User Authentication with Flask-Login

Brian Dickinson

University of Rochester

Fall 2019
Flask-Login - Management of user sessions for logged-in users
Werkzeug - Password hashing and verification
itsdangerous - Cryptographically secure token generation and verification

\(^0\)This stack is on the weaker side because PBKDF2 is not especially robust to GPU accelerated attacks, but this one is in the book and fine for this class. The most popular password hashing algorithms in order of security are PBKDF2 < Bcrypt < Scrypt < Argon2.
Securely Storing Passwords in a User Model

class User(db.Model):
    # . . .
    password_hash = db.Column(db.String(128))

@property
def password(self):
    raise AttributeError("password is write only")

@password.setter
def password(self, password):
    self.password_hash = generate_password_hash(password)

def verify_password(self, password):
    return check_password_hash(self.password_hash, password)
    # . . .
from flask_login import UserMixin

class User(UserMixin, db.Model):
    __tablename__ = "Users"
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(64), unique=True, index=True)
    username = db.Column(db.String(64), unique=True, index=True)
    password_hash = db.Column(db.String(128))
    role_id = db.Column(db.Integer,
                          db.ForeignKey("roles.id"))
Preparing App to Support User Login

```python
from flask import Flask
from flask_login import LoginManager

app = Flask(__name__)
app.config['SECRET_KEY'] = 'oEHYBreJ2QSefBdUhD19PkxC'
login_manager = LoginManager()
login_manager.login_view = 'login'
login_manager.init_app(app)

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField
from wtforms import BooleanField, SubmitField
from wtforms.validators import DataRequired, Length, Email

class LoginForm(FlaskForm):
    email = StringField("Email", validators=[DataRequired(),
                                              Length(1,64), Email()])
    password = PasswordField("Password", validators=[DataRequired()])
    remember_me = BooleanField("Keep me logged in")
    submit = SubmitField("Log In")
<!DOCTYPE html>
<html>
  <head>
    <title>Login</title>
  </head>
  <body>
    <form method="POST">
      {{ form.hidden_tag() }}
      {{ form.email.label }}: {{ form.email() }}<br>
      {{ form.password.label }}: {{ form.password() }}<br>
      {{ form.remember_me.label }}:<br>{{ form.remember_me() }}<br>
      {{ form.submit() }}
    </form>
  </body>
</html>
from flask import render_template, redirect, request
from flask import url_for, flash
from flask_login import login_user
@auth.route("/login", methods=["GET","POST"])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user is not None and \
            user.verify_password(form.password.data):
            login_user(user, form.remember_me.data)
            next = request.args.get("next")
            if next is None or not next.startswith("/"):
                next = url_for("index")
            return redirect(next)
        flash("Invalid Username or password.")
    return render_template("login_form.html", form=form)
<!-- DOCTYPE html -->
<html>
  
  . . .
  {% if current_user.is_authenticated %}
  <a href="{{ url_for('auth.logout') }}">
    <button type="button"> Log Out </button>
  </a>
  {% else %}
  <a href="{{ url_for('auth.login') }}">
    <button type="button"> Log In </button>
  </a>
  {% endif %}
  
  . . .
</html>
Protecting Routes

```python
from flask_login import login_required

@app.route("/secret")
@login_required
def secret():
    return "Congratulations on being logged in!"
```
Signing Users Out

@app.route("/logout")
@login_required
def logout():
    logout_user()
    flash("You have been logged out")
    return redirect(url_for("login"))
Registering New Users

Obviously you will need to register new users for your system to be fully functional. Most of the heavy lifting here is done in your WTForm definition

- Have the form validate that the email address is not already associated with an account
- Require that the user enter their email address twice to guarantee it is not mistyped
- Validate the username to ensure it meets your requirements (acceptable characters, uniqueness, possibly a profanity filter)
- Validate passwords meet minimum standards but don’t be too onerous (NIST has some new guidelines)
- Same as email address make them enter their password twice and verify that they match
Account Management Features and Requirements

- **Password Update**
  - Only allow changing password of currently logged in user
  - Require their current password in the same form
  - Require the new password twice for verification

- **Password Reset**
  - If security is not paramount users are accustomed to resetting their passwords using their verified email addresses
  - A unique signed token must be sent to the user in an email and they must submit it to the server within a short time frame (perhaps 15-30 minutes) preferably by clicking a link in the email or copying a URL into their address bar.

- **Email Address Change**
  - Only allow changing email address of currently logged in user
  - Require their current password in the same form
  - Require the new email address twice for verification
  - Give the new email address a pending status until the user goes through a verification process
Summary

1. User submits a GET request for the "/login" route
2. User submits a POST request with login credentials to 
   "/login" route
   1. Credentials are evaluated and either "login_user()" is called or 
      user is redirected to start over and told that the credentials 
      were invalid
   2. If credentials are correct then "login_user()" saves user login 
      state to the session. The user is then redirected to the home 
      page or some other appropriate destination.
3. User’s browser submits GET request to redirected route
4. The returned template may access the "current_user" object 
   provided by the session
5. This value (currently still null) is filled using the 
   "_get_user()" function and the user id saved in the current 
   session by the "login_user()" function.
Miguel Grinberg.
User logins.

Michele Preziuso.
Password hashing: Scrypt, bcrypt and argon2.