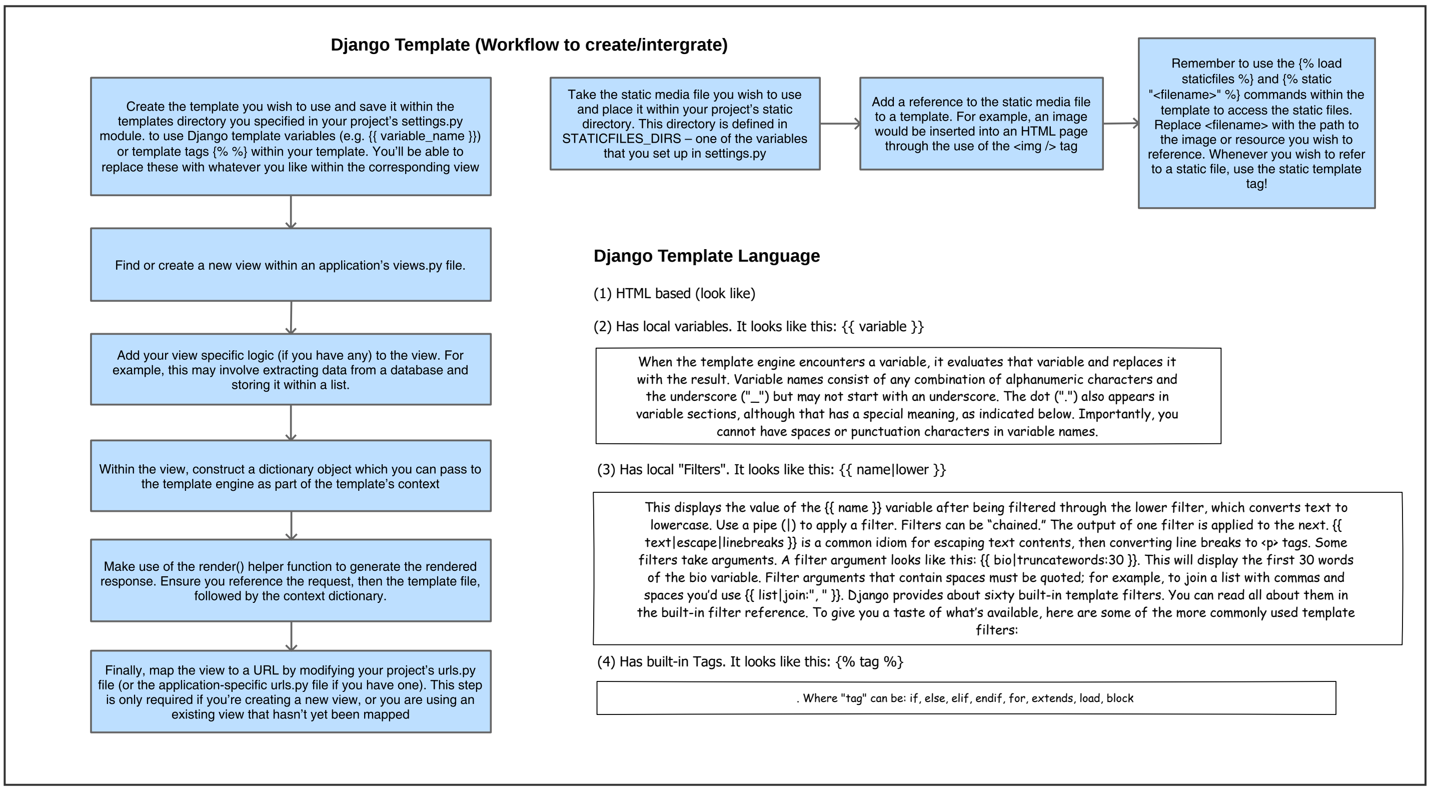
DjangoTemplateWorkflow



Django Template (Workflow to create/intergrate)

Create the template you wish to use and save it within the templates directory you specified in your project's settings.py module. to use Django template variables (e.g. {{ variable_name }}) or template tags {% %} within your template. You'll be able to replace these with whatever you like within the corresponding view

Find or create a new view within an application's views.py file.

Add your view specific logic (if you have any) to the view. For example, this may involve extracting data from a database and storing it within a list.

Within the view, construct a dictionary object which you can pass to the template engine as part of the template's context

Make use of the render() helper function to generate the rendered response. Ensure you reference the request, then the template file, followed by the context dictionary.

Finally, map the view to a URL by modifying your project's urls.py file (or the application-specific urls.py file if you have one). This step is only required if you're creating a new view, or you are using an existing view that hasn't yet been mapped

Take the static media file you wish to use and place it within your project's static directory. This directory is defined in STATICFILES_DIRS – one of the variables that you set up in settings.py

Add a reference to the static media file to a template. For example, an image would be inserted into an HTML page through the use of the <img /> tag

Remember to use the {% load staticfiles %} and {% static "<filename>" %} commands within the template to access the static files. Replace <filename> with the path to the image or resource you wish to reference. Whenever you wish to refer to a static file, use the static template tag!

Django Template Language

(1) HTML based (look like)

(2) Has local variables. It looks like this: {{ variable }}

When the template engine encounters a variable, it evaluates that variable and replaces it with the result. Variable names consist of any combination of alphanumeric characters and the underscore ("_") but may not start with an underscore. The dot (".") also appears in variable sections, although that has a special meaning, as indicated below. Importantly, you cannot have spaces or punctuation characters in variable names.

(3) Has local "Filters". It looks like this: {{ name|lower }}

This displays the value of the {{ name }} variable after being filtered through the lower filter, which converts text to lowercase. Use a pipe (|) to apply a filter. Filters can be "chained." The output of one filter is applied to the next. {{ text|escape|linebreaks }} is a common idiom for escaping text contents, then converting line breaks to <p> tags. Some filters take arguments. A filter argument looks like this: {{ bio|truncatewords:30 }}. This will display the first 30 words of the bio variable. Filter arguments that contain spaces must be quoted; for example, to join a list with commas and spaces you'd use {{ list|join:", " }}. Django provides about sixty built-in template filters. You can read all about them in the built-in filter reference. To give you a taste of what's available, here are some of the more commonly used template filters:

(4) Has built-in Tags. It looks like this: {% tag %}

. Where "tag" can be: if, else, elif, endif, for, extends, load, block