pyramid$_{u}niformDocumentation$

Release 0.2

Scott Torborg

March 01, 2014

Contents

Scott Torborg - Cart Logic

Form rendering and validation for Pyramid. Doesn't render HTML itself, so you have full control over form markup.

Heavily inspired by the `pyramid_simpleform` package, rewritten to work with `WebHelpers2` and Python 3 and fix some issues with the original.

CHAPTER 1

Contents

1.1 Quick Start

1.1.1 Install

Install with pip:

```
$ pip install pyramid_uniform
```

1.1.2 Use in a Pyramid App

No `config.include()` directive or setting modifications are required to use `pyramid_uniform`. Instead, two primary classes are used directly: `Form` and `FormRenderer`.

You can use `Form` alone to validate and process a form, with a FormEncode Schema.:

```
from formencode import Schema, validators
from pyramid_uniform import Form


class MySchema(Schema):
    name = validators.String()
    size = validators.Int()
    published = validators.Bool()


def myview(request):
    obj = ...

    # Initialize the form
    form = Form(request, MySchema)

    # Check for schema validity.
    if form.validate():

        # Update the attributes of obj based on validated form fields.
        form.bind(obj)
```

A common pattern is to use the same view (or handler action) to show a form and process the result. To do this, use a `FormRenderer` class to wrap the `Form` instance for presentation.:

```
from formencode import Schema
from pyramid_uniform import Form, FormRenderer


class MySchema(Schema):
    name = validators.String()
    size = validators.Int()
    published = validators.Bool()


def myview(request):
    obj = get_thing(request)

    # Initialize the form
    form = Form(request, MySchema)

    # Check for schema validity.
    if form.validate():

        # Update the attributes of obj based on validated form fields.
        form.bind(obj)
        return HTTPFound(...)

    # Form data is not present or not valid, so show the form.
    renderer = FormRenderer(form)
    return {'renderer': renderer, 'obj': obj}
```

To use `renderer` in a template, call methods on it to generate HTML form tags:

```
<h1>Edit ${obj}</h1>

${renderer.begin()}
  ${renderer.text('name', obj.name)}
  ${renderer.select('size', obj.size, range(10))}
  ${renderer.checkbox('published', checked=obj.published)}
${renderer.end()}
```

Extensive customization of the validation and rendering behavior is possible. For details, see the API documentation.

1.2 Recipes

1.2.1 Validating Non-User Input

It's often important to validate input that is not entered directly by a user, but is still untrusted. For example, a client process running on a remote machine may construct a URL algorithmically.

In this case, we don't want to deal with the full 'plumbing' of form error rendering, we just want to make sure the input is safe. `pyramid_uniform.Form.assert_valid()` can be used for this purpose.

```
from pyramid_uniform import Form

Form(request, MySchema).assert_valid()
```

1.2.2 Skipping CSRF Protection

CSRF protection is always on by default. To skip it, pass `skip_csrf=True` to any relevant methods.

```
form = Form(request, MySchema)
if form.validate(skip_csrf=True):
    pass
```

1.3 Logging

You may wish to maintain a separate log of form validation errors. Built-in support is included using the `pyramid_uniform.validate` logging key.

That logger will emit messages like:

```
Validation failure on http://example.com/some-url from 1.2.3.4 [Mozilla 5.0]
Params:
{'name': 'Something valid', 'integer_value': 'twelve'}
Errors:
{'integer_value': 'Must contain an integer'}
```

Loggers are propagated, so you can use `pyramid_uniform` directly.

1.4 API Reference

1.4.1 Forms

class pyramid_uniform.**State**(*request*)
A relatively simple state object for the schema being validated to use, with a reference to the request being validated.

class pyramid_uniform.**Form**(*request*, *schema*, *method='POST'*, *skip_csrf=False*)
Represents a set of fields (GET or POST parameters) to be validated using a `FormEncode` schema.

Parameters

- **request** (`WebOb.Request`) – The web request containing data to be validated
- **schema** (`FormEncode.Schema`) – The schema to validate against
- **method** (*str*) – HTTP request method that this form expects: GET or POST

assert_valid(***kw*)
Assert that this form is valid, the request method is appropriate, and the CSRF check passes (unless it is explicitly skipped).

bind(*obj*)
Bind the data from this form to an object: that is, try to set attributes on the client corresponding to the keys present in the validated data. The object can be a template object, SQLAlchemy object, etc.

If any value in the data is a list or another dictionary, recurse with that key.

Private attributes, which is anything that is prefixed with _, will be skipped.

Once done, return the object.

data
Once the form has been validated, contains the results of that validation as a dict.

Raises FormNotValidated if the form has not yet been validated

errors_for(*field*)
Return a list of errors for the given field.

is_error(*field*)
Check if the given field has any validation errors.

method_allowed
Is the method that was used to submit this form allowed?

If this form doesn't have a request method set (i.e., if it was explicitly set to `None`), any method is valid. Otherwise, the method of the form submission must match the method required by this form.

validate(*skip_csrf=False*, *assert_valid=False*)
Validate a form submission.

When `assert_valid` is `False` (the default), a bool will be returned to indicate whether the form was valid. (Note: this isn't strictly true–a missing or bad CSRF token will result in a immediate `400 Bad Request` response).

When `assert_valid` is `True`, certain conditions will be asserted. When an assertion fails, an `AssertionError` will be raised.

Parameters

- **skip_csrf** (*bool*) – if True, bypass the CSRF check
- **assert_valid** (*bool*) – if True, assert validity instead of returning status

validate_csrf(*params=None*)
Validate that the CSRF token is correct.

1.4.2 Rendering

class `pyramid_uniform.`**Renderer**(*data*, *errors*, *name_prefix=''*, *id_prefix=''*)

checkbox(*name*, *value='1'*, *checked=False*, *label=None*, *id=<class 'webhelpers2.misc.NotGiven'>*, ***attrs*)
Return a checkbox tag.

errorlist(*name*)
Return a list of errors for the given field as a `ul` tag.

errors_for(*name*)
Return a list of errors for the given field.

file(*name*, *value=None*, *id=<class 'webhelpers2.misc.NotGiven'>*, ***attrs*)
Return a `file` input tag.

hidden(*name*, *value=None*, *id=<class 'webhelpers2.misc.NotGiven'>*, ***attrs*)
Return a `hidden` input tag.

is_error(*name*)
Check if the given field has any validation errors.

password(*name*, *value=None*, *id=<class 'webhelpers2.misc.NotGiven'>*, ***attrs*)
Return a password input tag.

radio(*name*, *value=None*, *checked=False*, *label=None*, ***attrs*)
Return a radio button tag.

select (*name*, *selected_values*, *options*, *id=<class 'webhelpers2.misc.NotGiven'>*, ***attrs*)
Return a select tag.

submit (*name=None*, *value=None*, *id=<class 'webhelpers2.misc.NotGiven'>*, ***attrs*)
Return a submit button tag.

text (*name*, *value=None*, *id=<class 'webhelpers2.misc.NotGiven'>*, ***attrs*)
Return a `text` input tag.

textarea (*name*, *content=''*, *id=<class 'webhelpers2.misc.NotGiven'>*, ***attrs*)
Return a textarea tag.

value (*name*, *default=None*)
Return the value for the given field as supplied to the form.

class `pyramid_uniform.`**`FormRenderer`** (*form*, *csrf_field='_authentication_token'*, *name_prefix=''*, *id_prefix=''*)
Bases: `pyramid_uniform.Renderer`

Wraps a form to provide HTML rendering capability.

begin (*url=None*, *skip_csrf=False*, ***attrs*)
Return a `form` opening tag.

csrf (*name=None*)
Return a bare hidden input field containing the CSRF token and param name.

csrf_token (*name=None*)
Return a hidden field containing the CSRF token, wrapped in an invisible div, so that it is valid HTML regardless of context.

end()
Return a `form` closing tag.

1.4.3 Exceptions

exception `pyramid_uniform.`**`FormError`** (**args*, ***kw*)
Bases: `exceptions.Exception`

Superclass for form-related errors.

exception `pyramid_uniform.`**`FormInvalid`** (**args*, ***kw*)
Bases: `pyramid_uniform.FormError`

Raised when form data is used but the form is not valid.

exception `pyramid_uniform.`**`FormNotValidated`** (**args*, ***kw*)
Bases: `pyramid_uniform.FormError`

Raised when form data is used before form has been validated: for example, when `Form.bind()` is called.

1.5 Contributing

Patches and suggestions are strongly encouraged! GitHub pull requests are preferred, but other mechanisms of feedback are welcome.

`pyramid_uniform` has a comprehensive test suite with 100% line and branch coverage, as reported by the excellent `coverage` module. To run the tests, simply run in the top level of the repo:

```
$ tox
```

This will also ensure that the Sphinx documentation builds correctly, and that there are no PEP8 or Pyflakes warnings in the codebase.

Any pull requests should preserve all of these things.

CHAPTER 2

Indices and Tables

- *genindex*
- *modindex*