1. use VScode for this tutorial
2. get the code from the github
3. open the folder titled CHESAPEAKECOMMUNITYCONNECT in VScode
4. download/install python 3.11
5. use ctrl shift P or command shift P in VScode and search “Python: Create Environment”
6. select Venv
7. select 3.11 as a base
8. select requirements.txt as dependencies to install, let VScode create the virtual env
9. open the terminal in VS code and cd to the CHESAPEAKECOMMUNITYCONNECT if you're not there already.
   1. follow the instructions below depending on your machine
10. cd to cccSite with "cd cccSite"
11. enter "python manage.py makemigrations" and "python manage.py migrate"
12. then to view site enter "python manage.py runserver 8080"

**MAC:**

source .venv/bin/activate

cd cccSite

python manage.py makemigrations

python manage.py migrate

python manage.py runserver 8080

**Windows: remember to terminate your powershell process when done working**

Set-ExecutionPolicy Unrestricted -Scope Process

.venv\Scripts\Activate.ps1

pip install python-magic-bin

cd cccSite

python (or py) manage.py makemigrations python manage.py migrate

python (or py) manage.py runserver 8080

good tutorial <https://docs.djangoproject.com/en/5.0/intro/tutorial01/>

Django Notes

* There are a number of folders contained in the CHESAPEAKECOMMUNITYCONNECT folder.
  + the first cccSite is the project directory, which contains the cccSite/manage.py file that is used to update models in the database, run the server, add admin accounts, and more stuff, cccSite/cccSite is like a resources folder for the project. in it are templates and static files that do not belong to a particular django app. Global settings are managed here in the cccSite/cccSite/settings.py file and the root url configuration is defined in cccSite/cccSite/urls.py file. Idk why that stuff isn't just in the project folder but that's the way it is.
  + The other folders are django applications. These separate the site's functionality. right now we have
    - boiler, named for boilerplate, which is the catch all for "about us," "contact," and other miscellaneous things that are too small for their own apps.
    - account, which is for account management
    - mapviewer, for the map viewer
* Applications
  + Django applications are designed to be somewhat modular, so you in theory could reuse them across multiple projects.
  + Each application defines what its own urls point to, what the views do, what models they need, and how to display its own information.
* How I understand Django to work
  + Django uses **models**, **templates**, **views**, and **urls** to create a usable website
  + **models** are tables in a database, and define an object and its attributes. Posts would be a model, and a post would be made up of location and description and so on.
  + **urls** point to **views**. when a user enters a url, the root url config in combination with the relevant application's url config passes the user's http request to the stated **view**
  + **views** are the functionality of a given webpage. given an Http request, they take information contained in the request and do stuff with it or to it (store it in the database, update a database, choose what page to display based on it, etc. they must return some sort of response, for us that will be mostly render and redirect.
    - render takes the http request and a context dictionary that you can define in the view, and passes it to a template which django then displays based on what you passed to the template
  + **templates** are html documents that are like mad libs. they have a special markup language
    - {%block [name] %} {%endblock [name]%} in a templateA creates a content block that can be overwritten by a template that extends templateA
    - {{ [variable name] }} are variables that you define in the context that you provide when rendering a template
    - templates can use if statements and for loops and all sorts of stuff.
* There's lots of good django help available if you just add django to a google search
  + eg display image django
* Session management is implemented using django's built in sessions, check out the default view in account/views.py for an example of how to make a decision based on the permission level that a user holds