wtforms-json Documentation

Release 0.2.10

Konsta Vesterinen

August 10, 2015

Contents

WTForms-JSON is a WTForms extension for JSON data handling.

CHAPTER 1

What does it do?

- Adds support for booleans (WTForms doesn't know how to handle *False* boolean values)
- Adds support for `None` type `FormField` values
- Adds support for `None` type `Field` values
- Support for patch data requests with `patch_data` `Form` property
- Function for converting JSON data into dict that WTForms understands (`flatten_json()` function)

CHAPTER 2

Quickstart

In order to start using WTForms-JSON, you need to first initialize the extension. This monkey patches some classes and methods within WTForms and adds JSON handling support:

```
import wtforms_json

wtforms_json.init()
```

CHAPTER 3

First Example

After the extension has been initialized we can create an ordinary WTForms form. Notice how we are initalizing the form using `from_json()` class method:

```
from wtforms import Form
from wtforms.fields import BooleanField, TextField


class LocationForm(Form):
    name = TextField()
    address = TextField()


class EventForm(Form):
    name = TextField()
    is_public = BooleanField()


json = {
    'name': 'First Event',
    'location': {'name': 'some location'},
}

form = EventForm.from_json(json)
```

Here `from_json()` takes exactly the same parameters as `Form.__init__`.

If you want WTForms-JSON to throw errors when unknown json keys are encountered just pass `skip_unknown_keys=False` to `from_json()`:

```
json = {
    'some_unknown_key': 'some_value'
}

# Throws exception
form = EventForm.from_json(json, skip_unknown_keys=False)
```

CHAPTER 4

Using patch_data

The way forms usually work on websites is that they post all the data within their fields. When working with APIs and JSON data it makes sense to not actually post all the data that hasn't changed – rather make so called patch request which only post the data that the user actually changed.

You can get access to the patch data (data that only contains the actually set fields or fields that have defaults and are required) with form's `patch_data` property.

Now lets use the forms from the previous example:

```
>>> form.data
{
    'name': 'First Event',
    'is_public': False,
    'location': {
        'name': 'some location',
        'address': None
    }
}
>>> form.patch_data
{
    'name': 'First Event',
    'location': {
        'name': 'some location'
    }
}
```

CHAPTER 5

Internals

WTForm uses special flattened dict as a data parameter for forms. WTForms-JSON provides a method for converting JSON into this format.

Note this is done automatically internally:

```
from wtforms import Form
from wtforms.fields import FormField, StringField
from wtforms_json import flatten_json


class FormB(Form):
    b = TextField('B')

class FormA(Form):
    a = FormField(FormB)


assert flatten_json({'a': {'b': 'c'}}) == {'a-b': 'c'}
```

This neat little function understands nested lists and dicts as well:

```
from wtforms_json import flatten_json


class FormC(Form):
    c = IntegerField('C')


class FormB(Form):
    b = FormField(FormC)

class FormA(Form):
    a = FieldList(FormField(FormB))


deep_dict = {
    'a': [{'b': {'c': 1}}]
}

assert flatten_json(deep_dict) == {'a-0-b-c': 1}
```