Assignment: Build Core Features of a Flask Blogging Platform

Objective: Using your current knowledge of Flask routing, Jinja2 templating, form handling/validation, and static asset integration, implement the core functionality of a simple blogging site.

Requirements: Students should already have:

- A working Flask environment (Flask installed and `flask run` works)

Tasks

1. Routing & Views

Project Structure\ Organize your project directory as follows:

```
project/
├─ app.py                # Application factory and Blueprint registration
├─ routes/               # Route definitions
│  └─ post_routes.py
├─ controllers/          # Business logic (controllers)
│  └─ post_controller.py
├─ models/               # Data models
│  ├─ user.py
│  └─ post.py
├─ templates/            # Jinja2 HTML templates
└─ static/               # CSS, JS, images
```

Route Skeleton (``):

```python
from flask import Blueprint, render_template, request, redirect, url_for, flash
from controllers.post_controller import (
    list_posts, get_post, create_post, update_post, list_drafts
)

post_bp = Blueprint('posts', __name__, url_prefix='/posts')

@post_bp.route('/', methods=['GET'])
def show_posts():
    """
    TODO: Retrieve published posts via list_posts(published=True)
    and render 'posts.html' with the posts list.
    """
    pass
```

```
@post_bp.route('/<int:post_id>', methods=['GET'])
def post_detail(post_id):
    """
    TODO: Fetch a single post by ID using get_post(post_id)
    and render 'post_detail.html' with the post.
    """
    pass

@post_bp.route('/new', methods=['GET', 'POST'])
def new_post():
    """
    TODO: On GET, render 'post_form.html'.
    On POST, collect form data from request.form and call create_post(form).
    """
    pass

@post_bp.route('/<int:post_id>/edit', methods=['GET', 'POST'])
def edit_post(post_id):
    """
    TODO: On GET, fetch post and render 'post_form.html' with post data.
    On POST, collect form data and call update_post(post_id, form).
    """
    pass

@post_bp.route('/drafts', methods=['GET'])
def drafts():
    """
    TODO: Retrieve unpublished posts via list_drafts()
    and render 'posts.html' with the drafts list.
    """
    pass
```

Hint: In `app.py`, import and register the Blueprint:

```
from routes.post_routes import post_bp
app.register_blueprint(post_bp)
```

2. Controllers

Controller Skeleton (``):

```
from models.post import Post
from flask import flash, redirect, url_for
```

```
def list_posts(published):
    """
    TODO: Query Post model filtered by is_published status.
    Return a list of posts.
    """
    pass

def get_post(post_id):
    """
    TODO: Retrieve a post by its ID or abort with 404 if not found.
    """
    pass

def create_post(form):
    """
    TODO: Extract title, body, and is_published from form.
    Validate fields, flash errors, save new Post, flash success, redirect.
    """
    pass

def update_post(post_id, form):
    """
    TODO: Fetch existing post, update attributes from form,
    validate, save changes, flash messages, and redirect.
    """
    pass

def list_drafts():
    """
    TODO: Reuse list_posts to fetch unpublished posts.
    """
    pass
```

3. Jinja2 Templates

1. **Base Template (``)**
2. Common header, footer, and linked CSS/JS.
3. Define a `{% block content %}` for page-specific content.
4. **Child Templates**
5. `posts.html`: list view, loop with `{% for post in posts %}`.
6. `post_detail.html`: show full post details.

7. `post_form.html`: form for creating/editing posts.

4. Form Handling & Validation

- Use `request.form` to access fields: `title`, `body`, `is_published`.
- Validate that `title` and `body` are not empty.
- On errors: `flash('Error message', 'error')` and re-render form.
- On success: create/update the Post, `flash('Success', 'success')`, redirect.

5. Draft Workflow

- In the Post model, include a Boolean `is_published` (default `False`).
- In `post_form.html`, add a checkbox labeled **Publish now** bound to `is_published`.
- `/posts` shows only published posts; `/posts/drafts` shows drafts.

6. Static Assets (CSS & JS)

- Place `static/css/style.css` and include in `base.html` via `{{ url_for('static', filename='css/style.css') }}`.
- Place `static/js/script.js` and include at bottom of `base.html`.
- Add a simple JS feature, e.g. live character count of the title field.

7. Flash Messages

- Configure `flash()` for error and success messages.
- In templates, render via:

```
{% with messages = get_flashed_messages(with_categories=True) %}
  {% if messages %}
    <ul class="flashes">
      {% for category, msg in messages %}
        <li class="{{ category }}">{{ msg }}</li>
      {% endfor %}
    </ul>
  {% endif %}
{% endwith %}
```

Deliverables

- **Code**: GitHub repo with your Flask app.
- **Templates**: All HTML files under `templates/`.
- **Static Files**: CSS and JS under `static/`.
- **README**: Instructions to run the app and list implemented features.

Feel free to ask questions during class. Good luck!