MIT AITI
Mobile, Python, Software Development

Lab 05: Templates

You will be adding templates to your blog app. These templates will allow you to view your blogs and comments. If you get stuck, take a look at these resources:

1. Lecture slides
2. Previous labs
3. Other group members
4. Django documentation
    a. Template Syntax overview: https://docs.djangoproject.com/en/dev/topics/templates/
    b. Template reference: https://docs.djangoproject.com/en/dev/ref/templates/
5. Google
6. Instructors

Setting up your template directory

1. Change to your blog's directory
```
$ cd ~/Desktop/myblog
```

2. Create directories where you will store templates
```
$ mkdir templates
$ mkdir templates/blog
```

3. Edit `settings.py`. Add the following lines to the top of the file

```
import os
PROJECT_ROOT = os.path.realpath(os.path.dirname(__file__))
```

Edit TEMPLATE_DIRS to be.

```
TEMPLATE_DIRS = (
    os.path.join(PROJECT_ROOT, '../templates')
    …
)
```

This will tell Django to look in the template directory you just created for templates to be mentioned in your blog app.

4. Change directory to templates/blog
```
$ cd templates/blog
```

Creating templates

5. Create a new template file base.html in templates/blog and insert the code

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>My Blog</title>
</head>
<body>
  <a href="/blog/list"><h1>My Blog</h1></a>
  <div id="content">
    {% block content %}
    {% endblock %}
  </div>
</body>
</html>
```

6. Create a new template file list.html in templates/blog. Edit list.html so that it extends base.html and overrides the content block with the lines

```
{% for blog in blog_list %}
        <h2><a href="/blog/detail/{{ blog.id }}/True">{{ blog.title
}}</a></h2>
        <ul>
          <div>Created: {{ blog.created }}</div>
          <div>Last updated: {{ blog.updated }}</div>
          <div>{{ blog.body|linebreaks }}</div>
        </ul>
{% endfor %}
```

   Note that when extending, you must reference base.html as "blog/base.html" instead of "base.html". Why?

7. Create a new template file `detail.html` in `templates/blog`. The purpose of `detail.html` will be to display a single blog post and all of its comments. Edit `detail.html` so that it extends `base.html` and overrides the content block in order to display

- A blog's title, body, date of creation, and date of last update. Assume that you have access to the relevant Blog object with the variable named `blog`. You can inspect the blog with `{{ blog.title }}`, `{{ blog.body }}`, etc.
- All comments associated with the blog. For each comment, show the comment's author, body, date of creation, and date of last update. Assume that you have access to a variable `comment_list` which is a list of all Comment objects associated with the blog. Each Comment object `c` in `comment_list` can be inspected with `{{ c.author }}`, `{{ c.body }}`, etc. Hint: do a for loop over `comment_list` using the {% for …} tag.

8. Create a new template search.html in templates/blog. The purpose of search.html is to list all blogs that have a match against a search query. Edit search.html so that it extends base.html and overrides the content block in order to display

    - The search query term. Assume that it is stored in the variable `query`.
    - A list of the titles of all blogs that matched the query. Assume that you have a list of the Blog objects that matched in the variable `blog_list`.

Directing URLs to templates

9. Now that you've created templates, the next step is to declare when these templates are to be evaluated. You will do this by calling templates in the functions in views.py. Since these functions are called in response to regex matches in urls.py, you will effectively be mapping URLs to templates.

    Make sure that views.py has the following import statement

    ```
    from django.template import Context, loader
    ```

10. Edit the blog_list function in views.py to look like

    ```
    def blog_list(request, limit=100):
        blog_list = Blog.objects.all()
        t = loader.get_template('blog/list.html')
        c = Context({'blog_list':blog_list})
        return HttpResponse(t.render(c))
    ```

    The first line `blog_list = Blog.objects.all()` is the same as what you copied from Step 6 of Lab 4. The last three lines use new functions that you haven't seen before.

    - `loader.get_template('blog/list.html')` loads list.html

- `Context({'blog_list':blog_list})` maps the name 'blog_list' to the actual variable blog_list. This mapping will be used to evaluate the blog_list variable in list.html. In general, the Context function takes in a dictionary where the keys are names of variables to be used in the loaded template. This mapping is called a context.
- `return HttpResponse(t.render(c))` simply returns the HTML found by evaluating the template with respect to the context.

11. Go to http://localhost:8000/blog/list to see a list of your blog posts.

12. Edit the home function in views.py to look like

```
def home(request):
    t = loader.get_template('blog/base.html')
    c = Context(dict())
    return HttpResponse(t.render(c))
```

13. Go to http://localhost:8000/blog/ to see the home page of your blog. The page should simply contain a link to the list of your blog posts.

14. Edit the blog_detail function in views.py so that it loads detail.html and uses the mapping `{'blog':blog, 'comment_list':comment_list}`.

15. Go to http://localhost:8000/blog/list and then click on one of your blogs to see its details and comments.

16. Edit the blog_search function in views.py to load search.html and use the {'blog_list':blog_list, 'query':query}.

17. Let <term> be a word in the title of one of your blog posts. Go to http://localhost:8000/blog/search/<query> to see a list of blogs that matches a search for <query>.

18. (Cleaning up your code) Notice that you repeatedly use the same few lines when generating an HttpResponse. First you load the template, then you create a Context, and then you render the context in the template and return an HttpResponse. In software development, you never want to unnecessarily repeat code. Fortunately, Django provides a shortcut that combines these steps into a single function:

```
render_to_response( template_file, context_dictionary)
```

To use this shortcut: `from django.shortcuts import render_to_response`. For example, the last three lines of the `blog_list` function from above can be shortened to a single line:

```
render_to_response('blog/list.html', {'blog_list': blog_list})
```