Week 3

1. **Django Project Structure**

Create project with this command django-admin startproject <project\_name>. This will create a series of files in a directory with the project\_name.

File manage.py is used to execute commands with Django project

Within directory settings.py is used to hold configuration settings while urls.py is used to hold routes.

**To start server use command:**

* Python manage.py startserver

**To create new application:**

* Python manage.py startapp <app\_name>
* Add <app\_name> to setings.py under INSTALLED\_APPS

**How to add a view/route in an application**

* Within views.py, you can create a view by creating a new function and having it take an argument called request.
  + This request is the HTTP request sent by clients
  + The name of the function would be the name of the route
* Create a separate urls.py within the root of the application folder that was created
* Within urls.py create a list with variable name called urlpatterns
  + This list should contain all routes that exists for this app. Each element should therefore be 1 url path.
  + To create a url path, import path from django.urls library. Each path takes 2 arguments
* Link project wide urls.py and app urls.py

Note: Within 1 Django project can have multiple applications as different applications can be used for different things.

**How Django processes requests?**

Urls.py acts as a url dispatcher. To design URLs for an app, a Python module called URLconf (informally named) must be created. This is normally called urls.py. This module is pure Python code and is a mapping between URL path expression and views (python functions).

1. Django determines the root URLconf module to use. Ordinarily, this is the value of the [ROOT\_URLCONF](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-ROOT_URLCONF) setting [This is the urls.py contained at the project level], but if the incoming HttpRequest object has a [urlconf](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "django.http.HttpRequest.urlconf" \o "django.http.HttpRequest.urlconf) attribute (set by middleware), its value will be used in place of the [ROOT\_URLCONF](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-ROOT_URLCONF) setting.
2. Django loads Python module and looks for variable urlpatterns. This should be a sequence of of django.urls.path() and/or django.urls.re\_path() instances
3. Django runs through each URL pattern in order and stops at the first one that matches the requested URL matching against the attribute path\_info in either path() or re\_path() instances
4. Django imports and calls the given view – which is normally a Python function. The view gets passed the following arguments
   1. An instance of the HTTP request
   2. If the matched URL contains no named groups, then matches from regex are provided as positional arguments.
   3. The keyword arguments are made up of any named parts matched by the path expression that are provided, overridden by any arguments specified in the optional kwargs argument to [django.urls.path()](https://docs.djangoproject.com/en/3.2/ref/urls/" \l "django.urls.path" \o "django.urls.path) or [django.urls.re\_path()](https://docs.djangoproject.com/en/3.2/ref/urls/" \l "django.urls.re_path" \o "django.urls.re_path).
5. If no URL pattern matches, or if exception raised, Django invokes an error handling view.

**Django Best Practices for folder management**

1. Naming templates

In views.py we often want to return a rendered html template. In Django, we import the render package from django.shortcuts to use the render function. This function then takes 3 arguments, 1st a request which contains the client request, 2nd the link to the html template to be returned, 3rd optional argument which is a python dictionary containing variables to be passed in that html.

Note: The best practice for naming the template is as follows <appName>\nameOfHtml.

If done so then you need to create a template folder in application folder, and within that template folder you would need to add a folder with your appName and then a file inside that folder for nameOfHtml.

The above structure is done to make sure your project is organized and that if 2 templates have different names, they don’t accidentally get called as they are in different folders.

1. Creating links to static files

Static files such as css should be found in a folder called static within your application. This means at the application level, create a folder called static and add your css folder within that application. Rationale for doing so is static folder is the default location where django would look for files. Having a hardcoded location means easier to cache and increase performance. As static files aren’t dynamic i.e. they do not respond to

1. Creating multiple apps within a project

1 project can have multiple applications. Each app is basically a URL argument and within that URL argument you can have more URL arguments.

**Template languages in Django**

Django has templating language similar to jinja2. You can access a variable with {{ variableName }}

Has logic with {% %}

Templates also provide inheritance with the “extends” functionality. Important when designing your html to figure out which parts are boiler plate code that can easily be inherited for easy management.

When adding a link to a another route in Django, remember to note hardcode the link instead use the Django syntax {% url <urlName> %} so that whenever you change the url name in views.py you can automatically change all references in templates.

Project Wiki

1. Read through Markdown guide. Pay attention to:
   1. Headings -> indicated via how many # there is
   2. Bold text -> things between \*\* \*\* would be bold. Things between \_\_ \_\_ as well.
   3. Links -> Text wrapped in [ ] would be created as inline link. Then wrap the URL in ( ).
   4. Lists
2. Encyclopedia/utils contains useful methods. They are (a) list\_entries which returns a list of the names of all encyclopaedia entries currently saved (b) save\_entry which save a new entry given its title and some markdown content (c) get\_entry which retrieve an entry by its titles and returns Markdown contents or none if entry does not exists
3. Each entry saved as Markdown file inside entries directory

Entry Page

Index Page

Search

New Page

Edit Page

Random Page

Markdown to HTML Conversion

Things I learn

1. Logic between app-level urls.py + redirecting urls in forms + views.py
   1. How to pass arguments into urls from hrefs
      1. “{% url ‘nameOfView’ argument %}” this must be in href
      2. Remember also that in app level urls.py you will need to have a url pattern that fits the above. For example, the above means that there must be a pattern of ‘nameOfView’/argument in urls.py
      3. Also means that in views.py your view must take 1 argument
   2. How to redirect from 1 view to another and how to pass arguments this way
      1. HttpResponseRedirect(reverse(viewName, args=[])
2. Forms
   1. How to create Django forms
      1. In models.py you should have
   2. How to create default value in Django forms
      1. In built in Django, can use the initial argument
   3. How to pass arguments into Django forms
   4. How to style them with bootstrap
      1. At form definition not in templating
3. Thinking habits
   1. How to think about urls and plan them
      1. 3 places (a) urls.py app level (b) views.py (c) template -> Always think by saying what does your URL actually look like then the abstraction of that should be in urls.py then the views and template should reflect that
   2. How to think about html pages and forms in Django
      1. Think about functions then think about what templates can be reused
   3. How to google stuff about Django
4. How to add something other than HTML in Django templates
   1. Must use {{variableName | safe}} -> This is just a security feature in Django
5. Creating logic in Django templating language
   1. {{ variableName }}
   2. {% url “”%}
   3. {% for some in something %} …. {% endfor %}
   4. Still have if