
Fifty Flask Documentation

Release 0.31

Craig Slusher

Aug 27, 2018

Contents

1	Pluggable Views API	3
1.1	Mixins	3
1.2	Views	5
2	Indices and tables	15
	Python Module Index	17

Contents:

1.1 Mixins

```
class fifty_flask.views.generic.ResponseMixin
```

```
    render_response (**context)
```

Renders a response, optionally with the provided context.

Parameters **context** – An optional dict for use in a rendered response.

```
class fifty_flask.views.generic.ContextMixin
```

```
    get_context_data (**context)
```

Constructs and returns a dict for use as context in a rendered response. This method should only be used when directly providing context intended to be used in a response. There should be no side-effects introduced here.

If you need to provide a class-member set of shared context for use in an inherited mixin chain, then you should override `dispatch_request()` and assign that shared context before calling the *super* implementation of that method.

Parameters **context** – An optional dict for use in a rendered response.

Returns A dict

```
class fifty_flask.views.generic.TemplateResponseMixin
```

A mixin for rendering a template.

```
    get_template (**context)
```

Gets the template that needs to be rendered.

```
    render_response (**context)
```

Renders a template with the provided context. If a pjax template is defined, it will attempt to serve it if the request headers indicated that a PJAX response is needed.

```
template_name = None
    The name of the template to render

class fifty_flask.views.generic.TemplateMixin
    A template response with context.

class fifty_flask.views.generic.RedirectMixin

    get_redirect_endpoint (**context)
        Returns the endpoint to redirect to with the provided context.

    get_redirect_url (**context)
        Returns a URL that the view will redirect to by default when the form is validated.

    redirect (**context)
        Redirect to another URL with the provided context getting passed to url_for

class fifty_flask.views.generic.FormMixin
    A mixin for processing a form.

    form_cls = None
        A form class used to process POST data. Must inherit from flask-wtf form

    form_invalid (form, **context)
        If a form is not valid, the default behavior is to render the template and including the form as context.

    form_valid (form, **context)
        Default behavior on form validation is to simply redirect to the URL returned from self.get_success_url()

    get_form ()
        Instantiates a new form from the form class and any arguments that are defined for its constructor through
        self.get_form_kwargs()

    get_form_cls ()
        Default behavior is to return the form_cls defined on this instance.

    get_form_kwargs ()
        Any arguments that need to be passed to the form class constructor should be defined here and returned as
        a dict.

    get_form_obj ()
        The form object to pre-populate the form with.

    process_form ()
        Workflow for processing a form, from flask-wtf. If the request method is POST and the form is validated,
        it returns the result of self.form_valid(form). If the form data is invalid, or the request method is GET, it
        returns the result of self.form_invalid(form).

    render_response (**context)
        Renders a response, optionally with the provided context.

        Parameters context – An optional dict for use in a rendered response.

    validate (form)
        Validates a form.

class fifty_flask.views.generic.JsonResponseMixin

    render_response (**context)
        Returns a JSON response with the provided context.
```


class fifty_flask.views.generic.JsonMixin

A json response with context.

get_context_data (**context)

Constructs and returns a dict for use as context in a rendered response. This method should only be used when directly providing context intended to be used in a response. There should be no side-effects introduced here.

If you need to provide a class-member set of shared context for use in an inherited mixin chain, then you should override `dispatch_request()` and assign that shared context before calling the *super* implementation of that method.

Parameters **context** – An optional dict for use in a rendered response.

Returns A dict

render_response (**context)

Returns a JSON response with the provided context.

class fifty_flask.views.generic.MimeTypeResponseMixin

dispatch_request (*args, **kwargs)

Returns a custom Response object, using the result of the response from the dispatched request as well as any custom response parameters.

get_mimetype ()

Returns a mimetype.

get_response_cls ()

Returns a Response class.

get_response_kwargs (response)

If a mimetype is specified, include it when instantiating the Response instance.

response_cls

alias of flask.wrappers.Response

1.2 Views

class fifty_flask.views.generic.GenericView

The base generic view class.

classmethod **add_url_rule** (app, rule, endpoint, view_func=None, **options)

Convenience for registering this view on an app or blueprint, responding to the provided rule and given the specified endpoint name.

Parameters

- **app** – A flask app or blueprint
- **rule** – The url pattern to match
- **endpoint** – The name of this route, for use with `url_for()`
- **view_func** – A reference to the previous view func, if matching multiple routes
- **options** – Optional arguments passed directly into `Flask.add_url_rule()` or `Blueprint.add_url_rule()` depending on if *app* is a Flask or Blueprint instance

classmethod `add_url_rules` (*app*, *rules*, *endpoint*, *view_func=None*, ***options*)

If there needs to be multiple endpoints mapped to a single view, this class-method should be used in order to guarantee the `view_func` is the same. Takes the same arguments as `add_url_rule()`, with the only exception being that *rules* is a list of patterns.

Parameters *rules* – A list of url patterns to match

classmethod `as_view` (*name*, **class_args*, ***class_kwargs*)

Converts the class into an actual view function that can be used with the routing system. Internally this generates a function on the fly which will instantiate the `View` on each request and call the `dispatch_request()` method on it.

The arguments passed to `as_view()` are forwarded to the constructor of the class.

dispatch_request (**args*, ***kwargs*)

Sets `self.args` and `self.kwargs` from the request for convenience, in case there is a context in the code where the data is not directly passed in, but you need access to it.

class `fifty_flask.views.generic.TemplateView`

A view that renders a template on a GET request.

classmethod `add_url_rule` (*app*, *rule*, *endpoint*, *view_func=None*, ***options*)

Convenience for registering this view on an app or blueprint, responding to the provided rule and given the specified endpoint name.

Parameters

- **app** – A flask app or blueprint
- **rule** – The url pattern to match
- **endpoint** – The name of this route, for use with `url_for()`
- **view_func** – A reference to the previous view func, if matching multiple routes
- **options** – Optional arguments passed directly into `Flask.add_url_rule()` or `Blueprint.add_url_rule()` depending on if *app* is a Flask or Blueprint instance

classmethod `add_url_rules` (*app*, *rules*, *endpoint*, *view_func=None*, ***options*)

If there needs to be multiple endpoints mapped to a single view, this class-method should be used in order to guarantee the `view_func` is the same. Takes the same arguments as `add_url_rule()`, with the only exception being that *rules* is a list of patterns.

Parameters *rules* – A list of url patterns to match

classmethod `as_view` (*name*, **class_args*, ***class_kwargs*)

Converts the class into an actual view function that can be used with the routing system. Internally this generates a function on the fly which will instantiate the `View` on each request and call the `dispatch_request()` method on it.

The arguments passed to `as_view()` are forwarded to the constructor of the class.

dispatch_request (**args*, ***kwargs*)

Sets `self.args` and `self.kwargs` from the request for convenience, in case there is a context in the code where the data is not directly passed in, but you need access to it.

get (**args*, ***kwargs*)

Gets the context data and renders a template with it.

get_context_data (***context*)

Adds PJAX information to the context, for use in rendered templates.

get_pjax_template_name (***context*)

Gets the PJAX-friendly template that could potentially be rendered.

get_template (**context)
Renders a PJAX-friendly response.

is_pjax_request
Returns True if it's a PJAX request, False otherwise.

render_response (**context)
Renders a template with the provided context. If a pjax template is defined, it will attempt to serve it if the request headers indicated that a PJAX response is needed.

class fifty_flask.views.generic.RedirectView

classmethod **add_url_rule** (app, rule, endpoint, view_func=None, **options)
Convenience for registering this view on an app or blueprint, responding to the provided rule and given the specified endpoint name.

Parameters

- **app** – A flask app or blueprint
- **rule** – The url pattern to match
- **endpoint** – The name of this route, for use with `url_for()`
- **view_func** – A reference to the previous view func, if matching multiple routes
- **options** – Optional arguments passed directly into `Flask.add_url_rule()` or `Blueprint.add_url_rule()` depending on if *app* is a Flask or Blueprint instance

classmethod **add_url_rules** (app, rules, endpoint, view_func=None, **options)
If there needs to be multiple endpoints mapped to a single view, this class-method should be used in order to guarantee the view_func is the same. Takes the same arguments as `add_url_rule()`, with the only exception being that *rules* is a list of patterns.

Parameters **rules** – A list of url patterns to match

classmethod **as_view** (name, *class_args, **class_kwargs)
Converts the class into an actual view function that can be used with the routing system. Internally this generates a function on the fly which will instantiate the View on each request and call the `dispatch_request()` method on it.

The arguments passed to `as_view()` are forwarded to the constructor of the class.

dispatch_request (*args, **kwargs)
Sets self.args and self.kwargs from the request for convenience, in case there is a context in the code where the data is not directly passed in, but you need access to it.

get (*args, **kwargs)
Adds a generic view for redirecting to another endpoint.

get_context_data (**context)
Constructs and returns a dict for use as context in a rendered response. This method should only be used when directly providing context intended to be used in a response. There should be no side-effects introduced here.

If you need to provide a class-member set of shared context for use in an inherited mixin chain, then you should override `dispatch_request()` and assign that shared context before calling the *super* implementation of that method.

Parameters **context** – An optional dict for use in a rendered response.

Returns A dict

get_redirect_endpoint (**context)

Returns the endpoint to redirect to with the provided context.

get_redirect_url (**context)

Returns a URL that the view will redirect to by default when the form is validated.

redirect (**context)

Redirect to another URL with the provided context getting passed to `url_for`

class fifty_flask.views.generic.**ProcessFormView**

classmethod **add_url_rule** (app, rule, endpoint, view_func=None, **options)

Convenience for registering this view on an app or blueprint, responding to the provided rule and given the specified endpoint name.

Parameters

- **app** – A flask app or blueprint
- **rule** – The url pattern to match
- **endpoint** – The name of this route, for use with `url_for()`
- **view_func** – A reference to the previous view func, if matching multiple routes
- **options** – Optional arguments passed directly into `Flask.add_url_rule()` or `Blueprint.add_url_rule()` depending on if *app* is a Flask or Blueprint instance

classmethod **add_url_rules** (app, rules, endpoint, view_func=None, **options)

If there needs to be multiple endpoints mapped to a single view, this class-method should be used in order to guarantee the `view_func` is the same. Takes the same arguments as `add_url_rule()`, with the only exception being that *rules* is a list of patterns.

Parameters **rules** – A list of url patterns to match

classmethod **as_view** (name, *class_args, **class_kwargs)

Converts the class into an actual view function that can be used with the routing system. Internally this generates a function on the fly which will instantiate the `View` on each request and call the `dispatch_request()` method on it.

The arguments passed to `as_view()` are forwarded to the constructor of the class.

dispatch_request (*args, **kwargs)

Sets `self.args` and `self.kwargs` from the request for convenience, in case there is a context in the code where the data is not directly passed in, but you need access to it.

form_invalid (form, **context)

If a form is not valid, the default behavior is to render the template and including the form as context.

form_valid (form, **context)

Default behavior on form validation is to simply redirect to the URL returned from `self.get_success_url()`

get_form ()

Instantiates a new form from the form class and any arguments that are defined for its constructor through `self.get_form_kwargs()`

get_form_cls ()

Default behavior is to return the `form_cls` defined on this instance.

get_form_kwargs ()

Any arguments that need to be passed to the form class constructor should be defined here and returned as a dict.

get_form_obj()

The form object to pre-populate the form with.

post(*args, **kwargs)

Creates a form and validates it. If validation is successful, it will call `self.form_valid(form)`. Otherwise, it will call `self.form_invalid(form)`. The default behavior for each of these methods is documented in the `FormMixin` class.

process_form()

Workflow for processing a form, from flask-wtf. If the request method is POST and the form is validated, it returns the result of `self.form_valid(form)`. If the form data is invalid, or the request method is GET, it returns the result of `self.form_invalid(form)`.

render_response(context)**

Renders a response, optionally with the provided context.

Parameters context – An optional dict for use in a rendered response.

validate(form)

Validates a form.

class fifty_flask.views.generic.FormView

classmethod add_url_rule(app, rule, endpoint, view_func=None, **options)

Convenience for registering this view on an app or blueprint, responding to the provided rule and given the specified endpoint name.

Parameters

- **app** – A flask app or blueprint
- **rule** – The url pattern to match
- **endpoint** – The name of this route, for use with `url_for()`
- **view_func** – A reference to the previous view func, if matching multiple routes
- **options** – Optional arguments passed directly into `Flask.add_url_rule()` or `Blueprint.add_url_rule()` depending on if *app* is a Flask or Blueprint instance

classmethod add_url_rules(app, rules, endpoint, view_func=None, **options)

If there needs to be multiple endpoints mapped to a single view, this class-method should be used in order to guarantee the `view_func` is the same. Takes the same arguments as `add_url_rule()`, with the only exception being that *rules* is a list of patterns.

Parameters rules – A list of url patterns to match

classmethod as_view(name, *class_args, **class_kwargs)

Converts the class into an actual view function that can be used with the routing system. Internally this generates a function on the fly which will instantiate the `View` on each request and call the `dispatch_request()` method on it.

The arguments passed to `as_view()` are forwarded to the constructor of the class.

dispatch_request(*args, **kwargs)

Sets `self.args` and `self.kwargs` from the request for convenience, in case there is a context in the code where the data is not directly passed in, but you need access to it.

flash_invalid(form, **context)

Flashes the failure (message, category)

flash_valid(form, **context)

Flashes the success (message, category)

form_invalid (*form*, ****context**)

Default behavior when form is invalid is to optionally flash and execute default behavior from ProcessFormView.

form_valid (*form*, ****context**)

Default behavior on form validation is to optionally flash() and redirect.

get (**args*, ****kwargs**)

Creates a form and includes it as context to a template that will be rendered.

get_context_data (****context**)

Adds PJAX information to the context, for use in rendered templates.

get_flash_invalid_message (*form*, ****context**)

Returns the popped failure (message, category) from context if it exists, otherwise returns the class default.

get_flash_valid_message (*form*, ****context**)

Returns the popped success (message, category) from context if it exists, otherwise returns the class default.

get_form ()

Instantiates a new form from the form class and any arguments that are defined for its constructor through self.get_form_kwargs()

get_form_cls ()

Default behavior is to return the form_cls defined on this instance.

get_form_kwargs ()

Any arguments that need to be passed to the form class constructor should be defined here and returned as a dict.

get_form_obj ()

The form object to pre-populate the form with.

get_pjax_template_name (****context**)

Gets the PJAX-friendly template that could potentially be rendered.

get_redirect_endpoint (****context**)

Returns the endpoint to redirect to with the provided context.

get_redirect_url (****context**)

Returns a URL that the view will redirect to by default when the form is validated.

get_template (****context**)

Renders a PJAX-friendly response.

is_pjax_request

Returns True if it's a PJAX request, False otherwise.

post (**args*, ****kwargs**)

Creates a form and validates it. If validation is successful, it will call self.form_valid(form). Otherwise, it will call self.form_invalid(form). The default behavior for each of these methods is documented in the FormMixin class.

process_form ()

Workflow for processing a form, from flask-wtf. If the request method is POST and the form is validated, it returns the result of *self.form_valid(form)*. If the form data is invalid, or the request method is GET, it returns the result of *self.form_invalid(form)*.

redirect (****context**)

Redirect to another URL with the provided context getting passed to url_for

render_response (***context*)

Renders a template with the provided context. If a pjax template is defined, it will attempt to serve it if the request headers indicated that a PJAX response is needed.

validate (*form*)

Validates a form.

class `fifty_flask.views.generic.AjaxView`

classmethod **add_url_rule** (*app, rule, endpoint, view_func=None, **options*)

Convenience for registering this view on an app or blueprint, responding to the provided rule and given the specified endpoint name.

Parameters

- **app** – A flask app or blueprint
- **rule** – The url pattern to match
- **endpoint** – The name of this route, for use with `url_for()`
- **view_func** – A reference to the previous view func, if matching multiple routes
- **options** – Optional arguments passed directly into `Flask.add_url_rule()` or `Blueprint.add_url_rule()` depending on if *app* is a Flask or Blueprint instance

classmethod **add_url_rules** (*app, rules, endpoint, view_func=None, **options*)

If there needs to be multiple endpoints mapped to a single view, this class-method should be used in order to guarantee the `view_func` is the same. Takes the same arguments as `add_url_rule()`, with the only exception being that *rules* is a list of patterns.

Parameters **rules** – A list of url patterns to match

classmethod **as_view** (*name, *class_args, **class_kwargs*)

Converts the class into an actual view function that can be used with the routing system. Internally this generates a function on the fly which will instantiate the View on each request and call the `dispatch_request()` method on it.

The arguments passed to `as_view()` are forwarded to the constructor of the class.

dispatch_request (**args, **kwargs*)

Sets `self.args` and `self.kwargs` from the request for convenience, in case there is a context in the code where the data is not directly passed in, but you need access to it.

get (**args, **kwargs*)

Sends a jsonified response from a GET request.

get_context_data (***context*)

Constructs and returns a dict for use as context in a rendered response. This method should only be used when directly providing context intended to be used in a response. There should be no side-effects introduced here.

If you need to provide a class-member set of shared context for use in an inherited mixin chain, then you should override `dispatch_request()` and assign that shared context before calling the *super* implementation of that method.

Parameters **context** – An optional dict for use in a rendered response.

Returns A dict

render_response (***context*)

Returns a JSON response with the provided context.

class fifty_flask.views.generic.AjaxFormView

Processes an AJAX form.

classmethod add_url_rule(*app*, *rule*, *endpoint*, *view_func=None*, ***options*)

Convenience for registering this view on an app or blueprint, responding to the provided rule and given the specified endpoint name.

Parameters

- **app** – A flask app or blueprint
- **rule** – The url pattern to match
- **endpoint** – The name of this route, for use with `url_for()`
- **view_func** – A reference to the previous view func, if matching multiple routes
- **options** – Optional arguments passed directly into `Flask.add_url_rule()` or `Blueprint.add_url_rule()` depending on if *app* is a Flask or Blueprint instance

classmethod add_url_rules(*app*, *rules*, *endpoint*, *view_func=None*, ***options*)

If there needs to be multiple endpoints mapped to a single view, this class-method should be used in order to guarantee the *view_func* is the same. Takes the same arguments as `add_url_rule()`, with the only exception being that *rules* is a list of patterns.

Parameters **rules** – A list of url patterns to match

classmethod as_view(*name*, **class_args*, ***class_kwargs*)

Converts the class into an actual view function that can be used with the routing system. Internally this generates a function on the fly which will instantiate the View on each request and call the `dispatch_request()` method on it.

The arguments passed to `as_view()` are forwarded to the constructor of the class.

dispatch_request (**args*, ***kwargs*)

Sets `self.args` and `self.kwargs` from the request for convenience, in case there is a context in the code where the data is not directly passed in, but you need access to it.

form_invalid (*form*, ***context*)

If a form is not valid, the default behavior is to render the template and including the form as context.

form_valid (*form*, ***context*)

Default behavior on form validation is to simply redirect to the URL returned from `self.get_success_url()`

get_context_data (*form=None*, ***context*)

Constructs and returns a dict for use as context in a rendered response. This method should only be used when directly providing context intended to be used in a response. There should be no side-effects introduced here.

If you need to provide a class-member set of shared context for use in an inherited mixin chain, then you should override `dispatch_request()` and assign that shared context before calling the *super* implementation of that method.

Parameters **context** – An optional dict for use in a rendered response.

Returns A dict

get_form ()

Instantiates a new form from the form class and any arguments that are defined for its constructor through `self.get_form_kwargs()`

get_form_cls ()

Default behavior is to return the `form_cls` defined on this instance.

get_form_kwargs ()

Any arguments that need to be passed to the form class constructor should be defined here and returned as a dict.

get_form_obj ()

The form object to pre-populate the form with.

post (*args, **kwargs)

Creates a form and validates it. If validation is successful, it will call `self.form_valid(form)`. Otherwise, it will call `self.form_invalid(form)`. The default behavior for each of these methods is documented in the `FormMixin` class.

process_form ()

Workflow for processing a form, from flask-wtf. If the request method is POST and the form is validated, it returns the result of `self.form_valid(form)`. If the form data is invalid, or the request method is GET, it returns the result of `self.form_invalid(form)`.

render_response (**context)

Returns a JSON response with the provided context.

validate (form)

Validates a form.

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

f

`fifty_flask.views.generic`, 3

A

`add_url_rule()` (`fifty_flask.views.generic.AjaxFormView` class method), 12
`add_url_rule()` (`fifty_flask.views.generic.AjaxView` class method), 11
`add_url_rule()` (`fifty_flask.views.generic.FormView` class method), 9
`add_url_rule()` (`fifty_flask.views.generic.GenericView` class method), 5
`add_url_rule()` (`fifty_flask.views.generic.ProcessFormView` class method), 8
`add_url_rule()` (`fifty_flask.views.generic.RedirectView` class method), 7
`add_url_rule()` (`fifty_flask.views.generic.TemplateView` class method), 6
`add_url_rules()` (`fifty_flask.views.generic.AjaxFormView` class method), 12
`add_url_rules()` (`fifty_flask.views.generic.AjaxView` class method), 11
`add_url_rules()` (`fifty_flask.views.generic.FormView` class method), 9
`add_url_rules()` (`fifty_flask.views.generic.GenericView` class method), 5
`add_url_rules()` (`fifty_flask.views.generic.ProcessFormView` class method), 8
`add_url_rules()` (`fifty_flask.views.generic.RedirectView` class method), 7
`add_url_rules()` (`fifty_flask.views.generic.TemplateView` class method), 6
`AjaxFormView` (class in `fifty_flask.views.generic`), 11
`AjaxView` (class in `fifty_flask.views.generic`), 11
`as_view()` (`fifty_flask.views.generic.AjaxFormView` class method), 12
`as_view()` (`fifty_flask.views.generic.AjaxView` class method), 11
`as_view()` (`fifty_flask.views.generic.FormView` class method), 9
`as_view()` (`fifty_flask.views.generic.GenericView` class method), 6

`as_view()` (`fifty_flask.views.generic.ProcessFormView` class method), 8
`as_view()` (`fifty_flask.views.generic.RedirectView` class method), 7
`as_view()` (`fifty_flask.views.generic.TemplateView` class method), 6

C

`ContextMixin` (class in `fifty_flask.views.generic`), 3

D

`dispatch_request()` (`fifty_flask.views.generic.AjaxFormView` method), 12
`dispatch_request()` (`fifty_flask.views.generic.AjaxView` method), 11
`dispatch_request()` (`fifty_flask.views.generic.FormView` method), 9
`dispatch_request()` (`fifty_flask.views.generic.GenericView` method), 6
`dispatch_request()` (`fifty_flask.views.generic.MimeTypeResponseMixin` method), 5
`dispatch_request()` (`fifty_flask.views.generic.ProcessFormView` method), 8
`dispatch_request()` (`fifty_flask.views.generic.RedirectView` method), 7
`dispatch_request()` (`fifty_flask.views.generic.TemplateView` method), 6

F

`fifty_flask.views.generic` (module), 3
`flash_invalid()` (`fifty_flask.views.generic.FormView` method), 9
`flash_valid()` (`fifty_flask.views.generic.FormView` method), 9
`form_cls` (`fifty_flask.views.generic.FormMixin` attribute), 4
`form_invalid()` (`fifty_flask.views.generic.AjaxFormView` method), 12
`form_invalid()` (`fifty_flask.views.generic.FormMixin` method), 4

`form_invalid()` (`fifty_flask.views.generic.FormView` method), 9
`form_invalid()` (`fifty_flask.views.generic.ProcessFormView` method), 8
`form_valid()` (`fifty_flask.views.generic.AjaxFormView` method), 12
`form_valid()` (`fifty_flask.views.generic.FormMixin` method), 4
`form_valid()` (`fifty_flask.views.generic.FormView` method), 10
`form_valid()` (`fifty_flask.views.generic.ProcessFormView` method), 8
`FormMixin` (class in `fifty_flask.views.generic`), 4
`FormView` (class in `fifty_flask.views.generic`), 9

G

`GenericView` (class in `fifty_flask.views.generic`), 5
`get()` (`fifty_flask.views.generic.AjaxView` method), 11
`get()` (`fifty_flask.views.generic.FormView` method), 10
`get()` (`fifty_flask.views.generic.RedirectView` method), 7
`get()` (`fifty_flask.views.generic.TemplateView` method), 6
`get_context_data()` (`fifty_flask.views.generic.AjaxFormView` method), 12
`get_context_data()` (`fifty_flask.views.generic.AjaxView` method), 11
`get_context_data()` (`fifty_flask.views.generic.ContextMixin` method), 3
`get_context_data()` (`fifty_flask.views.generic.FormView` method), 10
`get_context_data()` (`fifty_flask.views.generic.JsonMixin` method), 5
`get_context_data()` (`fifty_flask.views.generic.RedirectView` method), 7
`get_context_data()` (`fifty_flask.views.generic.TemplateView` method), 6
`get_flash_invalid_message()`
 (`fifty_flask.views.generic.FormView` method), 10
`get_flash_valid_message()`
 (`fifty_flask.views.generic.FormView` method), 10
`get_form()` (`fifty_flask.views.generic.AjaxFormView` method), 12
`get_form()` (`fifty_flask.views.generic.FormMixin` method), 4
`get_form()` (`fifty_flask.views.generic.FormView` method), 10
`get_form()` (`fifty_flask.views.generic.ProcessFormView` method), 8
`get_form_cls()` (`fifty_flask.views.generic.AjaxFormView` method), 12
`get_form_cls()` (`fifty_flask.views.generic.FormMixin` method), 4
`get_form_cls()` (`fifty_flask.views.generic.FormView` method), 10
`get_form_cls()` (`fifty_flask.views.generic.ProcessFormView` method), 8
`get_form_kwards()` (`fifty_flask.views.generic.AjaxFormView` method), 12
`get_form_kwards()` (`fifty_flask.views.generic.FormMixin` method), 4
`get_form_kwards()` (`fifty_flask.views.generic.FormView` method), 10
`get_form_kwards()` (`fifty_flask.views.generic.ProcessFormView` method), 8
`get_form_obj()` (`fifty_flask.views.generic.AjaxFormView` method), 13
`get_form_obj()` (`fifty_flask.views.generic.FormMixin` method), 4
`get_form_obj()` (`fifty_flask.views.generic.FormView` method), 10
`get_form_obj()` (`fifty_flask.views.generic.ProcessFormView` method), 8
`get_mimetype()` (`fifty_flask.views.generic.MimeTypeResponseMixin` method), 5
`get_pjax_template_name()`
 (`fifty_flask.views.generic.FormView` method), 10
`get_pjax_template_name()`
 (`fifty_flask.views.generic.TemplateView` method), 6
`get_redirect_endpoint()` (`fifty_flask.views.generic.FormView` method), 10
`get_redirect_endpoint()` (`fifty_flask.views.generic.RedirectMixin` method), 4
`get_redirect_endpoint()` (`fifty_flask.views.generic.RedirectView` method), 7
`get_redirect_url()` (`fifty_flask.views.generic.FormView` method), 10
`get_redirect_url()` (`fifty_flask.views.generic.RedirectMixin` method), 4
`get_redirect_url()` (`fifty_flask.views.generic.RedirectView` method), 8
`get_response_cls()` (`fifty_flask.views.generic.MimeTypeResponseMixin` method), 5
`get_response_kwards()` (`fifty_flask.views.generic.MimeTypeResponseMixin` method), 5
`get_template()` (`fifty_flask.views.generic.FormView` method), 10
`get_template()` (`fifty_flask.views.generic.TemplateResponseMixin` method), 3
`get_template()` (`fifty_flask.views.generic.TemplateView` method), 6

I
`is_pjax_request` (`fifty_flask.views.generic.FormView` attribute), 10

`is_pjax_request` (`fifty_flask.views.generic.TemplateView` attribute), 7

J

`JsonMixin` (class in `fifty_flask.views.generic`), 4

`JsonResponseMixin` (class in `fifty_flask.views.generic`), 4

M

`MimeTypeResponseMixin` (class in `fifty_flask.views.generic`), 5

P

`post()` (`fifty_flask.views.generic.AjaxFormView` method), 13

`post()` (`fifty_flask.views.generic.FormView` method), 10

`post()` (`fifty_flask.views.generic.ProcessFormView` method), 9

`process_form()` (`fifty_flask.views.generic.AjaxFormView` method), 13

`process_form()` (`fifty_flask.views.generic.FormMixin` method), 4

`process_form()` (`fifty_flask.views.generic.FormView` method), 10

`process_form()` (`fifty_flask.views.generic.ProcessFormView` method), 9

`ProcessFormView` (class in `fifty_flask.views.generic`), 8

R

`redirect()` (`fifty_flask.views.generic.FormView` method), 10

`redirect()` (`fifty_flask.views.generic.RedirectMixin` method), 4

`redirect()` (`fifty_flask.views.generic.RedirectView` method), 8

`RedirectMixin` (class in `fifty_flask.views.generic`), 4

`RedirectView` (class in `fifty_flask.views.generic`), 7

`render_response()` (`fifty_flask.views.generic.AjaxFormView` method), 13

`render_response()` (`fifty_flask.views.generic.AjaxView` method), 11

`render_response()` (`fifty_flask.views.generic.FormMixin` method), 4

`render_response()` (`fifty_flask.views.generic.FormView` method), 10

`render_response()` (`fifty_flask.views.generic.JsonMixin` method), 5

`render_response()` (`fifty_flask.views.generic.JsonResponseMixin` method), 4

`render_response()` (`fifty_flask.views.generic.ProcessFormView` method), 9

`render_response()` (`fifty_flask.views.generic.ResponseMixin` method), 3

`render_response()` (`fifty_flask.views.generic.TemplateResponseMixin` method), 3

`render_response()` (`fifty_flask.views.generic.TemplateView` method), 7

`response_cls` (`fifty_flask.views.generic.MimeTypeResponseMixin` attribute), 5

`ResponseMixin` (class in `fifty_flask.views.generic`), 3

T

`template_name` (`fifty_flask.views.generic.TemplateResponseMixin` attribute), 3

`TemplateMixin` (class in `fifty_flask.views.generic`), 4

`TemplateResponseMixin` (class in `fifty_flask.views.generic`), 3

`TemplateView` (class in `fifty_flask.views.generic`), 6

V

`validate()` (`fifty_flask.views.generic.AjaxFormView` method), 13

`validate()` (`fifty_flask.views.generic.FormMixin` method), 4

`validate()` (`fifty_flask.views.generic.FormView` method), 11

`validate()` (`fifty_flask.views.generic.ProcessFormView` method), 9