Assignment 2: Database & ORM Integration in Flask Blog

Builds on the Mini Project Brief from Assignment 1 filecite turn0file1 .

Objective: Implement the database layer and ORM features of your Flask blogging platform: model definitions, relationships, migrations, CRUD operations via SQLAlchemy, plus validation and mixins as covered in class filecite turn0file0 .

Prerequisites:

- Completion of Assignment 1 (routing, templates, static assets, basic controllers)
- Flask-SQLAlchemy, Flask-Migrate, Flask-WTF installed

Tasks

1. Define SQLAlchemy Models & Mixins

1. **Declarative Base & Blueprint**
2. Create `models/base.py` with:

```
from sqlalchemy.orm import declarative_base
Base = declarative_base()
```

3. **TimestampMixin**
4. In `models/mixins.py`, define:

```
class TimestampMixin:
    """Adds created_at/updated_at auto-timestamp columns."""
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime,
                        default=datetime.utcnow,
                        onupdate=datetime.utcnow)
```

5. **User, Post, Tag & Association**
6. In `models/user.py`, `models/post.py`, and `models/tag.py`, subclass `Base` and/or `TimestampMixin`.
7. **User**: `id`, `username`, `_email` column + property setter for email validation (raise `ValueError` if invalid).
8. **Post**: `id`, `title`, `body`, `is_published`, `author_id` (ForeignKey), `tags` relationship.
9. **Tag**: `id`, `name`, `posts` relationship.
10. **Association Table**: define `post_tags = Table('post_tags', Base.metadata, ...)` in `models/tag.py` or a separate `models/association.py`.

2. Establish Relationships

- **One-to-Many**: in User and Post, use `relationship(..., back_populates=...)`.
- **Many-to-Many**: in Post and Tag, use `secondary=post_tags`, `back_populates` and ensure attribute names match on both sides.

3. Add Model Methods & Mixins

1. **Post.publish()/unpublish()**
2. Implement methods to toggle `is_published`, commit on success, raise if invalid.
3. **Override** `` on each model for debug-friendly output.
4. **Apply** `` ` to models for automatic `` created_at `/` updated_at`.

4. Configure & Run Migrations

1. **Flask-Migrate Setup** in your factory (`app.py`):

```
from flask_migrate import Migrate
migrate = Migrate(app, db)
```

2. **Initialize & Generate**:

```
flask db init      # create migrations folder
flask db migrate   # auto-detect model changes
flask db upgrade   # apply to database
```

3. **Verify** with `sqlite3 blog.db .tables` or via DB Browser.

5. Implement Controllers & Routes

- **Controllers**: in `controllers/post_controller.py`, implement:
- `list_posts(published=True)`, `get_post(id)`, `create_post(form)`, `update_post(id, form)`, `delete_post(id)`, `list_drafts()`.
- **Blueprint**: in `routes/post_routes.py`, define GET/POST routes, split via `@app.get` and `@app.post` or combined `methods=[...]`.
- **Flash**: use `flash(..., 'error'/'success')` and `get_flashed_messages(with_categories=True)` in templates.

6. Build & Validate Forms

- **WTForms**: in `forms/post_form.py`, create `PostForm` with `StringField`, `TextAreaField`, `BooleanField`, and appropriate validators (`DataRequired`, `Length`).
- **EmailForm**: optionally, create a simple form to test your `User.email` setter.
- **Views**: in routes, replace manual validation with `form.validate_on_submit()`, use `form.field.data` to populate models.

7. Update Templates for Database Features

- **posts.html**: loop over `posts`, display `post.tags` and `post.author.username`.
- **post_detail.html**: show full `post.body`, list of tags, author info, created_at
- **post_form.html**: render `{{ form.title }}`, `{{ form.body }}`, `{{ form.is_published }}` with WTForms helpers.

Deliverables

- **GitHub Repository**: push all model, migration, controller, route, form, and template files.
- **README**: document how to set up the database, run migrations, and test publishing/unpublishing.
- **Demo**: a short screencast or written walkthrough showing:
- Creating a draft post
- Publishing a post via `publish()` method
- Assigning tags and querying by tag
- Email validation preventing invalid addresses

Let me know if you need any further details or clarifications!