

User Login in Flask without Login Extensions

Posted December 13, 2017

In this tutorial we will learn how to build simple user authentication mechanism in Flask without using any login extensions. It's good idea to use existing extensions however if you want more control over how user session cookies are getting handled or if you are not sure about working of an existing extensions then you can quickly roll out your own code to build something very simple. We will make use of *Flask-SqlAlchemy* to read and write into the database.

Application Structure

- User DB - Store user details inside sqlite database.
- Sign Up Form - Creating new users.
- Login Form - User Login.
- Home Page - Home page for authorized users.
- Logout route - To logout users by removing cookie.

You can follow complete tutorial or directly jump into the [python code](#). At the [end of the tutorial](#) we will push working code on github.

User Database

We will create authentication database `auth.db` in Sqlite with single table `User`. This table will have following three columns.

User	
Column	Description
uid (pk)	Integer primary key for each user.
username (unique)	String username with unique constraint assigned to it
pass_hash	String to store password hash of user password

We will create `auth.db` with following `User` model using Flask SQLAlchemy.

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///auth.db'
```

Tags

[Davmail](#)
[Flask](#)
[Google Cloud](#)
[Informatica](#)
[Linux](#)
[Localtunnel](#)
[Open Source](#)
[Python](#)
[Raspberry Pi](#)
[Thunderbird](#)
[Ubuntu](#)
[Web](#)
[Windows](#)
[Wordpress](#)



[Feeds](#)

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///auth.db'
```

```
class User(db.Model):  
    uid = db.Column(db.Integer, primary_key=True)  
    username = db.Column(db.String(100), unique=True, nullable=False)  
    pass_hash = db.Column(db.String(100), nullable=False)  
  
    def __repr__(self):  
        return ' ' % self.username
```

Create empty db with User table

```
db.create_all()
```

Templates

We will create following three `jinja2` templates to serve html pages.

Signup Form

```
Username :   
Password :   

```

Login Form

```
Username :   
Password :   

```

User Home Page

Simple **Hello User** page for authorized user with logout link.

```
<h2>Hello {{username.title()}}</h2>  
<p><a href="{{ url_for('logout', username=username) }}">Logout</a></p>
```

Login App with Flask Session

1. We will authenticate user on login and store session cookie for user's session.
2. This session cookie will set value username equal to True. For example if Bob logs in successfully then

- we will store `session["Bob"]=True` .
3. For subsequent requests from Bob we will check value of `session["Bob"]` . We will only allow Bob to access his home page if value is `True` .
 4. If there is no **Bob** cookie or **Bob** cookie with Value other than True then will abort request with `HTTP 401` .
 5. Upon logout we will remove Bob's session cookie.

If you don't want to expose Username inside cookie then you can store some other unique value for the user.

Required Imports

We would need following imports from `flask` , `flask_sqlalchemy` and `werkzeug.security` .

Note that we have also imported functions `generate_password_hash` and `check_password_hash` from werkzeug's security module to [store hashed user passwords in database securely](#).

```
from flask import Flask, render_template, request, url_for, redirect, flash, \
session, abort
from flask_sqlalchemy import sqlalchemy, SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
```

Application Secret Key

To make use of flask session and message flashing we set application secret key. It is of **paramount importance** to configure *strong secret key* for the security of the application. Flask will make use of this key to sign session cookies.

```
app.config['SECRET_KEY'] = 'configure strong secret key here'
```

Routes

We will build following four flask routes for signup, login, user home page and logout.

1. Signup

This route will allow new user to register an account. Note that how it's making use of `generate_password_hash` to **store hashed password** into database. It will existing username in database and flash message. Once user is created it will redirect user to login page.

```
@app.route("/signup/", methods=["GET", "POST"])
def signup():
    if request.method == "POST":
        username = request.form['username']
        password = request.form['password']
```

```

if not (username and password):
    flash("Username or Password cannot be empty")
    return redirect(url_for('signup'))
else:
    username = username.strip()
    password = password.strip()

    # Returns salted pwd hash in format : method$salt$hashedvalue
    hashed_pwd = generate_password_hash(password, 'sha256')

    new_user = User(username=username, pass_hash=hashed_pwd)
    db.session.add(new_user)

    try:
        db.session.commit()
    except sqlalchemy.exc.IntegrityError:
        flash("Username {u} is not available.".format(u=username))
        return redirect(url_for('signup'))

    flash("User account has been created.")
    return redirect(url_for("login"))

return render_template("signup.html")

```

2. Login

This route will serve login page for User. It will accept Username and Password in plaintext. Using `check_password_hash` it will check plaintext password with hashed value stored in database. If hash value matches then it will store session cookie with `session[username] = True`.

```

@app.route("/login/", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form['username']
        password = request.form['password']

        if not (username and password):
            flash("Username or Password cannot be empty.")
            return redirect(url_for('login'))
        else:
            username = username.strip()
            password = password.strip()

            user = User.query.filter_by(username=username).first()

            if user and check_password_hash(user.pass_hash, password):

```

```

        session[username] = True
        return redirect(url_for("user_home", username=username))
    else:
        flash("Invalid username or password.")

    return render_template("login_form.html")

```

3. User Home

If user is successfully logged in then it should have session cookie for its username with value `True`. This route will check that value and show user its home page. It will abort request with `401` for the request with invalid cookie.

```

@app.route("/user/< username>/")
def user_home(username):
    if not session.get(username):
        abort(401)

    return render_template("user_home.html", username=username)

```

4. Logout

This route will get invoked when user logs out of the application. It will remove existing user session cookie.

```

@app.route("/logout/< username>")
def logout(username):
    session.pop(username, None)
    flash("successfully logged out.")
    return redirect(url_for('login'))

```

Security Caveats

- Always use HTTPS for deployment to serve cookies securely.
- Test application locally with HTTPS before deployment.
- Consider setting expiry date on cookie as flask does not set expiry date on the cookie by default. With flask defaults Session will be expired once browser is closed.
- Never store any sensitive information inside cookie. In an event of compromise flask session cookie can be easily decoded.
- Configure strong application secret before making use of flask's session.
- Use CSRF tokens on forms to prevent cross site request forgery.

Conclusion

CONCLUSION

We can build simple user authentication using flask's session to gain more control on user's session cookie. If you want to build complex system then you can consider using flask's user management extensions. We have pushed working source code of [simple login application](#) on github. Feel free to use it in your projects. If you want to create single user login system without database then learn how [HTTP Basic Access Authentication](#) works and [implement it in flask](#).

Tagged Under : [Flask Python Web](#)


[Previous](#)[Next](#)


ABOUT

Techmonger is a web log which shares interesting articles about computer science, programming and web development.

PAGES

[Privacy Policy](#)
[Contact](#)

 **Pagamenti sicuri al 100%**



This website uses cookies to ensure you get the best experience on our website. [Learn more](#)

Accept