Exercise 2.4: Django Views and Templates

Learning Goals

- Summarize the process of creating views, templates, and URLs
- Explain how the “V” and “T” parts of MVT architecture work
- Create a frontend page for your web application

Reflection Question

1. Do some research on Django views. In your own words, use an example to explain how Django views work.

   A Django view is a Python function (or class) that accepts a request and returns a response. These responses can be as simple as returning an HTML page or something more complex where it interacts with the database or accepts user input. Depending on what URL is coming from the application, Django will select the appropriate view. For example, if you’re on a webpage and you click on a link in the navigation, you’ll be navigated to the corresponding page. Depending on the URL associated with the page you navigated to, Django will find which view is mapped to it and the logic associated with that view (i.e. returning a static HTML page) is what is returned and what you, as the user, sees.

2. Imagine you’re working on a Django web development project, and you anticipate that you’ll have to reuse lots of code in various parts of the project. In this scenario, will you use Django function-based views or class-based views, and why?

   If you anticipate that you’ll need reusable parts, then it is recommended that class-based views are used instead of function-based. Because of their class-based nature, they cut down on the amount of code you must rewrite, versus function-based where customized logic/functionality is needed.

3. Read Django’s documentation on the Django template language and make some notes on its basics.

   Templates in Django are text files (e.g. HTML, XML, etc.) that contain, variables and tags. Variables in templates use the format `{{ variable }}`, so when the template encounters such – it will evaluate what is in the brackets and replace it with the appropriate value. Filters modify variables by using a pipe (`|`) to assert what filter the variable should go through (e.g. `{{ name|lower }}` will filter name through the lower filter). Tags use the format `{% tag %}` and are more complex than variables; rather than return a value, they can be used to perform loops, among other things. Inheritance in Django templates allow you to build a template skeleton that contain common elements and blocks that child templates can change.