How To Add Authentication to Your App with Flask-Login

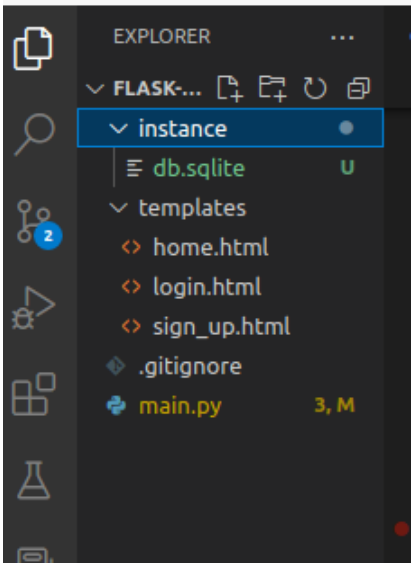
Last Updated : 24 Mar, 2023

Whether it is building a simple blog or a social media site, ensuring user sessions are working correctly can be tricky. Fortunately, Flask-Login provides a simplified way of managing users, which includes easily logging in and out users, as well as restricting certain pages to authenticated users. In this article, we will look at how we can add Authentication to Your App with Flask-Login in Flask using Python. To start, install flask, flask-login, and flask-sqlalchemy:

- Flask-Login helps us manage user sessions
- Flask-SQLAlchemy helps us store our user's data, such as their username and password

```
pip install flask flask-login flask-sqlalchemy
```

File structure



Stepwise Implementation

Step 1: Import the necessary modules.

We first import the classes we need from Flask, Flask-SQLAlchemy, and Flask-Login. We then create our flask application, indicate what database Flask-SQLAlchemy should connect to, and initialize the Flask-SQLAlchemy extension. We also need to specify a secret key, which can be any random string of characters, and is necessary as Flask-Login requires it to sign session cookies for protection again data tampering. Next, we initialize the *LoginManager* class from Flask-Login, to be able to log in and out users.

Python3

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager


# Create a flask application
app = Flask(__name__)

# Tells flask-sqlalchemy what database to connect to
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///db.sqlite"
# Enter a secret key
app.config["SECRET_KEY"] = "ENTER YOUR SECRET KEY"
# Initialize flask-sqlalchemy extension
db = SQLAlchemy()

# LoginManager is needed for our application
# to be able to log in and out users
login_manager = LoginManager()
login_manager.init_app(app)
```

Step 2: Create a User Model & Database

To be able to store users' information such as their username and password, we need to create a table with Flask-SQLAlchemy, this is done by creating a model that represents the information we want to store. In this case, we first create a Users class and make it a subclass of db.Model to make it a model with the help of Flask-SQLAlchemy. We also make the Users class a subclass of UserMixin, which will help to implement properties such as is_authenticated

to the Users class. We will also need to create columns within the user model, to store individual attributes, such as the user's username. When creating a new column, we need to specify the datatype such as db.Integer and db.String as well. When creating columns, we also need to specify keywords such as unique = True, if we want to ensure values in the column are unique, nullable = False, which indicates that the column's values cannot be NULL, and primary_key = True, which indicates that the row can be identified by that primary_key index. Next, the db.create_all method is used to create the table schema in the database.

Python3

```
# Create user model
class Users(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(250), unique=True,
                         nullable=False)
    password = db.Column(db.String(250),
                         nullable=False)


# Initialize app with extension
db.init_app(app)
# Create database within app context

with app.app_context():
    db.create_all()
```

Step 3: Adding a user loader

Before implementing the functionality for authenticating the user, we need to specify a function that Flask-Login can use to retrieve a user object given a user id. This functionality is already implemented by Flask-SQLAlchemy, we simply need to query and use the *get* method with the user id as the argument.

Python3

```
# Creates a user loader callback that returns the user object given an id
@login_manager.user_loader
def loader_user(user_id):
```

```
    return Users.query.get(user_id)
```

Step 4: Registering new accounts with Flask-Login

Add the following code to a file name sign_up.html in a folder called templates. To allow the user to register an account, we need to create the HTML. This will need to contain a form that allows the user to enter their details, such as their username and chosen password.

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Sign Up</title>
    <style>
      h1 {
        color: green;
      }
    </style>
  </head>
  <body>
    <nav>
      <ul>
        <li><a href="/login">Login</a></li>
        <li><a href="/register">Create account</a></li>
      </ul>
    </nav>
    <h1>Create an account</h1>
    <form action="#" method="post">
      <label for="username">Username:</label>
      <input type="text" name="username" />
      <label for="password">Password:</label>
      <input type="password" name="password" />
      <button type="submit">Submit</button>
    </form>
  </body>
</html>
```

Create a route that renders the template, and creates the user account if they make a POST request.

We create a new route with Flask by using the **@app.route decorator**. The @app.route decorator allows us to specify the route it accepts, and the methods it should accept. By default, it only accepts requests using the GET method, but when the form is submitted it is done using a POST request, so we'll need to make POST an accepted method for the route as well. Within the register function that is called whenever the user visits that route, we can check if the method used was a POST request using the request variable that Flask provides and that needs to be imported. If a post request was made, this indicates the user is trying to register a new account, so we create a new user using the Users model, with the username and password set to whatever the user entered, which we can get by using **request.form**.get Lastly, we add the user object that was created to the session and commit the changes made. Once the user account has been created, we redirect them to a route with a callback function called "login", which we will create in a moment. Ensure that you also import the redirect and url_for functions from Flask.

Python3

```
@app.route('/register', methods=["GET", "POST"])
def register():
  # If the user made a POST request, create a new user
    if request.method == "POST":
        user = Users(username=request.form.get("username"),
                     password=request.form.get("password"))
        # Add the user to the database
        db.session.add(user)
        # Commit the changes made
        db.session.commit()
        # Once user account created, redirect them
        # to login route (created later on)
        return redirect(url_for("login"))
    # Renders sign_up template if user made a GET request
    return render_template("sign_up.html")
```

Step 5: Allowing users to log in with Flask-Login

Like with creating the registered route, we first need a way for the user to log in through an HTML form. Add the following code to a file named login.html in the same templates folder.

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Login</title>
    <style>
        h1{
          color: green;
          }
    </style>
  </head>
  <body>
    <nav>
      <ul>
        <li><a href="/login">Login</a></li>
        <li><a href="/register">Create account</a></li>
      </ul>
    </nav>
    <h1>Login to your account</h1>
    <form action="#" method="post">
      <label for="username">Username:</label>
      <input type="text" name="username" />
      <label for="password">Password:</label>
      <input type="password" name="password" />
      <button type="submit">Submit</button>
    </form>
  </body>
</html>
```

Add the functionality to log in to the user within a login function for the /login route.

With the login route, we do the same thing of checking if the user made a POST request. If they did, we filter the users within the database for a user with the same username as the one being submitted. Next, we check if that

user has the same password as the password the user entered in the form. If they are the same, we log-in to the user by using the *login_user* function provided by Flask-Login. We can then redirect the user back to a route with a function called “home”, which we will create in a moment. If the user didn’t make a *POST* request, and instead a *GET* request, then we’ll render the login template.

Python3

```
@app.route("/login", methods=["GET", "POST"])
def login():
    # If a post request was made, find the user by
    # filtering for the username
    if request.method == "POST":
        user = Users.query.filter_by(
            username=request.form.get("username")).first()
        # Check if the password entered is the
        # same as the user's password
        if user.password == request.form.get("password"):
            # Use the login_user method to log in the user
            login_user(user)
            return redirect(url_for("home"))
        # Redirect the user back to the home
        # (we'll create the home route in a moment)
    return render_template("login.html")
```

Step 6: Conditionally rendering HTML based on the user’s authentication status with Flask-Login

When using Flask, it uses Jinja to parse the templates. Jinja is a templating engine that allows us to add code, such as if-else statements within our HTML, we can then use it to conditionally render certain elements depending on the user’s authentication status for example the current_user variable is exported by Flask-Login, and we can use it within the Jinja template to conditionally render HTML based on the user’s authentication status.

HTML

```
<!DOCTYPE html>
<html lang="en">
```

```
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Home</title>
  </head>
  <body>
    <nav>
```



```
        <li><a href="/register">Create account</a></li>
      </ul>
    </nav>
    {% if current_user.is_authenticated %}
    <h1>You are logged in</h1>
    {% else %}
    <h1>You are not logged in</h1>
    {% endif %}
  </body>
</html>
```

Add the functionality to render the homepage when the user visits the "/" route.

This will then render the template of home.html whenever the user visits the "/" route. After running the code in main.py, navigate to http://127.0.0.1:5000/

Python3

```
@app.route("/")
def home():
    # Render home.html on "/" route
    return render_template("home.html")
```

Step 7: Adding Logout Functionality

Here, we will **update the home.html** template to the following to add a logout link, and this will give the homepage a link to log out the user if they are currently logged in.

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Home</title>
    <style>
      h1 {
        color: green;
      }
    </style>
  </head>
  <body>
    <nav>
      <ul>
        <li><a href="/login">Login</a></li>
        <li><a href="/register">Create account</a></li>
      </ul>
    </nav>
    {% if current_user.is_authenticated %}
    <h1>You are logged in</h1>
    <a href="/logout">Logout</a>
    {% else %}
    <h1>You are not logged in</h1>
    {% endif %}
  </body>
</html>
```

Complete Code

Add the logout functionality and code initializer.

Python3

```
from flask import Flask, render_template, request, url_for, redirect
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, logout_user

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///db.sqlite"
app.config["SECRET_KEY"] = "abc"
db = SQLAlchemy()
```

```
login_manager = LoginManager()
login_manager.init_app(app)


class Users(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(250), unique=True, nullable=False)
    password = db.Column(db.String(250), nullable=False)


db.init_app(app)


with app.app_context():
    db.create_all()


@login_manager.user_loader
def loader_user(user_id):
    return Users.query.get(user_id)


@app.route('/register', methods=["GET", "POST"])
def register():
    if request.method == "POST":
        user = Users(username=request.form.get("username"),
                     password=request.form.get("password"))
        db.session.add(user)
        db.session.commit()
        return redirect(url_for("login"))
    return render_template("sign_up.html")


@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        user = Users.query.filter_by(
            username=request.form.get("username")).first()
        if user.password == request.form.get("password"):
            login_user(user)
            return redirect(url_for("home"))
    return render_template("login.html")


@app.route("/logout")
def logout():
    logout_user()
    return redirect(url_for("home"))
```

```
@app.route("/")
def home():
    return render_template("home.html")


if __name__ == "__main__":
    app.run()
```

Output:

To test the application, we would navigate to the /register route, create an account and we'll be redirected to the /login route. From there, we can log in, and we can verify that we have been logged in by the conditional HTML rendering.

Now, whenever the user is logged in, they can log out by clicking the link within the homepage, which will logout the user, by using the *logout_user* function provided by Flask-Login.

Previous Article

Sending Emails Using API in Flask-Mail

Next Article

Add User and Display Current Username in Flask

Similar Reads

How to Integrate Flask-Admin and Flask-Login

In order to merge the admin and login pages, we can utilize a short form or any other login method that only requires the username and password. This is know...

8 min read

Login and Registration Project Using Flask and MySQL

Project Title: Login and registration Project using Flask framework and MySQL Workbench. Type of Application (Category): Web application. Introduction: A...

6 min read

Flask login without database - Python

In this article, we will talk about Python-based Flask login without a database in this article. In order to use Flask login without a database in this method basical...

4 min read

Dockerize your Flask App

Python provides many ways to distribute your python projects. One such way is by using an important technology called Docker. Docker is an open-source...

4 min read

Documenting Flask Endpoint using Flask-Autodoc

Documentation of endpoints is an essential task in web development and being able to apply it in different frameworks is always a utility. This article discusses...

4 min read

How to use Flask-Session in Python Flask ?

Flask Session - Flask-Session is an extension for Flask that supports Server-side Session to your application.The Session is the time between the client logs in to...

4 min read

Minify HTML in Flask using Flask-Minify

Flask offers HTML rendering as output, it is usually desired that the output HTML should be concise and it serves the purpose as well. In this article, we would...

12 min read

Flask URL Helper Function - Flask url_for()

In this article, we are going to learn about the flask url_for() function of the flask URL helper in Python. Flask is a straightforward, speedy, scalable library, used f...

11 min read

Securing Django Admin login with OTP (2 Factor Authentication)

Multi factor authentication is one of the most basic principle when adding security for our applications. In this tutorial, we will be adding multi factor authentication...

2 min read

Using JWT for user authentication in Flask

Pre-requisite: Basic knowledge about JSON Web Token (JWT)I will be assuming you have the basic knowledge of JWT and how JWT works. If not, then I sugges...

7 min read

Article Tags : Python

Practice Tags : python
