In this document, I'll be breaking down SIFSearch's current code so that you can look at how I organized the project and the main files that you should be editing.
Just a side note, this Django app will use the Algolia App to perform the searches. So, Algolia's search algorithm is being used rather than one I created myself.

- For reference, **here's a link to the GitHub repo for SIFSearch: https://github.com/KaushikVejju/SIFSearch**

First, we will take a look at the files under SIFSearch/sifsearchdemo/sifsearchdemo:
The main files that I under this path that you should be looking at are:

- **models.py**
- **views.py**
- **urls.py**
- **serializers.py**
- **admin.py**
- **index.py**

Here's an explanation of each:

- **models.py (super important)** defines the fundamental data of the application. We need this class as it will help us set up a database that stores the data and its different features.
    - The **main** model is a **SearchEntry.** SearchEntry models represent the Search Entries that users upload/create when they access SIFSearch. A SearchEntry has 5 fields: name, description, link, tag, and file
    - The second model is a **Tag**. Tag's present the category that users place a SearchEntry in. Tags will be used in the Search process as they help filter out Search Entries based on the tag that was given to them.

- **views.py** contains a set of functions that are invoked when different endpoints of the SIFSearch app are accessed.
    - The first function, **sif_search_home(request)** renders and returns the home.html page, and passes in a variable called currtags, which will be explained later in this doc.
    - Following this function are 3 functions that are invoked when API calls are made to the Django backend.
        - **Add_entry_link() is handles POST** requests when the user uploads an entry that contains a link. The request is serialized into a SearchEntry using an EntrySerializer, which will be explained in the **serializers.py** function. The Tag_Serializer is also used to serialize the tag that the user gives to the uploaded entry. If the entry is serialized properly, we then proceed to check if the tag already exists or not. If it does not exist, a new Tag is created in the Django Admin database. After this process, a new SearchEntry is added to the Django Admin database.
        - **Add_entry_file()** does the exact same thing as the function above, but it handles entries that contain a file as opposed to a link.

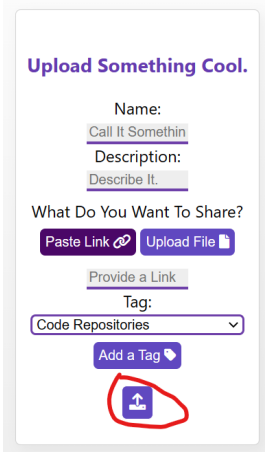
        - **update_entry()** handles PUT requests. As the name implies, it will update the entry's details (specifically the name and description-- for now) if they have been changed. This is done by searching for the original name of the entry, modifying its fields as needed, and then saving the changes. **NOTE:** THIS WILL NOT CREATE A NEW ENTRY IN THE DJANGO ADMIN DATABASE
- **urls.py** contains the endpoints of this Django application.
    - Here are the endpoints, and the functions in **views.py** that they map to
        - "" (yes, nothing) → this maps to the **sif_search_home()** function above, and will return the home page of SIFSearfch when the user enters http://127.0.0.1:8000/ in their browser window (may differ based on what your local host is)
        - "addentrylink/" → maps to the **add_entry_link()** function. **NOTE THIS IS AN API ENDPOINT**.
        - addentryfile/ → maps to the **add_entry_file()** function
        - update/ → maps to the **update_entry()** function
- **Serializers.py** defines how an API request body should be serialized. The API request body is in the form of Form Data, but the goal is to serialize this FormData into a Python object (in this case, a SearchEntry model or a Tag model) that can be used in the Django backend. For this project, there are **3 Serializer classes: EntrySerializer** (deals with SearchEntry's with links), **EntryFileSerializer** (deals with SearchEntry's with files), and **TagSerializer** (serializers tags).
- **admin.py** → when models are created, they need to be registered into the Django admin site. The only models we made are SearchEntry and Tag, so only these need to be registered.
- **index.py** → We've created models in Django, but these models need to be accessed by the Algolia API to handle all of the search logic. Because we are mainly searching for SearchEntries that users have uploaded, **the SearchEntry model is registered.**

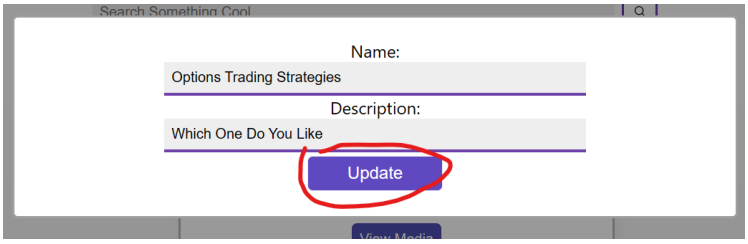
Under SIFSearch/sifsearchdemo/templates, the boiler plate html code is shown. I think it's worth at the entire, but there are two lines I just want to bring your attention to:

- line 2, which says **{% load static %}. Just know that this line is included so that the html code can work with the .css and .js files defined in SIFSearch/sifsearchdemo/static folder.**
- Line 69: which says {% for t in currtags %}
    - If you look at the views.py file in the **SIFSearch/sifsearchdemo/sifsearchdemo** folder, inside of the sif_search_home() function is a variable called currtags, which stores a list of all the Tag objects that have been defined. I used this variable so that when the html page is loaded, all of the tag names will be included in the Tag Dropdown list, shown below:

- 

Under SIFSearch/sifsearchdemo/static, you will find the style.css and scripts.js files

- **Style.css** is pretty straightforward, it just defines the style for the html elements
- **Scripts.js** is where the API calls are made (via the fetch() function) and the configurations for the instantSearch.js library are made (this is a JS library from Algolia that provides neat UI widgets for searching)
    - **I've commented on the entire file, but I recommend looking at the comments that detail the REST API calls. FYI, the API endpoints are defined at the top of the file.**
    - Just to give an idea, here is the UI the events that will cause API calls to be made:
        - 

            - **Will make API calls to the /addentryfile/ or /addentrylink/ endpoints, depending** if the user enters a link or a file
        - 

            - **Will make an API call to the /updateentry/ endpoint**, when the user modifies the fields of a Search Entry

Hope this guide helps. If you have any questions let me know.