Blabber: 331 Final Project

Burgin Luker, Tiegan Cozzie, Morgan Hanson, Josh Wilcox

Our Application:

A twitter-like application where users can create accounts, post their thoughts, and like other users messages

Work Breakdown:

Burgin	Tiegan	Morgan	Josh
• Back End • Database design • Python model classes • Jinja template creating	• Implemented routes • Managed cookies • Managed DevOps • Registered domain	• Front End • HTML, HTMX, CSS • Global styling • Jinja template creating	• Front End (HTML, HTMX, CSS) • Back End (delete, edit, and update routes)

Technologies Used:

- HTMX
- Redis
- Flask
- Peewee
- Jinja Templates

HTMX

- Open-source front-end JavaScript Library
- Extends HTML with custom attributes
- Hypermedia-driven approach
- Allows for dynamic definition of a web page directly in HTML and CSS without writing additional JS

```
<div style="padding: 20px;"
>
    <div hx-get="/tweets/1"
         hx-trigger="revealed"
         hx-swap="afterend"
         hx-indicator="#spinner-0"
    >
    </div>
    <img id="spinner-0"
         class="my-indicator"
         src="/img/cat.jpeg"
    />
</div>
```

Redis

- Remote Dictionary Server
- Open-source, in-memory data structure store
- Known for its high performance, low latency, and support for various data structures
- Makes data retrieval fast and efficient

```python
def get_tweets(page):
    tweet_data = []
    for tweet in tweets:
        likes = redis_client.hget(f"tweet:{tweet.id}", "likes") or 0

        has_user_liked = redis_client.sismember(f"tweet:{tweet.id}:userLikes", userID)
        heart_icon = "❤️" if has_user_liked else "🤍"

        tweet_data.append({
            "tweet": tweet,
            "likes": int(likes),
            "heart_icon": heart_icon,
        })

    return render_template("tweet.html", tweets=tweet_data, nextPage=page + 1)


@bp.post("/tweets/<int:tweet_id>/like")
def like_tweet(tweet_id):
    userID = request.cookies.get('userId')
    if not userID:
        return redirect('/')

    has_user_liked_tweet = redis_client.sismember(f"tweet:{tweet_id}:userLikes", userID)


    if has_user_liked_tweet:
        redis_client.srem(f"tweet:{tweet_id}:userLikes", userID)
        new_likes = redis_client.hincrby(f"tweet:{tweet_id}", "likes", -1)
        liked = False
    else:
        redis_client.sadd(f"tweet:{tweet_id}:userLikes", userID)
        new_likes = redis_client.hincrby(f"tweet:{tweet_id}", "likes", 1)
        liked = True

    heart_icon = "❤️" if liked else "🤍"
```

Flask

- App is run using Flask
- Micro web framework
- Keeps application core simple and extensible
- Easy to use

main.py > ...

```python
from flask import Flask, redirect
from src.model.base import db
from src.model.user import User
from src.model.tweet import Tweet
from routes.user_routes import bp as users_bp
from routes.tweet_routes import bp as tweets_bp

app = Flask(__name__, static_url_path='', static_folder='static')
app.register_blueprint(users_bp, url_prefix='/users')
app.register_blueprint(tweets_bp, url_prefix='/')
app.debug = True

with db:
    db.create_tables([User, Tweet], safe=True)


@app.before_request
def _db_connect():
    db.connect()


@app.teardown_request
def _db_close(exc):
    if not db.is_closed():
        db.close()


@app.route('/')
def index():
    return redirect("/users/signIn")


if __name__ == '__main__':
    app.run(port=8000)
```

Peewee

- Lightweight Object-Relational Mapping (ORM) library
- Intuitive to use
- Needed a ORM because Flask does not provide one
- Allows access to single database

```python
import datetime

from peewee import *

from src.model.base import BaseModel


class User(BaseModel):
    username = CharField()
    email = CharField()
    userID = CharField(primary_key=True)
    profilePic = CharField()
    joined_date = DateTimeField(default=lambda: datetime.datetime.now().strftime('%b %d, %Y'))

    @classmethod
    def all(cls, user_id, search=None):
        select = User.select()

        if search:
            select = select.where(User.username.ilike('%' + search + '%'))

        ## Remove the current user, no need to see themself
        filtered_out = []
        for user in select:
            if not (user_id == user.userID):
                filtered_out.append(user)

        return filtered_out

    @classmethod
    def find(cls, user_id):
        return User.get_or_none(User.userID == user_id)
```

Jinja

- Python templating engine
- Typically used in partnership with Flask
- Allows the creation of dynamic web pages by combining templates with data
- Straight forward syntax

templates > index.html

```
{% extends "base.html" %}

{% block content %}
    {% include "main.html" %}
{% endblock %}
```

```
<div>
    <h3 style="font-size: 18px; color: #f0f1f3; margin: 0; font-weight: bold;">{{tweet.tweet.user.username}}</h3>
    <span style="color: #4f8a8b; font-size: 15px;">Member Since: {{tweet.tweet.user.joined_date}}</span>
</div>
```

Demo: https://b4bble.com/