**Creating a Django project**

**Getting started**

1. **Open pycharm and create a project folder**
   1. This will contain only the venv to begin
2. **Install django**
   1. pip install django
3. **create a django project**
   1. use the django-admin CLI to create project
   2. run django-admin –help to view available commands
   3. run django-admin startproject <project\_name> to create project
   4. This will create a directory containing the main django project files under the name of <project\_name> that was used
4. **manage.py**
   1. Now that the project was created, a manage.py file was also created and placed in the main directory (outside of the django project directory). This file will be used from now on to run commands instead of django-admin. The manage.py file runs commands specifically for the project it was built for.
5. **Initialize a git repo**
   1. git init
   2. create a .gitignore file in the root folder of the project (where manage.py is)
   3. to get a default .gitignore file, go to gitignore.io and type in the framework/language
   4. git add . (dot is all) && git commit -m “Initial commit”
   5. (dot is to add all, -m is to be followed by notes about the commit taking place)
   6. After entering command from step 4, all files (excluding files in .gitignore) are pushed to the local repo.
   7. Add a remote repo
      1. git remote add origin <https://github.com/beezkneez406/btre.git>
   8. git branch -M main
   9. Push to remote repo
      1. git push -u origin main
      2. then add user name for github (jbies)
      3. then type in PAT (personal access token)
      4. ghp\_PJJpHPYyaA0ZkCwRpVjPGwulXUVGFZ1c8LY1 (for btre project, no expiration)
6. **Run the development server**
   1. To run the dev server, type python manage.py runserver into the terminal
   2. With the server now running we can go to localhost:8000 to see our website
7. **Overview of django project files**
   1. **settings.py**
      1. BASE\_DIR, base directory of project
      2. SECURITY\_KEY, used for production deployment. This needs to be hidden eventually.
      3. DEBUG, set to True for development but needs to be set to False once deployed
      4. ALLOWED\_HOSTS, list of host domains that the website can serve
      5. INSTALLED\_APPS, default apps are here to begin but the apps we build or need for other functionality will be added here
      6. MIDDLEWARE, list of middleware django uses. A lot of what is listed here is for security.
      7. ROOT\_URLCONF, string that represents the full import path to url file
      8. TEMPLATES, used to generate html for application. Tells django where to look for templates and which options to use.
      9. WSGI\_APPLICATION, path that the wsgi object that djangos built in servers use
      10. DATABASES, where we configure our database parameters. Default is sqlite.
      11. AUTH\_PASSWORD\_VALIDATORS, set rules for the types of passwords used
      12. STATIC\_URL, path used for static files (CSS, javascript, etc.)
   2. **urls.py**
      1. routing file
      2. urlpatterns list is where the path and view methods link should go
   3. **wsgi.py**
      1. Stands for web server gateway interface
      2. how web servers interact with web applications.
      3. deals with hosting site so other people can access it

**Creating an app**

1. **Create a django app**
   1. To create an app use the manage.py script
   2. python manage.py startapp <app\_name>
   3. This creates a new directory named <app\_name>, this will include:
      1. a directory called migrations for any migrations that are created
      2. an *\_\_init\_\_.py,* so the app is treated as a package
      3. *admin.py* for adding pieces of the app to the admin area
      4. *apps.py,* inculdes the class of the configuration for the app
      5. *models.py,* this is where the models for the app is created. Models are a description of the data to be stored (more or less).
      6. *tests.py,* used for running tests
      7. *views.py,* this is where we will create methods/functions that will be linked to urls
2. **Add the app to settings**
   1. navigate to settings.py under the project folder.
   2. Under the INSTALLED\_APPS list, add a new entry ‘<app\_name>.apps.<app\_name>Config’
   3. The new entry consists of the app name, then apps, then the class of the configuration from the apps.py file within the app we are adding.
3. **Create urls.py file in the app**
   1. Inside the app we need to create a urls.py file. When we modify the urls.py file under the project folder, we will include this file to point to the urls for this app.
   2. Make necessary imports
      1. from django.urls import path
      2. from . Import views
   3. Add urlpatterns list
      1. urlpatterns = []
   4. inside the urlpatterns, we need to add some paths. Using the path method that is imported add some urls to the list.
      1. Path(‘<path>’, views.<name\_of\_method\_in\_views>, name=’<name\_to\_access\_this\_path’>)
4. **Create a method inside views.py to link to**
   1. Add a function inside the views.py file. The name of the function needs to match the parameter views.<name\_of\_method\_in\_views> that is passed into the path function in urls.py.
5. **Add urls.py from app to urls.py inside project folder**
   1. make sure the “include” method is imported from django.urls
   2. Add a path to the urlpatterns
      1. path(‘<url>’, include(‘<app\_name>.urls’))
      2. leave <url> inside path function blank if you want the home page to be [www.website.com/](http://www.website.com/)<home>

**Templates and Base layout**

1. **Tell django where to look for templates**
   1. First step is to let django know where it can find our templates.
   2. Inside the settings.py file under the project folder, navigate to the TEMPLATES dictionary.
   3. Under DIRS we need to add the path to the directory where we will keep templates.
   4. Templates will be kept in the main folder or root foler (where manage.py is).
   5. To point to that location add the following to the DIRS list
      1. os.path.join(BASE\_DIR, ‘templates’)
      2. BASE\_DIR is defined at the top of settings.py as the path to the root directory
      3. may need to import os before step 1
2. **Add the templates directory**
   1. Inside the root folder, add a directory named ‘templates’
   2. inside the templates directory it is a good idea to create subdirectories with the names of each app to store templates separately.
3. **Add templates (html files) to these directories** 
   1. Once the templates have been created or added to the templates directory/s, we need to add the paths to the urlpatterns in the url.pys file for that specific app.
   2. Add index.html, and about.html templates
4. **Edit the urls.py file**
   1. Add the path() functions for the templates
      1. path(‘’, views.index, name=’index’)
      2. path(‘about’, views.about, name=’about’)
5. **Add view method for templates**
   1. Any newly created templates that were added to the urls.py file need view methods to go with them.
   2. Add/edit the views.py file to include view methods for the new templates
   3. Rather than return Httpresponse, we want to return the ‘render’ method
   4. render takes in two parameters (for now), request and the location of the template to render
6. **Create a base.html file**
   1. In the root of the templates folder, create base.html
   2. This will be “extended” on all other templates
   3. Inside the body tag of the base.html file, add jinja block {% block content %} {% endblock %}
      1. If pycharm project wasn’t created using django flag, the settings of the project will need to be modified to enable django support and also use jinja2 as the template language.
   4. Add the {% extends ‘base.html’ %} line to the top of all templates.

**Static files**

1. **Create static folder**
   1. Inside the django project folder (PycharmProjectFolder > django project) create a new folder named ‘static’
   2. Add subdirectories to the static folder for any static content (css, js, fonts, img)
2. **Add STATIC\_ROOT, and STATIC\_DIRS**
   1. navigate to settings.py inside the django project folder and add the following above/below STATIC\_URL.
      1. STATIC\_ROOT = os.path.join(BASE\_DIR, ‘static’)
      2. STATICFILES\_DIRS = [os.path.join(BASE\_DIR, ‘<project\_dir\_name>/static’)
3. **Run collectstatic**
   1. using manage.py run the collectstatic command, this will find any static files and create a static folder wherever we said was the STATIC\_ROOT directory.
   2. python manage.py collectstatic
4. **Add static to .gitignore**
   1. We do not want to push static files to our repo
   2. open .gitignore and add “/static”

**Bootstrap**

1. **Open the bootstrap theme in a separate text editor (sublime text)**
   1. Opening in a separate text editor isn’t crucial, but it helps to not get confused between project html and bootstrap theme html.
2. **Start by copying css and script links**
   1. These can be found in the header tag and the bottom of the body tag
   2. open the base.html file and copy these over
3. **Change href for links**
   1. the href’s need to be changed to point to the correct location to find the css and scripts
   2. We need to load the static files by adding some jinja
   3. At the very top of the base.html file, insert {% load static %}
   4. Change all href attribute for each we need to also add some jinja
      1. href=”{% static ‘<location\_of\_css\_or\_script>’ %}”
      2. ex → href=”{% static ‘css/all.css’ %}”
4. **Add the top bar and nav bar**
   1. From the bootstrap theme index.html, copy the nav bar and top bar (any other html pieces that will be shared across entire site) and paste them into base.html.
5. **Create partials**
   1. Inside the templates folder, create a directory named partials
   2. create separate html files for anything we want as partials (this will declutter the base.html file)
   3. the convention for creating partial html files is to precede the name with an underscore \_
   4. Create the following
      1. \_topbar.html
      2. \_navbar.html
      3. \_footer.html
   5. From base.html, copy out each partial into their own files.
   6. Using jinja, bring in each partials
      1. {% include ‘partials/\_topbar.html’ %}
      2. {% include ‘partials/\_navbar.html’ %}
      3. {% include ‘partials/\_footer.html’ %}

**Setting up the markup (html)**

1. **Next step is adding html for each page**
2. **We do not want to edit base html, we need to edit individual templates**
   1. open index.html (home page) and add any needed html for the block content.
   2. Open any other html pages and copy any html needed for the block content
3. **Setting up page links**
   1. inside the href anchor tag href attribute <a href=””> we need to insert some jinja2 syntax to point to the templates we are wanting to link
   2. inside the href use {% url ‘<name\_of\_template>’ %}
   3. The <name\_of\_template> comes from whatever name is specified in the urls.py file for the specific app.
   4. In the urls.py file we used the path() method where we specify the name using the name=’’ parameter.
4. **Dynamic markup content**
   1. to display certain html based on conditions we can use if statements using jinja2
   2. For this example we are changing the highlighting of items (active not active attribute) in the navbar based on the request path.
   3. Edit the navbar html partial by editing the list items where the anchor tags are. We want certain list items to use the “active” class when that page is the current page and the other items to be grayed out when they are not the current page.
   4. Edit the list items as follows

{% if request.path == ‘<url path>’ %}

class=”nav-item active mr-3”

{% else %}

class=”nav-item mr-3”

{% endif %}

1. <url\_path> should be whatever the path is set to in urls.py
2. For this example index (‘/’), about (‘/about’), etc.