password"
name="password">
<label class="login-field-icon fui-lock" for="login-pass"></label>
</div>
<input type="submit" value="Log in" class="btn btn-primary btn-
large btn-block">
```

```
</div>
</div>
</div>
</form>{% endif %}
{% endblock %}
```

Once you restart the application this screen should appear:



Python login screen Flask

Awesome, ain't it? 🙂

What about logout?

As you may have seen, there is no logout button or functionality. Creating that is very easy. The solution proposed below is only one of the many solutions. We create a new route /logout which directs to the function logout(). This function clears the session variable and returns to the login screen.

```
@app.route("/logout")
def logout():
session['logged_in'] = False
return home()
```

The full code:

```
from flask import Flask
from flask import Flask, flash, redirect, render_template, request,
session, abort
```

```
import os

app = Flask(__name__)

@app.route('/')
def home():
if not session.get('logged_in'):
return render_template('login.html')
else:
return "Hello Boss!  <a href="/logout">Logout</a>"

@app.route('/login', methods=['POST'])
def do_admin_login():
if request.form['password'] == 'password' and
request.form['username'] == 'admin':
session['logged_in'] = True
else:
flash('wrong password!')
return home()

@app.route("/logout")
def logout():
session['logged_in'] = False
return home()

if __name__ == "__main__":
app.secret_key = os.urandom(12)
app.run(debug=True,host='0.0.0.0', port=4000)
```

Connecting a database

If you want a multi-user login system, you should add a database layer to the application. Flask does not have out of the box database support. You have to use a third party library if you want database support. In this tutorial we will use SqlAlchemy. If you do not have that installed type:

```
$ sudo pip install Flask-SqlAlchemy
```

SQLAlchemy is an SQL toolkit and object-relational mapper (ORM) for the Python programming language. It has support for MySQL, Microsoft SQL Server and many more relational database management systems. If all of these terms are unfamiliar to you, just keep reading.

Create the file ***tabledef.py:***

```
from sqlalchemy import *
from sqlalchemy import create_engine, ForeignKey
from sqlalchemy import Column, Date, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, backref

engine = create_engine('sqlite:///tutorial.db', echo=True)
Base = declarative_base()

########################################################################
class User(Base):
"""""
__tablename__ = "users"

id = Column(Integer, primary_key=True)
username = Column(String)
password = Column(String)

#----------------------------------------------------------------------
def __init__(self, username, password):
"""""
self.username = username
self.password = password

# create tables
Base.metadata.create_all(engine)
```

Execute it with:

This file will create the database structure. Inside the directory you will find a file called *tutorial.db.* Create a file called **dummy.py** which will contain this code:

```
import datetime
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from tabledef import *

engine = create_engine('sqlite:///tutorial.db', echo=True)

# create a Session
Session = sessionmaker(bind=engine)
session = Session()

user = User("admin","password")
```

```
session.add(user)

user = User("python","python")
session.add(user)

user = User("jumpiness","python")
session.add(user)

# commit the record the database
session.commit()

session.commit()
```

Execute:

This will put dummy data into your database. Finally we update our **app.py**

Validating the login credentials with SqlAlchemy

The next step is to write the functionality that validates the user and password exist in the database. Using SqlAlchemy we can do this (dummy/pseudo code):

```
@app.route('/test')
def test():

POST_USERNAME = "python"
POST_PASSWORD = "python"

Session = sessionmaker(bind=engine)
s = Session()
query = s.query(User).filter(User.username.in_([POST_USERNAME]),
User.password.in_([POST_PASSWORD]) )
result = query.first()
if result:
return "Object found"
else:
return "Object not found " + POST_USERNAME + " " + POST_PASSWORD
```

We use SqlAlchemys Oject Relational Mapping (ORM). We map the objects to relational database tables and vice versa. The definition (User) is given in tabledef.py. The s.query function is where the query is build. We have two conditions: the username and password must match. The query.first() returns true if the object exists, false if it does not. This gives us this total code:

```
from flask import Flask
from flask import Flask, flash, redirect, render_template, request,
session, abort
```

```
import os
from sqlalchemy.orm import sessionmaker
from tabledef import *
engine = create_engine('sqlite:///tutorial.db', echo=True)

app = Flask(__name__)

@app.route('/')
def home():
if not session.get('logged_in'):
return render_template('login.html')
else:
return "Hello Boss!  <a href="/logout">Logout</a>"

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

POST_USERNAME = str(request.form['username'])
POST_PASSWORD = str(request.form['password'])

Session = sessionmaker(bind=engine)
s = Session()
query = s.query(User).filter(User.username.in_([POST_USERNAME]),
User.password.in_([POST_PASSWORD]) )
result = query.first()
if result:
session['logged_in'] = True
else:
flash('wrong password!')
return home()

@app.route("/logout")
def logout():
session['logged_in'] = False
return home()

if __name__ == "__main__":
app.secret_key = os.urandom(12)
app.run(debug=True,host='0.0.0.0', port=4000)
```

You can now login with any user defined in the database table.

What about security?

We have demonstrated a simple login app above. However, it is your job to properly secure it. There are many guys out there that are going to try to break into your app.

Download Flask Examples

Best practices:

- Hash your database passwords. Don't store them in plain text.
- Secure the connection, use HTTPS.
- Log the failed login attempts.
- Use captcha to prevent brute force of logins.
- Others? write them in comments.