Support Content Creation in Your Flask App

12 min to complete · By Brandon Gigous

Contents

Many social media websites have a way to submit some form of content to the site. (The ones that don't must be a strange place.) The website server is responsible for sending responses to all users who can see that content whenever they request it. In this section, you will implement the main feature of your app: to allow users to submit information about their musical creations. This lesson, in particular, will show you how your users can create content and how you can represent this content in the database.

Composing Website Content

In social media, the common term for a content submittal is called a "post." You can make a *post* on your Facebook wall to share something with your audience. You can make a *post* on Reddit to complain about melts. You can also *post* (the verb) your newest song on SoundCloud. For you and your app, it can be anything you want! But for the purposes of this course and to make it more simple for you, a *post* will be called a **composition**.

That's because the "posts," in this case, are users sharing their music with others, which can be singles, EPs, or albums. Each composition also has a title and a description, where users can describe their musical creation and post links to other places 1) where others can listen to it, and 2) to encourage others to support them by buying them. That's the idea anyway; feel free to get creative with it!

Coding the Composition Model

Now onto how to *program* this composition thingy. Well, the first thing to do would be to represent them in the database; otherwise, they might as well not exist at all. If you haven't noticed before, databases are often the first place to start adding new functionality, especially when it comes to storing data.

Ready to type some more code into `models.py`? Here's the new `Composition` database model, along with an addition to `User` to allow users to have their *own*

compositions:

```
class User(UserMixin, db.Model):
    # ...
    compositions = db.relationship('Composition',
        backref='artist', lazy='dynamic')


class ReleaseType:
    SINGLE = 1
    EXTENDED_PLAY = 2
    ALBUM = 3


class Composition(db.Model):
    __tablename__ = 'compositions'
    id = db.Column(db.Integer, primary_key=True)
    release_type = db.Column(db.Integer)
    title = db.Column(db.String(64))
    description = db.Column(db.Text)
    timestamp = db.Column(db.DateTime,
        index=True, default=datetime.utcnow)
    artist_id = db.Column(db.Integer, db.ForeignKey('users.id'))
```

As mentioned, each composition has a *release type* (single, EP, or album), a title, and a description. The `release_type` is represented as an integer, similar to how you set permissions in the User Roles section. Just like `Permissions`, you can define a `ReleaseType` class that defines constants for each type. The `title` is a simple `db.String` type column, and `description` is a `db.Text` column just like the `bio` for the `User` model.

A couple more things: a `timestamp` column captures the time when the composition was created, and an `artist_id` column contains the ID of the user who submitted the composition. Last but not least, a one-to-many `relationship` is formed between the `User` model and the `Composition` model. The `backref` is called `artist` so that the `Composition` instance can grab the exact "maker" of that composition.

Create a Composition Form

The next step? The form to make compositions, of course. Here it is in its natural habitat in `app/main/forms.py` :

```
class CompositionForm(FlaskForm):
    release_type = SelectField("Release Type", coerce=int, default=R
    title = StringField("Title", validators=[DataRequired()])
    description = TextAreaField("Tell us about your composition")
    submit = SubmitField("Submit")

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.release_type.choices = [
            (ReleaseType.SINGLE, 'Single'),
```

```
            (ReleaseType.EXTENDED_PLAY, 'EP'),
            (ReleaseType.ALBUM, 'Album')]
```

The `CompositionForm` isn't much different from other forms you've seen. An important thing to note, which you should also be familiar with, is that the `release_type` `SelectField` must have `coerce=int` in order to play nice with the database. Your `ReleaseType` s are integers, after all. The other half of making them play nice is to initialize the `SelectField` in the constructor. Lastly, both the release type and title are required to submit the form.

Change The Index Page

Instead of burying the `CompositionForm` somewhere in the user interface, displaying it on the main page would be more useful. As such, you can replace the currently used `NameForm` (if it's still in your `index()` view function) with the `CompositionForm` . So, in `app/main/views.py` , put in the new form:

```
@main.route('/', methods=['GET', 'POST'])
def index():
    form = CompositionForm()
    if current_user.can(Permission.PUBLISH)
            and form.validate_on_submit():
        composition = Composition(
            release_type=form.release_type.data,
            title=form.title.data,
            description=form.description.data,
            artist=current_user._get_current_object())
        db.session.add(composition)
        db.session.commit()
        return redirect(url_for('.index'))
    compositions = Composition.query.order_by(
        Composition.timestamp.desc()).all()
    return render_template(
        'index.html',
        form=form,
        compositions=compositions
    )
```

What's going on here? Hmm, well, it's the usual form handling stuff in a view function, but wait a sec... It's that `_get_current_object()` method showing its face again?! Remember this in `send_email()`? It was for the Flask `app` object because you were trying to speed up the email sending capabilities of your app. This time, you'll need it for the `current_user`.

You see, `current_user` is just like `current_app` in that it's a *proxy* for the current user, and not the actual `User` object that represents the user. By calling `_get_current_object()` on `current_user`, you are effectively taking a guilt-free shortcut.

One, you don't have to get the `current_user`'s ID, then query the database for the actual `User`, then give that `User` to the new `Composition`. Two, `current_user` already knows what `User` object you want because its big secret is that it thinly wraps the actual `User` object. You don't want to use `_get_current_object()` all the time as it isn't necessary, but in this case it works great.

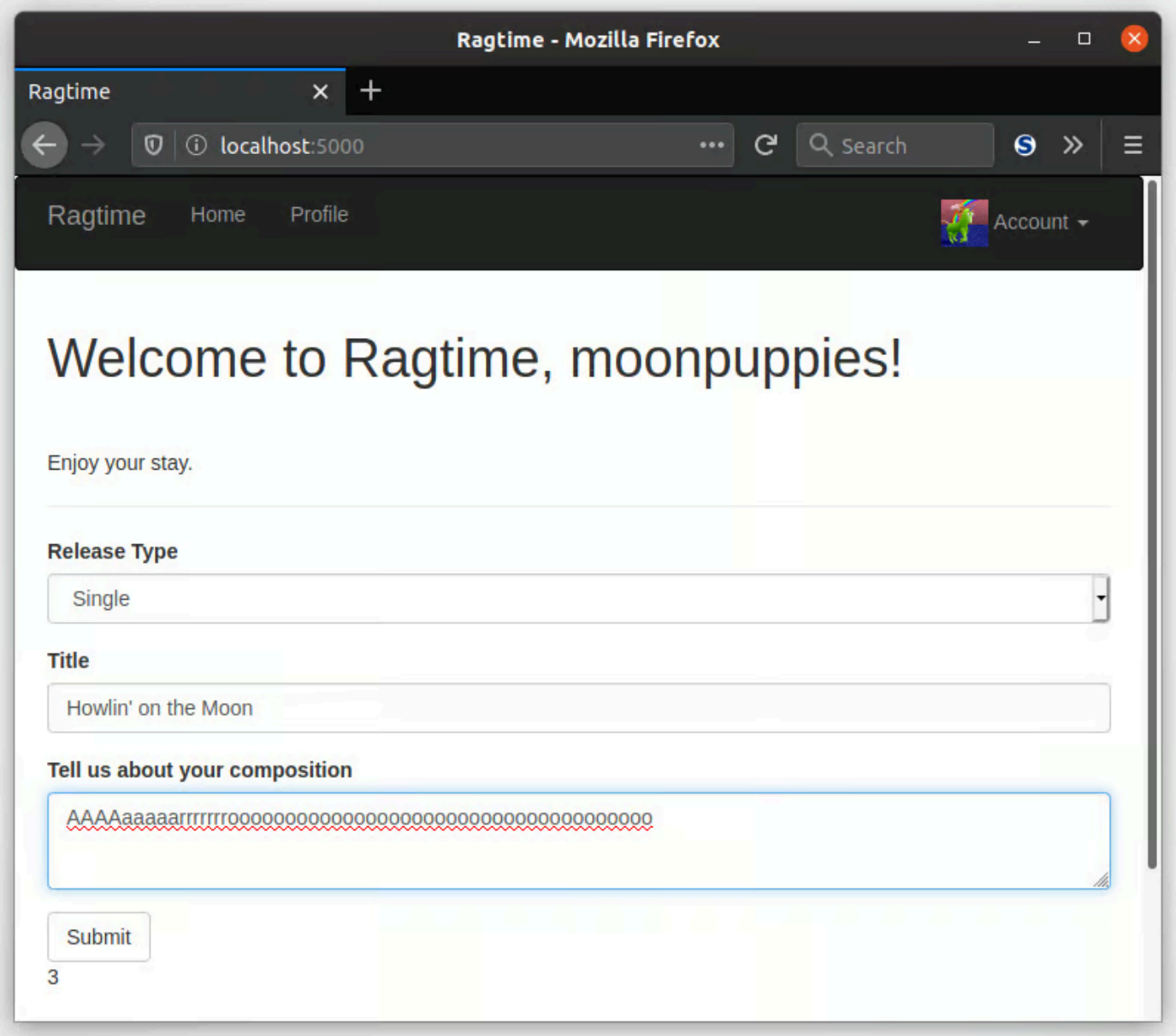
"Cool, thanks for that explanation. Now, what's going on with passing all compositions to the `index.html` template?" Ah, well, just like Facebook, Reddit, or SoundCloud, all these platforms show you content on their homepage. It's the same sort of thing, but the content you show users are the compositions. The compositions are ordered by timestamp, so the newer ones show up first. The reason it is commented out will be clear in a minute...

Make Sure It Works

Before you go about accommodating any compositions in your `index.html` template, you can first make sure your compositions can be created from your form. Without needing to render the compositions yet, you can test that a `Composition` can be created from your form by putting something like this somewhere in your template:

```
{{ compositions|length }}
```

This will show you how many compositions are currently in the database. It's only temporary to ensure that everything thus far works. It should all look something like this:



Once you've successfully proven to yourself that you can create compositions, you're ready to keep that momentum going. In the next lesson, you'll get your hands dirty with cleaning up your index template, including the display of your new content!

Get Mentorship

Summary: How to Let Users Create Content on Your Python + Flask App

- The first way to represent new content is in the database

```
class User(UserMixin, db.Model):
    # ...
    compositions = db.relationship('Composition',
        backref='artist', lazy='dynamic')


class ReleaseType:
    SINGLE = 1
    EXTENDED_PLAY = 2
    ALBUM = 3


class Composition(db.Model):
    __tablename__ = 'compositions'
    id = db.Column(db.Integer, primary_key=True)
    release_type = db.Column(db.Integer)
    title = db.Column(db.String(64))
    description = db.Column(db.Text)
    timestamp = db.Column(db.DateTime,
```

```
                index=True, default=datetime.utcnow)
        artist_id = db.Column(db.Integer, db.ForeignKey('users.id'))
```

- The next thing to do is create a form for users to submit their own content!

Previous Next → Lab: Make a Composition Index Page