Form Validation with WTForms

When you have to work with form data submitted by a browser view, code quickly becomes very hard to read. There are libraries out there designed to make this process easier to manage. One of them is WTForms which we will handle here. If you find yourself in the situation of having many forms, you might want to give it a try.

When you are working with WTForms you have to define your forms as classes first. I recommend breaking up the application into multiple modules (Larger Applications) for that and adding a separate module for the forms.

Getting the most out of WTForms with an Extension:

The Flask-WTF extension expands on this pattern and adds a few little helpers that make working with forms and Flask more fun. You can get it from PyPI.

The Forms

This is an example form for a typical registration page:

```
from wtforms import Form, BooleanField, StringField, PasswordField, val

class RegistrationForm(Form):
    username = StringField('Username', [validators.Length(min=4, max=25
    email = StringField('Email Address', [validators.Length(min=6, max=
    password = PasswordField('New Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords must match')
    ])
    confirm = PasswordField('Repeat Password')
    accept_tos = BooleanField('I accept the TOS', [validators.DataRequi
```

In the View

In the view function, the usage of this form looks like this:

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    form = RegistrationForm(request.form)
    if request.method == 'POST' and form.validate():
```

```
        user = User(form.username.data, form.email.data,
                    form.password.data)
        db_session.add(user)
        flash('Thanks for registering')
        return redirect(url_for('login'))
    return render_template('register.html', form=form)
```

Notice we're implying that the view is using SQLAlchemy here (SQLAlchemy in Flask), but that's not a requirement, of course. Adapt the code as necessary.

Things to remember:

1. create the form from the request **`form`** value if the data is submitted via the HTTP `POST` method and **`args`** if the data is submitted as `GET`.
2. to validate the data, call the **`validate()`** method, which will return `True` if the data validates, `False` otherwise.
3. to access individual values from the form, access *form.<NAME>.data*.

Forms in Templates

Now to the template side. When you pass the form to the templates, you can easily render them there. Look at the following example template to see how easy this is. WTForms does half the form generation for us already. To make it even nicer, we can write a macro that renders a field with label and a list of errors if there are any.

Here's an example `_formhelpers.html` template with such a macro:

```
{% macro render_field(field) %}
  <dt>{{ field.label }}
  <dd>{{ field(**kwargs)|safe }}
  {% if field.errors %}
    <ul class=errors>
    {% for error in field.errors %}
      <li>{{ error }}</li>
    {% endfor %}
    </ul>
  {% endif %}
  </dd>
{% endmacro %}
```

This macro accepts a couple of keyword arguments that are forwarded to WTForm's field function, which renders the field for us. The keyword arguments will be inserted as HTML attributes. So, for example, you can call `render_field(form.username, class='username')` to add a class to the input element. Note that WTForms returns

standard Python unicode strings, so we have to tell Jinja2 that this data is already HTML-escaped with the `|safe` filter.

Here is the `register.html` template for the function we used above, which takes advantage of the `_formhelpers.html` template:

```
{% from "_formhelpers.html" import render_field %}
<form method=post>
  <dl>
    {{ render_field(form.username) }}
    {{ render_field(form.email) }}
    {{ render_field(form.password) }}
    {{ render_field(form.confirm) }}
    {{ render_field(form.accept_tos) }}
  </dl>
  <p><input type=submit value=Register>
</form>
```

For more information about WTForms, head over to the WTForms website.