WTForms

Springboard

Download Demo Code

Flask Forms

You can make forms yourself!

- Write the HTML (including labels, etc)
- Write server-side validating code for each field
- Add logic for form for showing validation messages
- Add protection against security attacks

This is tedious.

WTForms

WTForms is a Python library providing:

- Validation
- HTML production
- Security

Flask-WTF

Flask-WTF is built on top of that, and adds integration with Flask (get data from request, etc)

Install

```
(env) $ pip install flask-wtf
```

Basic Example

Defining the Form Class

demo/forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, FloatField
class AddSnackForm(FlaskForm):
    """Form for adding snacks."""

    name = StringField("Snack Name")
    price = FloatField("Price in USD")
```

The Form Route Handler

demo/app.py

```
from forms import AddSnackForm
```

demo/app.py

```
@app.route("/add", methods=["GET", "POST"])
def add_snack():
    """Snack add form; handle adding."""

    form = AddSnackForm()

    if form.validate_on_submit():
        name = form.name.data
        price = form.price.data
        flash(f"Added {name} at {price}")
        return redirect("/add")

    else:
        return render_template(
            "snack_add_form.html", form=form)
```

This validates submitted form or passes instance of form to template.

Add-Form View

demo/templates/snack_add_form.html

```
<form id="snack-add-form" method="POST">
  {{ form.hidden_tag() }} <!--add type=hidden form fields -->

  {% for field in form
       if field.widget.input_type != 'hidden' %}

    <p>
      {{ field.label }}
      {{ field }}

      {% for error in field.errors %}
        {{ error }}
      {% endfor %}
    </p>

  {% endfor %}

  <button type="submit">Submit</button>
</form>
```

Models vs Forms

- SQLAlchemy provides **model**: class for logical object
- WTForm provides **form class**
- A single model may have different forms
  - Not all fields on add form might appear on edit form
  - Different validation might apply on add/edit
  - Different kinds of users (public v admin) have different fields

You'll often take the result of a form and create/edit an SQLAlchemy object.

Field Types

BooleanField
Normally appears as a checkbox

DateField* / *DateTimeField
Date or Date & Time

IntegerField* / *FloatField
Numeric types

StringField* / *TextAreaField
Single line of text / larger text area

Selection From Choices

RadioField
Series of radio buttons from ***choices***

SelectField
Drop-down menu from ***choices***

SelectMultipleField
Multi-select box from ***choices***

```
weather = SelectField('Weather',
  choices=[('rain', 'Rain'), ('nice', 'Nice Weather')]
)
```

To convert result to integer:

```
priority = SelectField('Priority Code',
  choices=[(1, 'High'), (2, 'Low')],
  coerce=int
)
```

Can set dynamic choices:

forms.py

```
class AddFriendForm(FlaskForm):
    """Form to pick a friend."""

    friend = SelectField("Friend", coerce=int)
```

app.py

```
@app.route("/get-friend")
def handle_friend_form():
    """Handle the add-friend form."""

    form = AddFriendForm()

    # get current list of users
    users = [(u.id, u.name) for u in User.query.all()]

    # dynamically set friend choices
    form.friend.choices = users
```

Validation

WTForm provides "validators":

demo/forms.py

```
from wtforms.validators import InputRequired, Optional, Email
```

demo/forms.py

```
class UserForm(FlaskForm):
    """Form for adding/editing friend."""

    name = StringField("Name",
                       validators=[InputRequired()])
    email = StringField("Email Address",
                        validators=[Optional(), Email()])
```

See https://wtforms.readthedocs.io/en/2.3.x/validators/

Update Forms

demo/app.py

```
@app.route("/users/<int:uid>/edit", methods=["GET", "POST"])
def edit_user(uid):
    """Show user edit form and handle edit."""

    user = User.query.get_or_404(uid)
    form = UserForm(obj=user)

    if form.validate_on_submit():
        user.name = form.name.data
        user.email = form.email.data
        db.session.commit()
        flash(f"User {uid} updated!")
        return redirect(f"/users/{uid}/edit")

    else:
        return render_template("user_form.html", form=form)
```

Passing `obj=data-obj` provides form with defaults from object

CSRF Security

Cross-Site Request Forgery

A form on any site can submit to any other site!

```
<form action="http://yourbank.com/transfer" method="POST">
  <input type="hidden" name="from" value="your-acct">
  <input type="hidden" name="to" value="my-acct">
  <input type="hidden" name="amt" value="$1,000,000">
  <button type=submit">I Love Kittens!</button>
</form>
```

Therefore, most sites use a "CSRF Token":

- This is generated by the server when a form is shown
- It is included in the HTML of the form
- It is checked by the server on form submission

Flask-WTF uses CSRF out-of-the-box:

- All forms include a hidden CSRF field
- The ***validate_on_submit*** method checks for this

Testing

For tests to work, need to disable CSRF checking in tests:

demo/tests.py

```
app.config['WTF_CSRF_ENABLED'] = False
```

demo/tests.py

```
class SnackViewsTestCase(TestCase):
    """Tests for views for Snacks."""

    def test_snack_add_form(self):
        with app.test_client() as client:
            resp = client.get("/add")
            html = resp.get_data(as_text=True)

            self.assertEqual(resp.status_code, 200)
            self.assertIn('<form id="snack-add-form"', html)

    def test_snack_add(self):
        with app.test_client() as client:
            d = {"name": "Test2", "price": 2}
            resp = client.post("/add", data=d, follow_redirects=True)
            html = resp.get_data(as_text=True)

            self.assertEqual(resp.status_code, 200)
            self.assertIn("Added Test2 at 2", html)
```

Best Practices

- Make distinct add/edit forms, if sensible
- Add lots of form validation, if appropriate
- All non-GET routes return ***redirect*** (not ***render_template***) on success