Formset & Form Improvements

Parth Patil

ABSTRACT

- **OVERVIEW OF PROBLEM**

    Django currently has a formset class which is used as a collection instance for the forms. It’s a layer of abstraction which makes it easier to work with multiple forms. But it's not perfect, there is a good room for improvement.

    One problem with the model formset is that it can’t be used for only editing a set of models. The user may not want to affect other models in any way. Trying this in the current framework raises issues such as one can create new Models by synthesizing a POST request, which is not the desired behavior.

    Another problem with formset objects is that the user loses some control over the initialization and saving process of individual forms. For e.g. passing the details of the logged-in user while creating the form turns out to be a difficult problem. There are few workarounds for this problem, but certainly, lack of documentation exists.

    Apart from these, there are many more issues that are addressed below. Also, this project proposal will aim to add a few features requested by users, e.g. pagination.

- **GOALS**

  The broad goal is obviously to have an error-free Formset and close all the tickets related to formset. The end goal of this project will be to address as many issues as possible and also make the formsets more user-friendly.

  I aim to give the user more control over the individual forms in the formset, by enabling the user to pass different arguments for `__init__()` and `save()` methods of each form.

  The next goal will be to enable the user to use formsets for edit only purpose, by implementing a robust framework that will disable the creating of new model objects, instead of workarounds that can't do much in case of forged POST requests.

  After that, the next goal will add some new features to formset like pagination, declarative syntax of ModelFormSet & InlineFormSet.

- **BENEFITS**

  Apart from apparent benefits like closing the tickets, I think this may encourage users to use formsets in their projects. With features like pagination, a user making a Shopping website can directly use a formset to display the order page which can be used to properly transverse through different pages of an ordered item without refreshing.

  Also, it will give user granular control over each form of how it will be saved, etc. Also, formsets can become a quick way to edit many Model objects at once, without creating any new objects, or affecting the existing objects.

The Issues to be addressed

- **The passing of arguments to forms inside a formset (see this):**
  Consider this example

```
class ServiceForm(forms.Form):
    option = forms.ModelChoiceField(queryset = ServiceOption.objects.none())
    rate = forms.DecimalField(widget = custom_widgets.SmallField())
    units = forms.IntegerField(min_value=1,
widget=custom_widgets.SmallField())

    def __init__(self, *args, **kwargs):
        affiliate = kwargs.pop('affiliate')
        super(ServiceForm, self).__init__(*args, **kwargs)
        self.fields["option"].queryset = \
            ServiceOption.objects.filter(affiliate=affiliate)
```

Here affiliate is the current user. Normally the user having something like this will initiate the form like

```
form = ServiceForm(affiliate=request.affiliate)
```

But if we use formsets, it's difficult to pass the request to the individual form.

```
ServiceFormSet = forms.formsets.formset_factory(ServiceForm, extra=3)
formset = ServiceFormSet()
```

The FrameWork I propose will enable to pass individual arguments to each form while initialization and also while saving.
So in the new BaseFormSet class

```
class BaseFormSet:
     def __init__(self, ..., extra_form_parameters = {}, request=None):
        Self.request = request
```

```
  def _construct_forms(self, extra_form_parameters):
      '''
      This function will pass the extra parameters to each form individually
      Can pass the self.request to forms or create forms depending on request
      '''
      do_some_stuff
```

- **Enable model formsets to block new object creation (edit only mode) (#26142):**

  There is no good way in which we can use Model Formset for just editing a set of models without creating or affecting any other model. Few workarounds are present, but none produces the desired behavior.

  For e.g. one can argue that setting `extra_form = 0` , `max_num = 0` & `validate_max = True` will work. But if by some way the POST request is edited, and a different pk value that doesn't appear in the queryset is sent, it will create a new model object, which is not desirable.

  So this issue can be solved by adding an extra parameter `no_edit` in the `BaseModelFormSet` class. This will disable the `save_new_objects()` method thus ensuring no new object will be created. And also one can pass a list of pk values to the BaseFormSet while creating an object of formset so that only the pk values from that list can be edited, thus acting as a security feature so that no one can simply edit the POST request and affect the database.

  The new BaseModelFormSet class

```
class BaseModelFormSet:
     def __init__(se wlf, ..., no_edit = False, only_edit = None):
         self.only_edit = only_edit or None
```

If `only_edit is not None` then the formset will only edit the model object whose pk is present in the list else it will do nothing. This will ensure a robust edit only behavior for the ModelFormSet. Similar can be done for the InlineFormSets too.

- **Allowing the swapping of model object's pk values using the formset (see #25139):**

  If let's say you have two model objects `p0 (slug = 'car-red')` and `p1(slug = 'car-blue')` in the database. And you receive a POST request containing the following data.

```
data = {
    'form-TOTAL_FORMS': '2',
    'form-INITIAL_FORMS': '2',
    # Swap slug values.
    'form-0-id': p0.pk,
    'form-0-slug': 'car-blue',
    'form-1-id': p1.pk,
    'form-1-slug': 'car-red',
}
```

  And if we save the form we would expect this to swap the pk of both the objects, and save them in the database, but this is not the actual behavior.

  So there is a need to implement a method that can handle a permutation of the object's pk values. Adding this with the "edit only" mode mentioned previously will make more sense since restricting the pk values of objects which will be affected is a must. The swapping of models will be a feature added for the "edit only" mode.

- **Pagination (suggested by Dmitriy):**

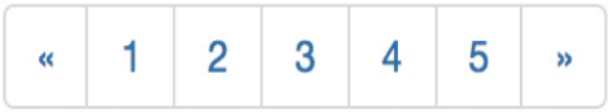
One more feature I would like to add is of paginated HTML output for formsets. It would be a method in BaseFormSet class similar to as_p, as_table methods.

```
def paginate(self, no_of_elements = 3):
    '''
    Return the paginated formset with exactly three forms on each Page
    '''
    do_some_stuff
```

This function will take the two inputs, that is the instance of the formset and the `no_of_elements` that should appear on each page in the HTML template. This will help

in case of someone having a large number of forms and don't want them to be printed at once.

The end result of this may look something like this at the bottom of the form.



- **Declarative Syntax for the Model and Inline Formsets (see #10403):**

One more feature which was requested and will be a nice addition to the formset class would be the declarative syntax for the Model and Inline FormSet classes. Below is an example of how this will look like in practice.

```
class DeclarativeAuthorFormSet(forms.models.ModelFormSet):
    form = AuthorForm
    model = Author
    extra = 3

class DeclarativeAuthorBooksFormSet(forms.models.InlineFormSet):
    model = Book
    form = BookForm
    parent = 'author'
```

Adding this may be a little tough, as I would have to learn first about the declarative syntax and I feel Form or Model base class will be a good starting point. Also while doing this backward compatibility should be maintained.

For doing this I will have to edit the `__init__()` functions of BaseModelFormSet and BaseInlineFormSet (according to me), more planning will be required on this one, which will be done in the discussion phase.

- **Handling the Relation of the extra forms in Inline Formset (see #25880):**

The extra forms in the Inline Formsets are not initiated with the relative instance of the parent object by the `formset_factory()`. That is referring to the `extra_form.related_field` throws an exception that `RelatedObjectDoesNotExist`

Even though the `_contruct_form()` methods seem to assign the form instance, I was able to duplicate the error, so a little more investigation will be needed to solve this error.

- **Other related issues:**
  - **Updating the exception messages in the Management Form (#13060)**

    This may be a small change in the cleanup of code. The complaint of this lies in the fact that the error messages that are shown for the management forms are not quite helpful, that is they don't explain what the actual problem is. This may be quite painful sometimes.

    My task will be to understand the different errors that may occur in the management form of the formset and document them properly and also add the proper help text in each of them.

Timeline And Milestones

My summer vacation starts from May 6, 2019, during which I plan to spend around 35 - 40 hours a week. My third academic year starts from July 29, after which I might have to cut it down working to 3-4 hours on weekdays and the regular 5-7 hours on weekends. This period is a little shorter, however, I may work during the semester until the official end of GSoC. Hence I intend to do most of GSoC during the period 6th May to the 30th July instead of the usual period of 27th May to 19th August. I have tried to structure the timeline in a way that there is a good mix of coding, learning, and documentation. So that I can learn every aspect, and don't get to saturated by one thing. I also plan to write a blog post every week to track my GSoC progress, containing links to my contributions, a brief overview, and the plan for next week. This will ensure that these are not lost in the future.

- **Pre-Summer Most of April (Application Review Period):**
    - Solve a few bugs related forms (easy pickings maybe) to get a little more involved in forms.
    - Getting familiarize with the conventions. I already have a pull request merge into Django (see ticket - #30189, PR - #11039), So it won't take much time to know the conventions.
- **May 6 - May 10 (Community Bonding Period)**
    - Discuss the Implementation of the project in detail with a mentor and senior contributors from the community
    - Understand whether my approach is correct and figure out a good way to approach the problem, if there is any.
    - As previously mentioned I'm pretty comfortable with the coding environment of Django, so I can directly skip to actually start working on the project. (Making new friends won't hurt anyway :))

- **May 11 - May 28: Passing the custom parameter to each form**
    - Starting with the first issue mentioned above, I will test already available solution for passing the request variable around the formset and documenting them (May 11 - May 14)
    - Formalizing on the new architecture for the `BaseFormSet` class and the `_construct_form` method (May 15 - May 18)
    - Implementing the changes in code will be the next milestone. Ensuring that everything works as intended. (May 18 - May 25)
    - Writing tests, documentation (May 26 - May 28)
- **May 29 - June 10: Implementing the Edit Only mode**
    - Deciding on the complete features and specifics of how to do this will be decided (May 29 - May 30)
    - Changing the `BaseModelFormSet` class. (May 31 - June 8)
    - Writing tests, documentation (June 8 - June 10)
- **June 11 - June 21: Pagination**
    - Learning about different ways to accomplish pagination (June 11 - June 12)
    - Implementing the `pagination` function. I will try to add some good styling to go along with this. (June 13 - June 18)
    - Writing tests, documentation (June 19 - June 21)
- **June 22 - July 8: Declarative Syntax for Model and inline Formsets**
    - Learn about declarative syntax, how it is used, its features and how to write classes that use declarative syntax. A good place to start may be reading the `BaseModel` class. (June 22 - June 26)
    - Finalizing the changes in `BaseModelFormSet` and `BaseInlineFormSet` classes. (June 27)
    - Implementing it in code (June 28 - July 4)
    - Writing tests, documentation (July 5- July 8)

- **July 9 - July 16: Swapping of model object's pk values using the formset**
    - Testing the different edge cases as the test given the tickets are not complete. So I would be writing complete tests which will address the problem (July 9 - July 10)
    - Writing the patch for this (July 11 - July 16)
- **July 16 - July 27: Fixing relation of the extra forms in InlineFormset**
    - Documenting a Proper way to replicate this bug would be the first task. (July 16 - July 18)
    - Analyzing the `_construct_form` method and figuring out the best way to fix this (July 19 - July 20)
    - Writing a patch and documentation (July 21 - July 27)
- .**July 27 - July 31: Updating the exception messages in the Management Form**
    - Finding and documenting the help text for the management forms.

If time permits...

If I finish earlier, I would like to address more issues like #28283 or #9739. I also have few ideas like FormSetContainer, which may need a more discussion before implementation.

What will I do in the remaining time?

As mentioned earlier, my semester will start around July 29th, after which my working hours will drop. To compensate that I'll be starting my project early in order to do 12 weeks' worth of work. Also, I will use the remaining time to document everything and jobs like code clean up and some small maintenance tasks.

ABOUT ME

I'm **Parth Patil** pursuing my B.Tech. in Electrical Engineering at the Indian Institute of Technology, Bombay. I am in my second year. I was introduced to programming right in my childhood, and I developed a passion for it since then. I have always enjoyed Mathematics in solving real-world problems. I started coding from a very young age, I built my first app in 8th grade and my first custom ROM for my android tab in 9th grade. I'm currently a part of the software sub-division Team AUV-IITB. Our team was runner up winner in 2016's Robosub Competition which sees participation from the team all across the world. Currently, I'm working on an Extended Kalman Filter for POSE estimation of the Autonomous Underwater vehicle (AUV). I'm also active in many Technical clubs in the institute and mostly fit in well with the community. I also recently started some work in the Django foundation and have fixed a bug(#30189). I have been using Django for the past year, and have done a couple of projects using that. I also maintain the site http://tech-iitb.org/ which is the official website of the technical council of IIT Bombay.

PROJECTS(involving Django**):**

- **Augmented Reality** Glasses: I made a local web server in the glass that was used to connect the glasses to the phone and also transfer data between the glass and the phone.
- **Web-based testing interface**: As a part of Team AUV-IITB, I was given a task to create a GUI for controlling our vehicle which will also run on Windows/Mac/Ubuntu or any system. So I chose to use Django and make a Web application that would be locally hosted on our vehicle. This project involved heavy integration of Django with the ROS framework which was a little hard, as no reference was available for this and I had to fix many bugs on my own. This involved using Django-socket to implement a live broadcast of ROS-topics.
- **Porting club website from Jekyll to Django**: One of the tough projects I did until now with Django was to port our old club website which was static and written in Jekyll to Django framework (check here). Maybe a little dirty port but had to do it in only two weeks.

- **Deploying a website from scratch**: In winter, I did an intern under DSCFE, which was to help a startup to make a whole new website from scratch and deploy it in less than a month. It was an intensive job as I was given the task to handle the whole backend and then using the REST framework interface with the Angular frontend.

These are a few projects I did which involved Django. I have also done many other projects. You can check my CV for more information.

Personal Details

- Name : Parth Patil
- Email : parthvin@gmail.com
- IRC nick : Parth1811
- Telephone : +91 99203 85585 / +91 90824 27853
- Country of Residence : India
- Timezone : Indian Standard Time (UTC + 5:30)
- Primary Language : English