{% include things.that.render %}

- Write reusable UIs –

clemens.pleban@gmail.com

```
{% include "foo/bar.html" %}
```

This example includes the contents of the template whose name is contained in the variable **`template_name`**:

```
{% include template_name %}
```

Changed in Django 1.7:

The variable may also be any object with a **`render()`** method that accepts a context. This allows you to reference a compiled **`Template`** in your context.

An included template is rendered within the context of the template that includes it. This example produces the output **`"Hello, John"`**:

- Context: variable **`person`** is set to **`"john"`**.
- Template:

```
{% include "name_snippet.html" %}
```

- The **`name_snippet.html`** template:

```
{{ greeting }}, {{ person|default:"friend" }}!
```

Example 1 - List of objects

view

```
class ListView(TemplateView):

    def get_context_data(self, **kwargs):
        context = super(ListView, self).get_context_data(**kwargs)
        context.update({
            'items': (ListItemUI(o) for o in self.get_objects())
        })
        return context
```

template

```
{% for item in items %}
{{ item.name }}
  {% include item %}
{% endfor %}
```

Example 1 - List of objects

ui

```
class ListItemUI(object):
    def __init__(self, obj):
        self.obj = obj

    @property
    def name(self)
       return obj.name

    def get_context_data(self):
        return {'obj': obj}

    def get_template(self):
        info = self.obj._meta
        return loader.select_template(
            'ui/{}/{}/list_item.html'.format(info.app_label, info.model_name),
            'ui/list_item.html',
        )

    def render(self, context):
        template = self.get_template()
        context.update(self.get_context_data())
        return template.render(context)
```

Example 1 - List of objects

- You can switch templates based on objects.
- You can prepare the context for a list item.
- You can reuse the ui element on other pages (e.g. teaser, search results)

Example 2 - List of objects prefetched

view

```
class PizzaOverview(TemplateView):

    def get_toppings(self, objs):
        query = """
        SELECT * FROM (
            SELECT topping.*, pizza_topping.pizza_id, row_number() OVER
                (PARTITION BY pizza, ORDER BY pos) as row FROM pizza_topping
                JOIN topping ON (pizza_topping.topping_id = topping.id)
                WHERE pizza_topping.pizza_id in %s) s
            WHERE row <= 3
            ODER BY row ASC;
        """
        toppings = defaultdict(list)
        for topping in Topping.objects.raw(query, [tuple(o.id for o in objs)]):
            toppings[topping.pizza_id].append(topping)
        return toppings

    def get_context_data(self, **kwargs):
        context = super(ListView, self).get_context_data(**kwargs)
        objs = self.get_objects()
        context.update({
            'item_ui': PizzaListItem(self.get_toppings(objs)),
            'items': objs,
        })
        return context
```

Example 2 - List of objects prefetched

template

```
{% for item in items %}
 {% include item_ui with obj=item %}
{% endfor %}
```

ui

```
class PizzaListItem(object):
    def __init__(self, toppings):
        self.toppings = toppings

    def get_context_data(self, context):
        obj = context['obj']
        return {
            'toppings': self.toppings[obj.id]
        }

    def render(self, context):
        context.update(self.get_context_data(context))
        template = loader.get_template('ui/pizza_list_item.html')
        return template.render(context)
```

Example 2 - List of objects prefetched

- You don’t have to initialize an UI instance per item.
- You can prefetch data to tweak performance.

Example 3 - Form UI

view

```
class ObjectDetail(DetailView):

    def get(self, request, *args, **kwargs):
        self.object = self.get_object()
        comment_ui = CommentUI(self.object)
        context = self.get_context_data(comment_ui=comment_ui, **kwargs)
        return self.render_to_response(context)

    def post(self, request, *args, **kwargs):
        self.object = self.get_object()
        comment_ui = CommentUI(self.object)
        response = comment_ui.handle_post(request)
        if response:
            return response
        context = self.get_context_data(comment_ui=comment_ui, **kwargs)
        return self.render_to_response(context)
```

template

```
{% include comment_ui %}
```

Example 3 - Form UI

ui

```
class CommentUI(object):

    def __init__(self, obj):
        self.obj = obj
        self.form = None

    def get_context_data(self):
        return {'form': self.form or CommentForm(self.obj)}

    def render(self, context):
        context.update(self.get_context_data())
        return loader.get_template('ui/comment_form.html')

    def handle_post(self, request):
        if '_add_comment' in self.request.POST
            self.form = CommentForm(obj, request.POST)
            if self.form.is_valid():
                self.form.save()
                return HttpResponseRedirect('.')
```

Example 2 - List of objects prefetched

- Separate form handling from views.
- Use multiple forms in a single view.
- optional: return only the uis html for ajax responses.

thanks