Django Dynamic Fields Documentation

Release 0.1.0

James Pic

Mar 24, 2017

Contents

CHAPTER 1

Toc

Design documentation for django-dynamic-field

Classes

The way this is going, we're going to have an equivalent JS class for each Python class. For example, using the RemoveField Action would remove a field from the form object right before server side validation, but it should also remove it in the browser (or rather, hide it, so that it can be re-added later).

While classes have equivalents on the client and the server, they are of course not quite the same. On the client, an Action can be un-applied if a user changes their mind and makes a Condition that did pass before fail now. On the server, we don't need to un-apply actions, because data is completely out of the hands of user at this point.

Form Form objects contain the list of Rules, one per field, it's responsible for executing them. In Python it's represented by the FormMixin which should work on any Form class.

Rule Rule objects contain all the Actions for one field, and is responsible for executing them. We could have named this Field too, but it would have been confusing with actual form Field objects.

Action An action may change user data or even the form definition, but it also contains a list of Conditions it should test before doing anything. On the client it can un-apply itself.

Condition Condition objects should just check something in the form, for example that a particular field has or hasn't a value. When it passes, this should prevent the Action from being applied on the server, and un-apply it dynamically on the client.

Form

Python

Form classes should execute each Action before Django's `full_clean()` method is called, because Actions may hack data and form fields. It must be done in `full_clean()` because Actions may also raise ValidationErrors.

JavaScript

On the client side, Form objects are also responsible for executing their list of Rules when a field value change. This makes them responsible for DOM manipulation. DOM manipulation is also encapsulated inside Form objects, in functions such as `fieldShow()`, `fieldHide()`, `fieldValueGet()`, and so on, for example:

```
// Return the jQuery field container for a field name, it's the element that
// contains both the field and label.
ddf.form.Form.prototype.fieldContainerGet = function(field) {
    return this.fieldGet(field).parents().has(this.fieldLabelGet(field)).first();
}
```

The more logic that couples the DOM we can put inside the Form object, the easier it will be to subclass it and replace the ones that are not compatible with a particular DOM tree outside the admin:

```
ddf.form.BootstrapForm.prototype = Object.create(ddf.form.Form.prototype);
ddf.form.BootstrapForm.prototype.fieldContainerGet = function(field) {
    // something else
}
```

Then, actions can be abstracted from the DOM as such:

```
// Hide the field.
ddf.action.Remove.prototype.apply = function(form, field) {
    form.fieldHide(field);
}

// Show the field.
ddf.action.Remove.prototype.unapply = function(form, field) {
    form.fieldShow(field);
}
```

Another important thing to note here is that a Form in JS really matches a Form in Python. This means that it supports the `prefix` attribute, and that there would be as many instances of `Form` objects in a formset in the client exactly like there would on the server.

Python

Nothing exceptional here, we're going with the standard ingredients:

- tdd,
- coverage,
- py.test,

JavaScript

We have the script you can see in `src/ddf/static/ddf.js`, but I'm currently revamping this:

- code in ES6 and build ES5 code, because we're trying to have some sort of equivalent mindset between python and js code, so it'll be easier if the class syntax look alike,
- use standard module import code, with requirejs and browserify,
- we need unit tests and coverage, let's learn from jal !
- use gulp to simplify the workflow of course !

CHAPTER 2

Readme

This app provides features a form class dynamic, both on the client and server side. For example if you want a field to be remove if another field has a particular value or to filter choices based on a set conditions. The demo is automatically updated when a commit is tested on master.

Pypy, Python 2.7, 3.4, Django 1.8+ are supported.

CHAPTER 3

Example

For example, if your form should allow support only if linux is selected:

```
from ddf import shortcuts as ddf

class TestForm(ddf.FormMixin, forms.Form):
    platform = forms.ChoiceField(choices=(
        ('Linux', 'Linux'),
        ('BSD', 'BSD'),
        ('Windows', 'Windows'),
    ))
    service = forms.ChoiceField(choices=(
        ('Setup', 'Setup'),
        ('Support', 'Support')
    ))

    _ddf = dict(
        # List of Actions to execute on the service field
        service=[
            # Remove the service field
            ddf.Remove(
                # If platform field is Windows
                ddf.ValueIs('platform', 'Windows'),
            ),
            # Remove a list of choices from the service field
            ddf.RemoveChoices(
                ['Support'], # That's the list of choices to remove
                # If platform field value is Windows
                ddf.ValueIs('platform', 'BSD'),
            )
        ]
    )
```

This example will cause the “service” field to be removed when the user selects platform=Windows. If the user selects BSD, then it will just remove the Support choice from the service field.

The configuration field is able to render the configuration as a JSON dict, with the form prefix it's being rendered with. Then, the equivalent of each Action and Condition objects are instanciated in JavaScript, allowing dynamic user experience.

A configuration is structured as such: for each field, you may add a list of actions, for each action a list of conditions. When the user changes a field, each action's conditions are evaluated and if they all pass then the action is applied, otherwise it is unapplied. Possibilities are huge here.

CHAPTER 4

Dual-license

It is released with the Creative Commons Attribution-NonCommercial 3.0 Unported License, but a commercial license is available, please get in touch with the author by email (see setup.py) if you are interrested.

Note that the money goes to YourLabs, a non-profit foundation to promote the role of hackers in the process of making our society more fair and free, while using their skills to develop local economy and give internet back to the people.

CHAPTER 5

Status

The project is pretty young, but the basic building blocks are there. We should be able to add actions and conditions easily.

CHAPTER 6

Resources

- **Documentation** graciously hosted by RTFD
- Live demo graciously hosted by RedHat,
- Mailing list graciously hosted by Google
- For **Security** issues, please contact yourlabs-security@googlegroups.com
- Git graciously hosted by GitHub,
- Package graciously hosted by PyPi,
- Continuous integration graciously hosted by Travis-ci
- **Online paid support** provided via HackHands,

CHAPTER 7

Why

We've been inventing this over and over again for years. The first time I invented this was in 2009 and honnestly my python, django and javascript skills were pretty weak back then. Since then, I've seen users asking this, paying me as a consultant for this, making pull requests to have this in a per-app basis. It's about time we have a generic solution that works for all kinds of fields, and not just the ones of the apps we maintain.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search