django-dynamic-forms

Release 0.5.0

July 21, 2016

Contents

Contents:

Warning: **django-dynamic-forms** 0.5.x will only support Django >= 1.7! If you need support for Django < 1.7 use **django-dynamic-forms** 0.4.x!

CHAPTER 1

Installation

Warning: **django-dynamic-forms** 0.5.x will only support Django >= 1.7! If you need support for Django < 1.7 use **django-dynamic-forms** 0.4.x!

Install **django-dynamic-forms** into your virtual environment or you site-packages using pip:

```
$ pip install django-dynamic-forms
```

If you already use the wheel package format you can use the wheel build:

```
$ pip install --use-wheel django-dynamic-forms
```

To make **django-dynamic-forms** available in your Django project, you first have to add it to the `INSTALLED_APPS` in your `settings.py`. If you are unsure where to put it, just append it:

```
INSTALLED_APPS = (
    ...
    'dynamic_forms.apps.DynamicFormsConfig',
    ...
)
```

To make Django aware of the dynamic forms while processing the requests / responses you need to add the `FormModelMiddleware` to the list of `MIDDLEWARE_CLASSES`. The best place is probably at the end of the list. If your forms are not shown please refer to the known problems section of the documentation:

```
MIDDLEWARE_CLASSES = (
    ...
    'dynamic_forms.middlewares.FormModelMiddleware'
)
```

Last but not least you need to add the `'dynamic_forms.urls'` urlpatterns to your project's URL patterns:

```
urlpatterns = patterns('',
    ...
    url(r'^dynamic_forms/',
        include('dynamic_forms.urls', namespace='dynamic_forms')),
    ...
)
```

Important: Make sure that you get the namespace straight: `dynamic_forms`!

Finally you have to update your database. Run:

```
$ python manage.py migrate dynamic_forms
```

CHAPTER 2

Usage

2.1 Using Custom Templates

django-dynamic-forms comes with a basic template that just displays the form or a success page. You can customize these templates to your needs.

2.1.1 The Form Template

The following code shows the default template rendering a dynamic form.

```
{% load i18n %}
<h2>{{ name }}</h2>
<form method="post" action="{{ submit_url }}">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">{% trans "Submit" %}</button>
</form>
```

The `DynamicFormView` exposes three variables to the template context related to the form:

`form` An instance of the form that will be shown on this page. As the form is a normal Django form, all rules from the Django documentation apply.

`model` An instance of the form model providing the form and assigned to this URL.

`name` The form's name as defined in *`dynamic_forms.models.FormModel.name`*.

`success_url` The URL the form will be submitted to as defined in *`dynamic_forms.models.FormModel.submit_url`*. This is *not* the *`success_url`*!

2.1.2 The Success Template

The following code shows the success template after a successful form submit.

```
{% load i18n %}
<h2>{% trans "Success" %}</h2>
<p>{% trans "Form submitted successfully" %}</p>
{% if data %}<p>{% blocktrans with link=data.show_url_link %}For your convenience you can see your da
```

The `DynamicTemplateView` exposes two variables to the template context related to the form:

`model` An instance of the form model assigned to this URL.

data If an instance of *FormModelData* if a existing `display_key` is given and the form model (`model`) has set `allow_display` to `True`.

2.2 Third Party Apps

2.2.1 django-simple-captcha

django-simple-captcha provides easy CAPTCHA support for Django forms. This contrib package integrates django-simple-captcha into **django-dynamic-forms** allowing users to add a CAPTCHA field to their dynamic forms.

To use it make sure you installed **django-simple-captcha**:

```
$ pip install django-simple-captcha
```

Next put `'captcha'` and `'dynamic_forms.contrib.simple_captcha'` in the `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    'captcha',
    'dynamic_forms.contrib.simple_captcha',
    ...
)
```

CHAPTER 3

Problems

3.1 My form is not shown

If you are sure you followed the Installation instructions, there are several reasons:

1. You might have misspelled the URL path. Please check again.
2. There is another view that maps to the URL you defined in your model. 1. If you have Django's flatpages framework installed, check please check

   that there is not page mapping to this URL.
3. An error occurs while constructing the form or rendering the template. Set `DEBUG = True` in your `settings.py` and have a look at the exception that is raised.

CHAPTER 4

Changes

4.1 v0.5 (under development)

Warning: **django-dynamic-forms** 0.5.x will only support Django >= 1.7!

- Form action take a required third argument `request`.
- Support for Django 1.9
- Removed `six` dependency
- Added missing contrib packages (thanks Seán Hayes) (#23)

4.2 v0.4.1 (SECURITY RELEASE)

- Fixed a CSRF vulnerability in the `POST` handling of forms. Thanks Marten Kenbeek for the report and patch. (#26)

4.3 v0.4

Warning: **django-dynamic-forms** 0.4.x will be the latest version branch that supports Django < 1.7!

- Added support for Django 1.8 and experimental support for Django 1.9. (#19)
- Removed `django-appconf` dependency.
- Added per form email receivers (thanks to Nar Chhantyal). (#13)
- Increased form fields' label `max_length` to 255 (thanks to Nar Chhantyal). (#18)

4.4 v0.3.4

- Fixed a issue with missing migrations on Python 2. (#11)

4.5 v0.3.3

- Updated Portuguese translation (thanks Gladson Simplicio). (#8)

4.6 v0.3.2

- Introduced the settings variables *DYNAMIC_FORMS_FORM_TEMPLATES* and *DYNAMIC_FORMS_SUCCESS_TEMPLATES* to make defining the templates for the form and success display easier and more usable for non-developers. (#1)
- Allowed delayed registration of actions and dynamic form fields.
- Allowed dynamic fields to exclude their value from the `mapped_data` by overriding *do_display_data()*.
- Added support for django-simple-captcha. (#2)
- Added Portuguese translation (thanks Gladson Simplicio). (#4)
- Replaced *formfields.dynamic_form_field_registry* with *formfields.formfield_registry* and deprecated the former.
- Fixed sorting of actions and field types by their label. (#5)
- Allowed users to get a link to see the data they submitted before at a later time. (#6)

4.7 v0.2

- Fixed some packaging issues. (thanks Jannis Leidel)
- Added Django 1.7's db.migrations.
- Moved to tox for development testing.

4.8 v0.1

- Initial release

CHAPTER 5

References

5.1 Actions

Actions define what happens once a user submits a *FormModelForm*. **django-dynamic-forms** provides two basic actions *dynamic_form_send_email()* and *dynamic_form_store_database()* that, as their names indicate, either send the submitted data via e-mail to the recipients defined in the *DYNAMIC_FORMS_EMAIL_RECIPIENTS* settings variable or stores it into the database (precisely the *FormModelData* model).

Any action that should be available for usage must be registered in the *ActionRegistry*. This can be done with the following code:

```
>>> def my_function(form_model, form, request):
...     # do something
...     pass
...
>>> from dynamic_forms.actions import action_registry
>>> action_registry.register(my_function, 'My Label')
```

This allows one to register an action during runtime. But many actions are already available during compile or start-up time and can be registered then by using a handy decorator *formmodel_action()*. Given the above situation, this would look like:

```
>>> from dynamic_forms.actions import formmodel_action
>>> @formmodel_action('My Label')
... def my_function(form_model, form, request):
...     # do something
...     pass
...
```

New in version 0.3: When a dynamic form is submitted through `DynamicFormView` the return values of actions are kept for further usage. This allows the view to e.g. add a link to a permanent URL refering to some stored values.

5.1.1 Providing and accessing actions

ActionRegistry

class dynamic_forms.actions.**ActionRegistry**

The ActionRegistry keeps track of all available actions available to the software. It is available to the outside through the *action_registry* singleton

Warning: You should not import the *ActionRegistry* directly! Always use the singleton instance *action_registry*!

```
>>> from dynamic_forms.actions import action_registry
```

get(*key*)

Parameters **key** (*str*) – The key to get an action

Returns Either the action previously registered or `None` if no action with the given key has been found.

get_as_choices()

Changed in version 0.3: Returns a generator instead of a list

Returns a generator that yields all registered actions as 2-tuples in the form `(key, label)`.

register(*func*, *label*)

Registers the function `func` with the label `label`. The function will internally be referred by it's full qualified name:

```
'%s.%s' % (func.__module__, func.__name__)
```

Parameters

- **func** (*callable*) – The function to register.
- **label** (*str*) – A string / unicode giving the action a human readable name

unregister(*key*)

Looks up the given key in the internal dictionary and deletes the action if it exists.

Parameters **key** (*str*) – The key an action is assigned to

dynamic_forms.actions.**action_registry**

The singleton instance of the *ActionRegistry*.

Action registry utilities

@dynamic_forms.actions.**formmodel_action**(*label*)

Registering various actions by hand can be time consuming. This function decorator eases this heavily: given a string as the first argument, this decorator registeres the decorated function withing the *action_registry* with its fully dotted Python path.

Usage:

```
@formmodel_action('My super awesome action')
def my_action(form_model, form, request):
    # do something with the data ...
```

This is equivalent to:

```
def my_action(form_model, form, request):
    # do something with the data ...

action_registry.register(my_action, 'My super awesome action')
```

5.1.2 Default Actions

dynamic_forms.actions.**dynamic_form_send_email**(*form_model*, *form*, *request*)

Sends the data submitted through the form `form` via e-mail to all recipients listed in *DYNAMIC_FORMS_EMAIL_RECIPIENTS*.

Parameters

- **form_model** (dynamic_forms.models.FormModel) – The instance of the model defining the form.
- **form** (dynamic_forms.forms.FormModelForm) – The instance of the submitted form. One can get the data either using `form.cleaned_data` or, if the labels defined in the `form_model` for each field are needed, in the appropriate order by calling *get_mapped_data()*.
- **request** (*django.http.Request*) – The current request

New in version 0.5: The `request` parameter was added.

dynamic_forms.actions.**dynamic_form_store_database**(*form_model*, *form*, *request*)

This action takes the mapped data from the `form` and serializes it as JSON. This value is then stored in the *FormModelData*.

See also:

dynamic_form_store_database() for a detailed explaination of the arguments.

New in version 0.3: To allow linking to a stored data set, the action now returns the inserted object.

New in version 0.5: The `request` parameter was added.

5.2 Admin

5.3 Fields

class dynamic_forms.fields.**TextMultiSelectField**([*separate_values_by='\n'*, ***options*])

Provides multiple choice field storage for strings without limiting the total length of the string or giving any restrictions of which characters are not allowed because they are used to split the input value into its different choices.

Parameters **separate_values_by** (*str*) – The string used to split the input value into its choices. Defaults to `'\n'`.

See also:

The respective form field as part of **django-dynamic-forms** *dynamic_forms.forms.MultiSelectFormField*. The common field options and the specifics for the `django.db.models.TextField`.

Note: The implementation is based on http://djangosnippets.org/snippets/2753/ but has been modified to the needs of this project. Thus, there is no conversion of the selected items to `int` or similar.

5.4 Form fields

dynamic_forms.formfields.**format_display_label**(*cls_name*)

dynamic_forms.formfields.**load_class_from_string**(*cls_string*)

5.4.1 DynamicFormFieldRegistry

class dynamic_forms.formfields.**DynamicFormFieldRegistry**(*object*)

get(*key*)

get_as_choices()
Changed in version 0.3: Returns a generator instead of a list

Returns a generator that yields all registered dynamic form fields as 2-tuples in the form `(key, display_label)`.

register(*cls*)

unregister(*key*)

dynamic_forms.formfields.**formfield_registry**
New in version 0.3: Use this instead of *dynamic_form_field_registry*

dynamic_forms.formfields.**dynamic_form_field_registry**
Deprecated since version 0.3: Deprecated in favor of *formfield_registry*

@dynamic_forms.formfields.**dynamic_form_field**(*cls*)
A class decorator to register the class as a dynamic form field in the *DynamicFormFieldRegistry*.

5.4.2 Base Form Field Classes

DFFMetaclass

class dynamic_forms.formfields.**DFFMetaclass**
Metaclass that inspects the `Meta` class of a class inheriting from *BaseDynamicFormField* and merges the different attributes that are later being passed to the respective `django.forms.Field`.

You are free to add an attribute `_exclude` of type `list` or `tuple` to the `Meta` class of a field to exclude any attributes inherited from a super DynamicFormField. Look at the implementation of the *BooleanField* for an example.

BaseDynamicFormField

class dynamic_forms.formfields.**BaseDynamicFormField**

cls
None

display_label
None

widget
None

options

class Meta

help_text
[six.string_types, ‘’, (forms.CharField, forms.Textarea)]

required
[bool, True, forms.NullBooleanField]

dynamic_forms.formfields.**__init__**(*name*, *label*, *widget_attrs={}*, ***kwargs*)

dynamic_forms.formfields.**__str__**()
dynamic_forms.formfields.**__unicode__**()

dynamic_forms.formfields.**construct**([***kwargs*])

dynamic_forms.formfields.**contribute_to_form**(*form*)

dynamic_forms.formfields.**get_display_label**()
Returns a class’s `display_label` is defined or calls *format_display_label()* with the class’s name.

This function is only available to the class itself. It is not callable from an instance.

dynamic_forms.formfields.**get_widget_attrs**()

dynamic_forms.formfields.**set_options**([***kwargs*])

dynamic_forms.formfields.**options_valid**()

classmethod dynamic_forms.formfields.**do_display_data**()
Default: `True`

5.4.3 Default Fields

class dynamic_forms.formfields.**BooleanField**

cls
’django.forms.BooleanField

display_label
’Boolean

class Meta

_exclude
(’required’,)

class dynamic_forms.formfields.**ChoiceField**

cls
’django.forms.ChoiceField

display_label
’Choices

class Meta

choices
[six.string_types, ‘’, (forms.CharField, forms.Textarea)]

dynamic_forms.formfields.**construct**([***kwargs*])

dynamic_forms.formfields.**options_valid**()

class dynamic_forms.formfields.**DateField**

cls
'django.forms.DateField

display_label
'Date

class Meta

localize
[bool, True, forms.NullBooleanField]

class dynamic_forms.formfields.**DateTimeField**

cls
'django.forms.DateTimeField

display_label
'Date and Time'

class Meta

localize
[bool, True, forms.NullBooleanField]

class dynamic_forms.formfields.**EmailField**

cls
'django.forms.EmailField

display_label
'Email

class dynamic_forms.formfields.**IntegerField**

cls
'django.forms.IntegerField

display_label
'Integer

class Meta

localize
[bool, True, forms.NullBooleanField]

max_value
[int, None, forms.IntegerField]

min_value
[int, None, forms.IntegerField]

class dynamic_forms.formfields.**MultiLineTextField**

cls
'django.forms.CharField

display_label
'Multi Line Text

widget
'django.forms.widgets.Textarea

class dynamic_forms.formfields.**SingleLineTextField**

cls
'django.forms.CharField

display_label
'Single Line Text

class Meta

max_length
[int, None, forms.IntegerField]

min_length
[int, None, forms.IntegerField]

class dynamic_forms.formfields.**TimeField**

cls
'django.forms.TimeField

display_label
'Time

class Meta

localize
[bool, True, forms.NullBooleanField]

5.5 Forms

5.5.1 Form Fields

class dynamic_forms.forms.**MultiSelectFormField**([*separate_values_by='\n'*, ***options*])
Provides multiple choice field storage for string objects without limiting the total length of the string.

Parameters **separate_values_by** (*str*) – The string used to split the input value into its choices. Defaults to '\n'.

See also:

The respective database field as part of **django-dynamic-forms** *dynamic_forms.fields.TextMultiSelectField*. The core form field arguments and the specifics for the django.forms.MultipleChoiceField.

Note: The implementation is based on http://djangosnippets.org/snippets/2753/ but has been modified to the needs of this project.

5.5.2 Forms

class dynamic_forms.forms.**FormModelForm**(*model*[, **args*, ***kwargs*])

get_mapped_data([*exclude_missing=False*])
Returns an dictionary sorted by the position of the respective field in its form.

Parameters **exclude_missing** (*boolean*) – If `True`, non-filled fields (those whose value evaluates to `False`) are not present in the returned dictionary. Default: `False`

5.6 Middlewares

class dynamic_forms.middlewares.**FormModelMiddleware**
This middleware intercepts all HTTP 404 responses and checks if there is a form mapped to this URL. This way an explicit URL mapping from the projects ROOT_URLCONF cannot accidentally be overridden by wrong setting for *submit_url* or *success_url* on *dynamic_forms.models.FormModel*.

This technique is comparable to the one used by Django's `FlatpageFallbackMiddleware`.

process_response(*request*, *response*)
The algorithm that decides if and which form to display works like this:

1. If the `status_code` for `response` is **not** 404 (`NOT FOUND`) this the *FormModelMiddleware* will return the response as-is and will not modify it. Thus, server error (5xx) will also not be affected by the middleware.
2. If there is a *FormModel* whose *submit_url* matches the request's `path_info`, this model is used to construct and render the view.
3. If there is a *FormModel* whose *success_url* matches the request's `path_info`, this model is used to display the success page.

   Note: Since the `success_url` of a `FormModel` is not necessarily be unique, the first model that matches the request path will be used.

4. If any errors occur while processing a form the original request is returned (if `DEBUG = True` the respective exception is raised).

5.7 Models

5.7.1 FormModel

class dynamic_forms.models.**FormModel**

name
django.db.models.CharField

- max_length = 50
- unique = True

submit_url
django.db.models.CharField

- max_length = 100
- unique = True

success_url
django.db.models.CharField

- max_length = 100
- blank = True
- default = ''

actions
dynamic_forms.fields.TextMultiSelectField

- default = ''
- choices = *dynamic_forms.actions.ActionRegistry.get_as_choices()*

form_template
django.db.models.CharField

- max_length = 100
- choices = *dynamic_forms.conf.DYNAMIC_FORMS_FORM_TEMPLATES*

success_template
django.db.models.CharField

- max_length = 100
- choices = *dynamic_forms.conf.DYNAMIC_FORMS_SUCCESS_TEMPLATES*

allow_display
django.db.models.BooleanField

- default = False

recipient_email
django.db.models.EmailField

fields
Related name by *FormFieldModel*

data
Related name by *FormModelData*

class **Meta**

ordering
['name']

dynamic_forms.models.**__str__**()
dynamic_forms.models.**__unicode__**()

dynamic_forms.models.**get_fields_as_dict**()

dynamic_forms.models.**save**([**args*, ***kwargs*])

5.7.2 FormFieldModel

class dynamic_forms.models.**FormFieldModel**

parent_form
django.db.models.ForeignKey

- Foreign key to *FormModel*
- on_delete = django.db.models.CASCADE

field_type
django.db.models.CharField

- max_length = 255
- choices = *dynamic_forms.formfields.DynamicFormFieldRegistry.get_as_choices()*

label
django.db.models.CharField

- max_length = 255

name
django.db.models.CharField

- max_length = 50
- blank = True

_options
django.db.models.TextField

- blank = True
- null = True

position
django.db.models.SmallIntegerField

- blank = True
- default = 0

options
Property wrapping JSON serialization and deserialization around the *_options*.

class Meta

ordering
['parent_form', 'position']

unique_together
("parent_form", "name",)

dynamic_forms.models.**__str__**()
dynamic_forms.models.**__unicode__**()

dynamic_forms.models.**generate_form_field**(*form*)

dynamic_forms.models.**get_form_field_kwargs**()

dynamic_forms.models.**save**([*args*, ***kwargs*])

5.7.3 FormModelData

class dynamic_forms.models.**FormModelData**

form
django.db.models.ForeignKey

- Foreign key to *FormModel*
- on_delete = django.db.models.SET_NULL
- null = True

value
django.db.models.TextField

- blank = True
- default = ''

submitted
django.db.models.DateTimeField

- auto_now_add = True

__str__()
__unicode__()

pretty_value()

5.8 Settings

5.8.1 DYNAMIC_FORMS_EMAIL_RECIPIENTS

dynamic_forms.conf.**DYNAMIC_FORMS_EMAIL_RECIPIENTS**
A list of email addresses. Used to define the receipients form data will be send to if the action *dynamic_form_send_email()* is activated.

Defaults to all email addresses defined in the ADMINS setting.

5.8.2 DYNAMIC_FORMS_FORM_TEMPLATES

dynamic_forms.conf.**DYNAMIC_FORMS_FORM_TEMPLATES**
New in version 0.3.

A tuple of 2-tuples passed to the *FormModel*'s *form_template* attribute. This setting provides easier and less error-prone definition of the form template.

Defaults to:

```
(
    ('dynamic_forms/form.html', _('Default form template')),
)
```

5.8.3 DYNAMIC_FORMS_SUCCESS_TEMPLATES

dynamic_forms.conf.**DYNAMIC_FORMS_SUCCESS_TEMPLATES**
New in version 0.3.

A tuple of 2-tuples passed to the *FormModel*'s *success_template* attribute. This setting provides easier and less error-prone definition of the form template.

```
(
    ('dynamic_forms/form_success.html', _('Default success template')),
)
```

5.9 Views

5.10 Contrib packages

5.10.1 django-simple-captcha

class dynamic_forms.contrib.simple_captcha.models.**CaptchaField**

cls
'captcha.fields.CaptchaField

display_label
'CAPTCHA

class Meta

_exclude
('required',)

classmethod dynamic_forms.contrib.simple_captcha.models.**do_display_data**()
Default: True

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

a

c

f

m

v

Symbols

A

B

C

D

E

F

G

H

I