pyramid$_{u}niformDocumentation$

Release 0.2

Scott Torborg

March 01, 2014

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 API Reference

class pyramid_uniform.**Form**(*request*, *schema*, *method='POST'*)

assert_valid(***kw*)

bind(*obj*)

errors_for(*field*)

is_error(*field*)

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 `assert''ed. When an assertion fails, the resulting ''AssertionError` will cause an internal server error, which will in turn cause an error email to be sent out.

validate_csrf(*params=None*)

exception pyramid_uniform.**FormError**(**args*, ***kw*)
Superclass for form-related errors.

exception pyramid_uniform.**FormInvalid**(**args*, ***kw*)
Raised when form data is used but the form is not valid.

message = 'Form is invalid'

exception pyramid_uniform.**FormNotValidatedError**(**args*, ***kw*)
Raised when form data is used before form has been validated: for example, when `form.bind()` is called.

message = 'Form has not been validated; call validate() first'

class pyramid_uniform.**FormRenderer**(*form*, *csrf_field='_authentication_token'*, *id_prefix=None*)

begin(*url=None*, ***attrs*)

csrf(*name=None*)

csrf_token(*name=None*)

end()

class pyramid_uniform.**Renderer**(*data*, *errors*, *id_prefix=None*)

checkbox(*name*, *value='1'*, *checked=False*, *label=None*, *id=None*, ***attrs*)

errorlist(*name*)

errors_for(*name*)

file(*name*, *value=None*, *id=None*, ***attrs*)

hidden(*name*, *value=None*, *id=None*, ***attrs*)

is_error(*name*)

password(*name*, *value=None*, *id=None*, ***attrs*)

radio(*name*, *value=None*, *checked=False*, *label=None*, ***attrs*)

select(*name*, *selected_values*, *options*, *id=None*, ***attrs*)

submit(*name*, *value=None*, *id=None*, ***attrs*)

text(*name*, *value=None*, *id=None*, ***attrs*)

textarea(*name*, *content=''*, *id=None*, ***attrs*)

value(*name*, *default=None*)

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

1.3 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*

p