06 – User Authentication

web application အများစုမှာ ၎င်းကိုအသုံးပြုသူ users များနှင့် ပတ်သက်တဲ့ လမ်းကြောင်းခြေရာကို ထိမ်းသိမ်းထားဖို့ လိုအပ်ပါတယ်။ users တွေက application နဲ့ ချိတ်ဆက်တဲ့အချိန်မှာ အသုံးပြုမဲ့ user ဟာ မှန်ကန်ကြောင်း ထင်ရှားစေဖို့ authenticate ဖြစ်ဖို့ လိုပါတယ်။

ယေဘုယျ အများအားဖြင့် applications တွေဟာ user က ၎င်းမည်သူမည်ဝါ ဖြစ်ကြောင်း ကို ခွဲခြားရွေးထုတ်ခြင်းလုပ်ငန်း အစိတ်အပိုင်းတွေဖြစ်ကြတဲ့ email လိပ်စာတွေ ၊ username တွေနဲ့ ၊ password တို့လို secret key တွေကို ထောက်ပံ့ပေးရမဲ့ authentication method တွေကို အသုံးပြုပါတယ်။

Authentication system အတွက် အသုံးပြုလို့ရနိုင်တဲ့ extensions တွေ Flask မှာ အများအပြားရှိတဲ့အထဲကမှ Password hashing and varification အတွက် flask-bcrypt extension နဲ့ log-in users တွေအတွက် user sessions ကို manage လုပ်ဖို့ flask-login ကို သုံးပါမယ်။ အဲဒါကြောင့် ပထမဆုံး flask-bcrypt ကို install လုပ်ပြီး၊ စမ်းကြည့်ပါမယ်။

Terminal ကိုသွားပါ။ virtualenv ကို activate လုပ်ပါ။

```
(venv) pip install flask-bcrypt
```

နောက် Python shell ကို သွားပါ။

```
# Encrypt Password
>>> from flask-bcrypt import Bcrypt
>>> bcrypt = Bcrypt()
>>> bcrypt.generate_password_hash('testing')
>>> bcrypt.generate_password_hash('testing').decode('utf-8')

# Check Password
>>> hashed_pw = bcrypt.generate_password_hash('testing').decode('utf-8')
>>> bcrypt.check_password_hash(hashed_pw, 'testing')
```

User passwords တွေကို database ထဲမှာ လုံလုံခြုံခြုံ သိမ်းဆည်းထားနိုင်တဲ့ သော့ချက်ကတော့ passwords တွေကို မူရင်းအနေအထားအတိုင်း မထားသိုပဲ hash လုပ်လိုက်ဖို့ပါပဲ။ Password hashing function တစ်ခုမှာ password ကို input အနေနဲ့ ပေးလိုက်ရင်၊ cryptographic transformations နည်းလမ်း အသွယ်သွယ်ထဲက တစ်နည်းနည်းကို အသုံးချပြီး၊ hashing လုပ်ပေးပါလိမ့်မယ်။ original password နဲ့ ဆင်တူခြင်းမရှိပဲ၊ မူရင်းအတိုင်း ပြန်ပြောင်းနိုင်တဲ့ နည်းလမ်းကိုလည်း မသိနိုင်တဲ့ characters အတွဲသစ်တစ်ခု ကို ရရှိမှာဖြစ်ပါတယ်။

bcrypt ရဲ့ generate_password_hash() function က byte ကို ပေးတဲ့အတွက် decode method သုံးပြီး text string ပြောင်းသုံးပါမယ်။ user ပေးလိုက်တဲ့ password နဲ့ database မှာ သိမ်းထားတဲ့ password ကို တိုက်ဆိုင်စစ်ဆေးဖို့ check_password_hash() function ကို သုံးပါမယ်။
bcrypt ကို နည်းလည်ပြီဆိုရင် application ကို ဆက်ရေးကြည့်ပါမယ်။

__init__.py မှာ Bcrypt ကို import လုပ်ပြီး bcrypt instance တစ်ခုဖန်တီးပါမယ်။

```
from flask_bcrypt import Bcrypt
bcrypt = Bcrypt(app)
```

/register route ကို ပြင်ပါမယ်။ validate_on_submit() function က True return ပြန်တာနဲ့ form ရဲ့ password field ထဲမှာ ရိုက်ထားတဲ့ password ကို hash လုပ် string ပြောင်းပြီး variable တစ်ခုနဲ့ သိမ်းထားပါမယ်။ username field, email field တွေထဲက data တွေနဲ့ hash လုပ်ထားတဲ့ password ကို သုံးပြီး user ဆောက်ပါမယ်။ နောက် user ကို database ထဲသိမ်းပြီး account ဖန်တီးပြီးပြီဆိုတာနဲ့ login ဝင် ဖို့ message ပြပြီး၊ login form ကို အဆင်သင့် ချပေးထားပါမယ်။ပြီးရင် run ကြည့်ရအောင်။

```
@app.route("/register", methods=['GET', 'POST'])
def register():
    form = RegistrationForm()
    if form.validate_on_submit():
hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
    user = User(username=form.username.data, email=form.email.data,
password=hashed_password)
    db.session.add(user)
```

```
            db.session.commit()
        flash('Your account has been created! You are now able to log in', 'success')
    return redirect(url_for('login'))
        return render_template('register.html', title='Register', form=form)
```

ပထမအကြိမ် register လုပ်တာအဆင်ပြေပေမဲ့ username , email အတူတူနဲ့ နောက်တစ်ခါ register လုပ်တဲ့အခါ error ပြပါမယ်။ username, email အစရှိတဲ့ field တွေကို ဖန်တီးခဲ့တုန်းက unique ဖြစ်ရမယ်လို့ ဖန်တီးပေးထားတဲ့အတွက် validation error ပြတာပါ။ ဒီ error ကို handle လုပ်ဖို့ RegistrationForm class မှာ method နှစ်ခု ထည့်ပါမယ်။ code ကို ပေါင်းထည့်ပါ။ ပြီးရင် ပြန် run ကြည့်ပါမယ်။

```
    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('That username is taken. Please choose a
different
                                                                    one.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('That email is taken. Please choose a different
one.')
```

register အတွက် အားလုံးအဆင်ပြေပြီဆိုရင်၊ Login ကိုကြည့်ပါမယ်။ အရင်ဆုံး flask-login ကို install လုပ်ပါမယ်။

Terminal ကိုသွားပါ။ virtualenv ကို activate လုပ်ပါ။

```
(venv) pip install flask-login
```

Application ကို log in လုပ်တဲ့အခါ၊ users တွေရဲ့ authenticated state ကို user session မှာ မှတ်ထားရပါမယ်။ Flask-Login က application ရဲ့ ကိုယ်ပိုင် User objects တွေနဲ့ နီးနီးကပ်ကပ် အလုပ်လုပ်ပါတယ်။ application ရဲ့ User model နဲ့ အလုပ်လုပ်နိုင်ဖို့ အချို့သော properties နဲ့ methods တွေကို implement လုပ်ပေးဖို့လဲလိုပါတယ်။ properties နဲ့ methods တွေကို model class မှာ တိုက်ရိုက် အကောင်အထည်ဖော်ပေးနိုင်ပါတယ်။ ဒါပေမယ့်လည်း cases အများစုအတွက် သင့်လျော်ပြီး default implementations တွေပါရှိတဲ့ UserMixin class ကို Flask-Login က ထောက်ပံ့ပေးထားပါတယ်။ အဲတော့ User model ကို update လုပ်ရအောင်။ login_manager နဲ့ UserMixin ကို import လုပ်ပါ။

models.py မှာ လိုအပ်တဲ့ code ကို ပြထားတဲ့အတိုင်းထည့်ပါ။

```
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

@login_manager.user_loader ဆိုတဲ့ decorator က function ကို register လုပ်တာပါ။ logged-in user နဲ့ပတ်သတ်တဲ့ အချက်အလက်တွေကို database မှ ပြန်ယူဖို့ လိုအပ်တဲ့အခါမှာ call လုပ်ပါလိမ့်မယ်။ function မှာ user_id က string အနေနဲ့ passing လုပ်တဲ့အတွက် int ကို type conversion လုပ်ပေးပါမယ်။ ပြီးရင် login route ကို သွားပါမယ်။ login_user ကို import လုပ်ပါ။

```
@app.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 and bcrypt.check_password_hash(user.password,

            form.password.data):
        login_user(user, remember=form.remember.data)
        return redirect(url_for('home'))
    else:
```

```
                flash('Login Unsuccessful. Please check email and password',

        'danger')
            return render_template('login.html', title='Login', form=form)
```

Login form မှာ အချက်အလက်ဖြည့်ပြီး login button မှာ တစ်ချက်နှိပ်ပြီဆိုတာနဲ့ validate_on_submit() function က form variables တွေကို valid ဖြစ်မဖြစ်စစ်ပြီး user ကို login လုပ်ဖို့ ကြိုးစားပါလိမ့်မယ်။ ပထမ form မှာ ရိုက်ထည့်ပေးလိုက်တဲ့ email နဲ့ database ထဲမှ user ကို loading လုပ်ပါမယ်။ ပေးထားတဲ့ email လိပ်စာနဲ့ user ရှိတယ်ဆိုရင် check_password_hash() method နဲ့ password ကို စစ်ပါမယ်။ အားလုံးကိုက်ညီတယ်ဆိုရင်တော့ login_user() function ကနေ remember Boolean value နဲ့အတူ user ကို record လုပ်ပါမယ်။ ပြီးရင် home page ကို redirect လုပ်ထားပြီး၊ login မအောင်မြင်ခဲ့ရင်တော့ flash message နဲ့အတူ login form ကို ပြထားပါမယ်။

login လုပ်ပြီး home page ကို ရောက်သွားပေမဲ့ nav bar မှာ login နဲ့ register link တွေက ရှိနေပြီး၊ login form တွေကို ပြန်သွားလို့ရနေပါလိမ့်မယ်။ ဒါကြောင့် လက်ရှိ current_user ရှိမရှိကို စစ်ဆေးပါမယ်။ login နဲ့ register route မှာ current_user က authenticated ဖြစ်မဖြစ် စစ်ပေးတဲ့ code ကို ရေးပေးပါမယ်။ current_user ကို import လုပ်ပါ။

```
        if current_user.is_authenticated:
        return redirect(url_for('home'))
```

ပြီးရင် logout လုပ်ဖို့ route ရေးပါမယ်။ logout_user ကို import လုပ်ပြီး code ကိုရေးပါ။

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

login / logut ရပြီဆိုရင် home page ရဲ့ nav bar ကို ပြင်ပေးပါမယ်။ layout.html မှာ Jinja2 ရဲ့ code တစ်ချို့ သွားဖြည့်ပေးပါ။

```
          <div class="navbar-nav">
      {% if current_user.is_authenticated %}
```

```
            <a class="nav-item nav-link" href="{{ url_for('logout') }}">Logout</a>
        {% else %}
            <a class="nav-item nav-link" href="{{ url_for('login') }}">Login</a>
            <a class="nav-item nav-link" href="{{ url_for('register') }}">Register</a>
        {% endif %}
```

login လုပ် ထားတဲ့ user account ကိုကြည့်ဖို့ account.html နဲ့ route ကို ရေးပါမယ်။ login ဝင်ထားမှသာ username ဖော်ပြစေချင်ပြီး login user မရှိပါက login form ကို error message နဲ့ အတူ ပြပေးစေပါမယ်။ အဲဒီအတွက် login_required ကို import လုပ်ရပါမယ်။ နောက် login_view နဲ့ login_message_category ကို __init__.py မှာ initialize လုပ်ထားပါမယ်။

```
@app.route("/account")
@login_required
def account():
    return render_template('account.html', title='Account')
```

account.html

```
{% extends "layout.html" %}
{% block content %}
    <h1>{{ current_user.username }}</h1>
{% endblock content %}
```

User account ကိုကြည့်လို့ အဆင်ပြေသွားပါပြီ။ ဒါပေမယ့် login user မရှိပဲနဲ့လည်း account ကို ကြည့်မယ်ဆိုရင် login function ကို အလုပ်လုပ်ဖို့ပြောထားပါတယ်။ ဒါကြောင့် login form ကို URL နှစ်မျိုးနဲ့ သွားနိုင်သလို၊ ဘာနဲ့သွားသွား home ကိုပဲ redirect လုပ်ထားတာပါ။ အကယ်၍များ account url ကနေ user မရှိလို့ login ကို သွားပြီး login လုပ်ပြီးသွားရင် home ကို သွားမယ့်အစား user name ကို တခါတည်းပြပေးစေချင်တယ်ဆိုရင်တော့ next query string argument ထဲမှာ original URL ကို သိမ်းထားပြီး redirect ကိုလည်း ပြင်ပေးရပါမယ်။ next query string ကို request.args dictionary မှ

ပြန် access လုပ်နိုင်ပါတယ်။ ဒါကြောင့် request ကို import လုပ်ပြီး return redirect ကိုပြန်ပြင်ရေးကြရအောင်။

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
    return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
    user = User.query.filter_by(email=form.email.data).first()
    if user and bcrypt.check_password_hash(user.password,

            form.password.data):
        login_user(user, remember=form.remember.data)
        next_page = request.args.get('next')
        return redirect(next_page) if next_page else redirect(url_for('home'))
    else:
        flash('Login Unsuccessful. Please check email and password','danger')
    return render_template('login.html', title='Login', form=form)
```

ပြီးရင် ပြန် run ကြည့်ပါမယ်။

__init__.py

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

```
app = Flask(__name__)
app.config['SECRET_KEY'] = '5791628bb0b13ce0c676dfde280ba245'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
login_manager.login_message_category = 'info'

from flaskblog import routes
```

routes.py

```
from flask import render_template, url_for, flash, redirect, request
from flaskblog import app, db, bcrypt
from flaskblog.forms import RegistrationForm, LoginForm
from flaskblog.models import User, Post
from flask_login import login_user, current_user, logout_user, login_required

posts = [
    {
        ...
    },
    {
        ...
    }
```

```
]


@app.route("/")
@app.route("/home")
def home():
    return render_template('home.html', posts=posts)


@app.route("/about")
def about():
    return render_template('about.html', title='About')




@app.route("/register", methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username=form.username.data, email=form.email.data,
password=hashed_password)
        db.session.add(user)
        db.session.commit()
        flash('Your account has been created! You are now able to log in', 'success')
        return redirect(url_for('login'))
```

```
    return render_template('register.html', title='Register', form=form)


@app.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('home'))
        else:
            flash('Login Unsuccessful. Please check email and password', 'danger')
    return render_template('login.html', title='Login', form=form)


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


@app.route("/account")
@login_required
def account():
    return render_template('account.html', title='Account')
```

forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, BooleanField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from flaskblog.models import User

class RegistrationForm(FlaskForm):
     username = StringField('Username', validators=[DataRequired(), Length(min=2,

          max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')



    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('That username is taken. Please choose a different one.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('That email is taken. Please choose a different one.')
```

```
class LoginForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember = BooleanField('Remember Me')
    submit = SubmitField('Login')
```

models.py

```
from datetime import datetime
from flaskblog import db, login_manager
from flask_login import UserMixin

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    image_file = db.Column(db.String(20), nullable=False, default='default.jpg')
    password = db.Column(db.String(60), nullable=False)
    posts = db.relationship('Post', backref='author', lazy=True)

    def __repr__(self):
        return f"User('{self.username}', '{self.email}', '{self.image_file}')"

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    date_posted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    content = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

```
    def __repr__(self):
        return f"Post('{self.title}', '{self.date_posted}')"
```

layout.html

```
<!DOCTYPE html>
<html>
<head>
 …
</head>
<body>
    <header class="site–header">
      …
    </header>
    <main role="main" class="container">
…
    </main>

    <!–– Optional JavaScript ––>
    …
</body>
</html>
```

account.html

```
{% extends "layout.html" %}
{% block content %}
```

```
    <h1>{{ current_user.username }}</h1>
{% endblock content %}
```

Resources

```
# install crypt
pip install flask-bcrypt

# Encrypt Password
>>> from flask-bcrypt import Bcrypt
>>> bcrypt = Bcrypt()
>>> bcrypt.generate_password_hash('testing')
>>> bcrypt.generate_password_hash('testing').decode('utf-8')

# Check Password
>>> hashed_pw = bcrypt.generate_password_hash('testing').decode('utf-8')
>>> bcrypt.check_password_hash(hashed_pw, 'testing')

# Database
>>> from flaskblog import db
>>> from flaskblog.models import User
>>> user = User.query.first()
>>> user
>>> user.password
```

```
# install login
pip install flask-login
```