django-merlin Documentation

Release 0.8

Travis Chase, Chad Gallemore

Apr 18, 2017

Contents

CHAPTER 1

Getting started

Session Wizard

Django comes with an optional "form wizard" application that allows you to split forms across multiple web pages in a sequential order. This ability is provided by using the `FormWizard` class. You would use this when you have, for example, a long registration process that needs to be split up in small digestable chunks, making it easier on your users to complete the process.

You can see the `FormWizard` documentation at: http://docs.djangoproject.com/en/dev/ref/contrib/formtools/form-wizard/

Is there a need for a different one?

In a word, yes. A few things the `FormWizard` does that may not work for your projects, as it did not for ours. First, the `FormWizard` using an HTTP `POST` to process a form. This makes it tough when you are trying to use the browser's back button to change some data on a previous step. The `FormWizard` checks for any `GET` requests and moves you to the first step in the wizard process, YUCK! Secondly, the `FormWizard` docs recommends using your wizard subclass as the callable in a `urlconf` in your urls.py. This is a really nice feature except that it will only create one copy of your `FormWizard` for all requests. This works well until you start messing with the hooks it provides to inserting or removing steps based on data from a form submission. Once you insert or remove a form, the steps are now changed for any subsequent users.

How is `SessionWizard` different?

1. The *SessionWizard* is given a list of *Step* objects instead of a list of Django `Form` classes.
2. *SessionWizard* stores all of its state in the Django `Session` object. This allows you to use the `SessionWizard` in the `urlconf` and keep state seperate by user (or session). When the `SessionWizard` starts it makes a copy of the *Step* list for the session so it can be manipulated independantly of any other session.
3. The *SessionWizard* processes all `GET` requests as a form view and only moves to the next step in the sequence on a succesful `POST` request. This allows for the browser's Back button to function correctly.

4. Each *Step* in the sequence has a unique slug for that step. This slug is used in the `urlconf` to be able to go to any part of the wizard. This allows you to provide proper "Back" and "Next" buttons on your forms.

How to use `SessionWizard`

Here is the basic workflow needed to use the `SessionWizard` object:

1. Make sure you have enabled the Django session middleware.
2. Create a subclass the `SessionWizard` class and override the `done()` method. The `done()` method allows you to collect all of the validated form data, process that data and move on to the next web page after successful processing of the wizard. You are able to redirect out of done if there are some post processing errors you need the user to be notified of. If you have processed everything correctly then you can call the `clear()` method to clean up the data stored in the session. If `clear()` is not called then the next time the same session goes through the wizard the existing form data from the original run will be put into the forms.
3. Override the `get_template()` method to return the path to the template the forms should use. The default is to return "forms/wizard.html", which you provide. Based on the step passed in you could return different templates for different forms.
4. Create a url that will be the entry point of your wizard. This url should provide a `(?P<slug>[A-Za-z0-9_-]+)` option in the url pattern.
5. Point this url to the subclass of `SessionWizard`, providing a `list` of *Step* objects that the wizard should process in the order it should process them.
6. Sit back and enjoy form wizard goodness!

How it works

1. The user makes a `GET` request to your wizard url with the first slug of the sequence.
2. The wizard returns the form using the template you specify.
3. The user submits the form using a `POST` request.
4. The wizard validates the form data. If the data is invalid it returns the user to the current form and you can display to the user any errors that have occured. If the data is valid then the wizard stores the clean data in its state object.
5. If there is another step in the process the wizard sends a redirect to the user to the next step in the sequence. If not next step is found the wizard then calls the `done()` method, which expects to return some `HttpResponse` to the user letting them know they are finished with the process.

Creating templates for the forms

You'll need to create a template that renders the step's form. By default, every form uses a template called `forms/wizard.html`. (You can change this template name by overriding `get_template()`)

The template recieves the following context:

- `current_step` – The current *Step* being processed
- `form`– The current form for the current step (with any data already available)
- `previous_step` – The previous *Step* or `None`
- `next_step` – The next *Step* or `None`

- `url_base` – The base URL that can be used in creating links to the next for previous steps
- `extra_context` – Any extra context you have provided using overriding the `process_show_form()` method

A couple of goodies

There are couple of hooks in the `SessionWizard` that allow you to modify the execution of the wizard in interesting ways. For more in depth information make sure to check out the API docs for *SessionWizard*.

- `process_show_form()` – allows you to provide any extra context data that needs to be provided to the template for processing
- `process_step()` – allows for changing the internal state of the wizard. For example, you could use this hook to add or remove steps in the process based off some user submitted information. You can use the methods `remove_step()`, `insert_before()` and `insert_after()` to accomplish this.
- `get_template()` – allows you to return a template path to use for processing the currently executing step.
- `render_form()` – allows you the ability to render the form however you see fit. The default is to use the `render_to_response` Django shortcut; but, you could use this hook to provide a `PageAssembly` render method from the excellent django-crunchyfrog project found at : http://github.com/localbase/django-crunchyfrog
- `initialize()` – allows you the ability to initialize the wizard at each request. This can be used to put data into the wizard state object that can then be used in the `done()` method.

I am tired, can't I just cancel this wizard?

When you have a long form process and the user decides they don't want to finish the wizard you would to provide a Cancel button or link they can click that will reset the wizard and redirect the user to a different screen. It would be great if the *SessionWizard* provided a way to handle this and also clean up the data it has been tracking as well. Well pine no more because the *SessionWizard* has got your back!

When you want to cancel a wizard you can just pass "cancel" as the step slug in the url. By just doing this the wizard will, by default, clear the session data it was tracking and send an *HttpResponseRedirect* to the / url. You can provide the query string parameter ?rd=yoururl to redirect to a different url. If you have a *Step* with the slug of "cancel" then the wizard will proceed to this step and you will have to handle the cancel action yourself.

For example, let's say we have a wizard and url /mywizard and we have steps "form1" and "form2".

1. The user sends a GET request to /mywizard/form1.
2. The user fills out the form information and clicks the Next button.
3. The browser sends a POST request with the form data and the wizard does its tricks and redirects the user to /mywizard/form2.
4. The user is sleepy and decides to come back tomorrow and finish the wizard. The user then clicks the cancel link you have provided in the template.
5. The cancel link in your template points to /mywizard/cancel?rd=/thanks.
6. The browser sends a GET request to /mywizard/cancel?rd=/thanks and the *SessionWizard* sees it has no step called "cancel".
7. The *SessionWizard* calls its internal cancel method, which cleans up any session and form data the wizard was tracking, and redirects the user to /thanks!
8. No harm, no foul.

- `cancel()` – cleans up the session data that has been tracked by the wizard. You can override this method and provide other features you would like when cancelling, for example; You could track the cancel actions from wizards.

Enjoy!

We are always looking for updates to make `SessionWizard` even better and provide even more form wizards to this tool chest. If you have any questions, comments or suggestions please email us at development@localbase.com. You can always particapte by using the projects GitHub account as well: http://github.com/localbase/django-merlin

Credits

This was mostly inspired by the Django form wizard and the SessionWizard snippet located here

CHAPTER 2

API documentation

SessionWizard

class merlin.wizards.session.**SessionWizard**(*steps*)

This class allows for the ability to chop up a long form into sizable steps and process each step in sequence. It also provides the ability to go back to a previous step or move on to the next step in the sequence. When the wizard runs out of steps it calls a final function that finishes the form process. This class should be subclassed and the subclass should at a minimum override the `done` method.

New in version 0.1.

Parameters **steps** – Provides a list of `Step` objects in the order in which the wizard should display them to the user. This list can be manipulated to add or remove steps as needed.

cancel(*request*)

Hook used to cancel a wizard. This will be called when slug is passed that matches "cancel". By default the method will clear the session data.

Parameters **request** – A `HttpRequest` object for this request.

clear(*request*)

Removes the internal wizard state from the session. This should be called right be for the return from a successful `done()` call.

done(*request*)

Responsible for processing the validated form data that the wizard collects from the user. This function should be overridden by the implementing subclass. This function needs to return a `HttpResponse` object.

Parameters **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.

get_after(*request*, *step*)

Returns the next `Step` in the sequence after the provided `Step`. This function will return `None` if there is no next step.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The `Step` to use as an index for finding the next `Step`

get_before(*request*, *step*)
Returns the previous `Step` in the sequence after the provided `Step`. This function will return `None` if there is no previous step.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The `Step` to use as an index for finding the next `Step`

get_cleaned_data(*request*, *step*)
Returns the cleaned form data for the provided step.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The `Step` to use to pull the cleaned form data.

get_form_data(*request*)
This will return the form_data dictionary that has been saved in the session. This will mainly be used in the done to query for the form_data that has been saved throughout the wizard process.

Parameters **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.

get_step(*request*, *slug*)
Returns the `Step` that matches the provided slug.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **slug** – The unique identifier for a particular `Step` in the sequence.

get_steps(*request*)
Returns the list of :class:‘Step‘s used in this wizard sequence.

Parameters **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.

get_template(*request*, *step*, *form*)
Responsible for return the path to the template that should be used to render this current form.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The current `Step` that is being processed.
- **form** – The Django `Form` object that is being processed.

initialize(*request*, *wizard_state*)
Hook used to initialize the wizard subclass. This will be called for every request to the wizard before it processes the GET or POST.

Parameters

- **request** – A `HttpRequest` object for this request.
- **wizard_state** – The `WizardState` object representing the current state of the wizard. Extra information can be appended to the state so it can be available to `Step`‘s of the wizard.

  For example::

  if ‘profile’ not in wizard_state: wizard_state.profile = request.user.get_profile()

insert_after(*request*, **args*, ***kwargs*)
Inserts a new step into the wizard sequence after the provided step.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **current_step** – The `Step` to use as an index for inserting a new step
- **step** – The new `Step` to insert.

insert_before(*request*, **args*, ***kwargs*)
Inserts a new step into the wizard sequence before the provided step.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **current_step** – The `Step` to use as an index for inserting a new step
- **step** – The new `Step` to insert.

process_GET(*request*, *step*)
Renders the `Form` for the requested `Step`

process_POST(*request*, *step*)
Processes the current `Step` and either send a redirect to the next `Step` in the sequence or finished the wizard process by calling `self.done`

process_show_form(*request*, *step*, *form*)
Hook used for providing extra context that can be used in the template used to render the current form.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The current `Step` that is being processed.
- **form** – The Django `Form` object that is being processed.

process_step(*request*, *step*, *form*)
Hook for modifying the `SessionWizard`‘s internal state, given a fully validated `Form` object. The `Form` is guaranteed to have clean, valid data.

This method should *not* modify any of that data. Rather, it might want dynamically alter the step list, based on previously submitted forms.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The current `Step` that is being processed.

- **form** – The Django `Form` object that is being processed.

remove_step(*request*, **args*, ***kwargs*)
Removes step from the wizard sequence.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The `Step` to remove.

render_form(*request*, *step*, *form*, *context*)
Renders a form with the provided context and returns a `HttpResponse` object. This can be overridden to provide custom rendering to the client or using a different template engine.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The current `Step` that is being processed.
- **form** – The Django `Form` object that is being processed.
- **context** – The default context that templates can use which also contains any extra context created in the `process_show_form` hook.

set_cleaned_data(*request*, **args*, ***kwargs*)
Sets the cleaned form data for the provided step.

Parameters

- **request** – A `HttpRequest` object that carries along with it the session used to access the wizard state.
- **step** – The `Step` to use to store the cleaned form data.
- **data** – The cleaned `Form` data to store.

Step

class `merlin.wizards.utils.`**Step**(*slug*, *form*)
When constucting a form wizard, the wizard needs to be composed of a sequental series of steps in which it is to display forms to the user and collect the data from those forms. To be able to provide these forms to the *SessionWizard*, you must first wrap the Django `django.forms.Form` in a `Step` object. The `Step` object gives the ability to store the `django.forms.Form` class to be used, as well as, a unique slug to be used in the wizard navigation.

New in version 0.1.

Parameters

- **slug** – Each step in the wizard should have a unique "slug" that identifies that `Step` in the process. By using slugs the wizard has the ability to go forward, as well as, back in the process adjusting what data it collects from the user.
- **form** – This *MUST* be a subclass of `django.forms.Form` or `django.forms.ModelForm`. This should not be an instance of that subclass. The *SessionWizard* will use this class to create instances for the user. If going back in the wizard process, the *SessionWizard* will prepopulate the form with any cleaned data already collected.

WizardState

class merlin.wizards.utils.**WizardState**(**args*, ***kwargs*)

This class provides the ability for a *SessionWizard* to keep track of the important state of a multi-step form. Instead of keeping track of the state through <input type="hidden"> fields, it subclasses the python UserDict object and stores its data in the properties steps,‘‘current_step‘‘ and form_data.

New in version 0.1.

Parameters

- **steps** – A list of the *Step* objects that provide the sequence in which the forms should be presented to the user.
- **current_step** – The current *Step* that the user is currently on.
- **form_data** – A dict of the cleaned form data collected to this point and referenced using the *Step*‘s slug as the key to the dict

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index