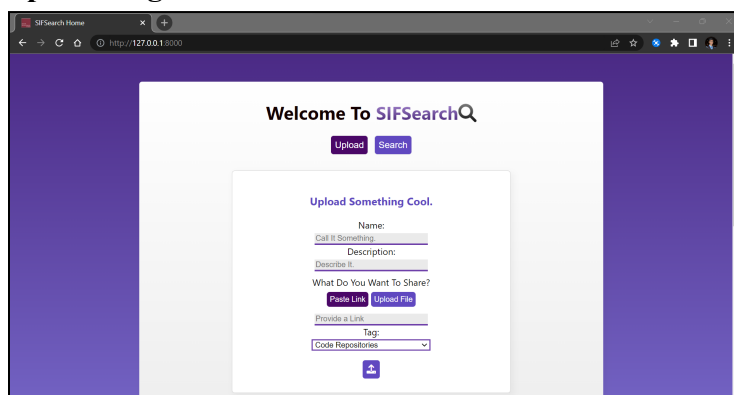


Part I: Introduction

This semester, we began our work on SIFSearch, a search engine designed exclusively for the Smith Investment Fund that enables club members to upload and search for media pertaining to quantitative finance. The goal of this internal tool is to provide members with immediate access to useful resources such as research articles, code repositories, and design documents without spending as much time to find them. This can in turn streamline the development process for many of SIF's other projects.

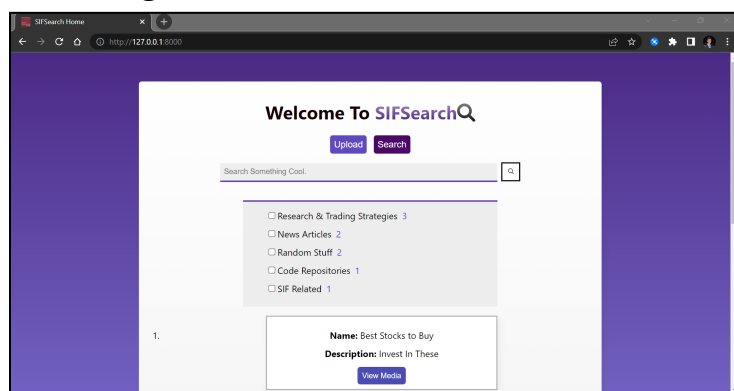
We are developing SIFSearch with the Django framework and are utilizing Algolia's Search API and InstantSearch.js library to handle the search logic and style the user interface. Here's a look at the UI for reference:

Upload Page



The screenshot shows the 'Upload' page of SIFSearch. The page has a purple header and footer. The main content area is white and contains a form titled 'Welcome To SIFSearchQ'. Below the title are two buttons: 'Upload' and 'Search'. The form itself is titled 'Upload Something Cool.' and contains several input fields: 'Name:' (with a placeholder 'Call it Something'), 'Description:' (with a placeholder 'Describe it'), 'What Do You Want To Share?' (with a placeholder 'Paste a Link'), 'Provide a Link:' (with a placeholder 'Code Repositories'), and a 'Tag:' dropdown menu. There are also two buttons: 'Upload File' and 'Upload'.

Search Page



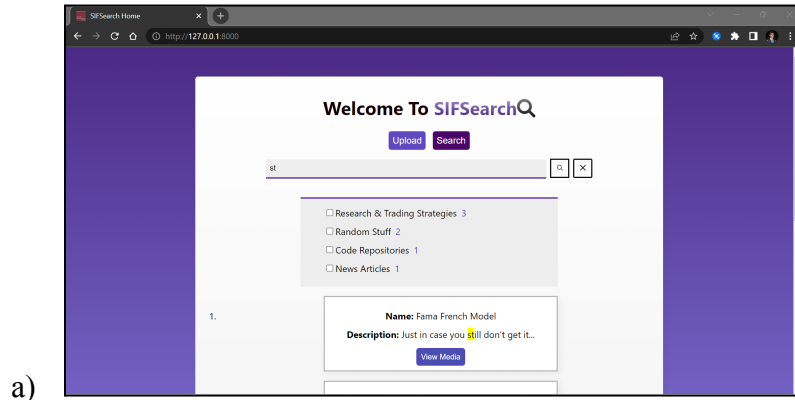
The screenshot shows the 'Search' page of SIFSearch. The page has a purple header and footer. The main content area is white and contains a search bar titled 'Welcome To SIFSearchQ'. Below the search bar are two buttons: 'Upload' and 'Search'. The search bar has a placeholder 'Search Something Cool.' and a search icon. Below the search bar is a list of search results. The first result is titled '1.' and has a 'Name: Best Stocks to Buy' and a 'Description: Invest In These'. There is a 'View Media' button below the description. The search results are categorized by checkboxes: 'Research & Trading Strategies 3', 'News Articles 2', 'Random Stuff 2', 'Code Repositories 1', and 'SIF Related 1'.

As seen above, there are two modes, or features to SIFSearch:

- 1) **Upload**, where users are able to enter details (title and description) about an article/form of media they have found and provide a link or upload a file. In addition, the user selects

a tag that categorizes the entry they are uploading, which helps refine the process of searching for entries.

- 2) **Search**, where users can find all of the entries that they and other club members have uploaded. By default, the search page will display all of the uploaded entries. Only when the user begins to make a query does it filter out other entries and show the desired one (this is handled by the InstantSearch.js library), as shown below. Users also have the option to select the tags in the refinement list that appears below the search bar. This will filter out entries as desired.



Now that we've covered what SIFSearch does, let's break down how we are developing it.

Part II: Implementation of SIFSearch

One of the most important components of SIFSearch are the entries that users are searching for. Internally, these search entries are implemented as Python classes known as models, which represent the data of this Django application. The fields of the model include the name, description, link to a search entry, and a file for the search entry. Shown below is the SearchEntry class:

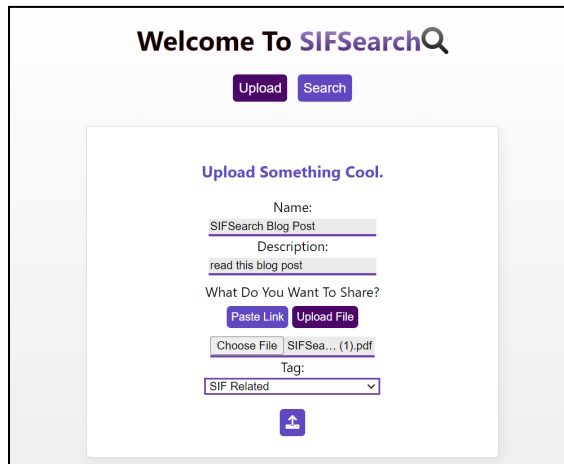
```
from django.db import models
class SearchEntry(models.Model):
    name = models.CharField(max_length=1000)
    description = models.CharField(max_length=1000)
    link = models.CharField(max_length=1000, default="")
    tag = models.CharField(max_length=1000, default="")
    file = models.FileField(upload_to="static", blank=True)
```

Once the SearchEntry model was defined, we registered this model into the Django Admin Site for this application. From here, we could add/store multiple SearchEntrys in the admin site's database.

Of course, rather than us having to login to the Django Admin site and manually add SearchEntry's ourselves, we wanted to be able to do this through the UI of SIFSearch. More specifically, when the user clicks the upload button, a POST request would be sent to and

handled by an API endpoint in the Django backend. This endpoint invokes a function that serializes the request body into a SearchEntry object and saves it to the Django Admin site's database. In addition, this function interfaces with Algolia's Search API to register the new entry and make it a searchable item. Here's an example of this process in action:

Completing the Form



The screenshot shows a web interface titled "Welcome To SIFSearchQ". Below the title are two buttons: "Upload" and "Search". The main content area is titled "Upload Something Cool." and contains the following fields and controls:

- Name:** A text input field containing "SIFSearch Blog Post".
- Description:** A text input field containing "read this blog post".
- What Do You Want To Share?:** A section with two buttons: "Paste Link" and "Upload File".
- File Selection:** Below the buttons, there is a "Choose File" button and a file name "SIFSea... (1).pdf".
- Tag:** A dropdown menu with "SIF Related" selected.
- Submit:** A blue button with a white upload icon at the bottom.

POST Request sent to “/addentryfile/” endpoint

Request URL: <code>http://127.0.0.1:8000/addentryfile/</code>
Request Method: POST
Status Code: 🟢 201 Created

Update on Django Admin Site

Note: Link field is empty since the user uploaded a file entry

SearchEntry object (108)	
Name:	<input type="text" value="SIFSearch Blog Post"/>
Description:	<input type="text" value="read this blog post"/>
Link:	<input type="text" value=""/>
Tag:	<input type="text" value="SIF Related"/>
File:	<div>Currently: <code>static/SIFSearch_Blog_Post_1_KiYofSy.pdf</code> <input type="checkbox"/> Clear</div> <div>Change: <input type="button" value="Choose File"/> No file chosen</div>

Update on Algolia

8	objectID	"108"
Q	description	"read this blog post"
Q	name	"SIFSearch Blog Post"
Q	tag	"SIF Related"
	link	""
	file	"static/SIFSearch_Blog_Post_1_KIYofSy.pdf"

Now that we've covered how the upload functionality works, we'll discuss how our application uses Algolia's Search API to search for the uploaded entries.

Fortunately for us, Algolia's Search API abstracts away all of the complexities of developing an efficient searching algorithm. To integrate this external API, all we needed to do was apply some configurations to our Django settings and register the SearchEntry model to Algolia. As mentioned above, we also needed to ensure that Algolia would be able to store new search entries as they were added by the upload form.

Once this was done, we utilized the built-in functions provided by Algolia's InstantSearch.js library to make changes to the UI. InstanceSearch.js provided us with a wide variety of UI components/widgets, such as search bar, hits page, pagination tool, and search refinement/filtration list. We included these components into our UI and applied our own styling to them, completing the look and functionality of SIFSearch's search mode (for now).

As of now, we have a basic MVP of SIFSearch. There are more features that we hope to add, and we'll be covering these in the following section.

Part III: Challenges & Future Plans

As we developed SIFSearch, one challenge we faced was becoming accustomed to the Django framework and following the Model-View-Controller design pattern Django is based off of. This became apparent when we began to utilize the Django REST Framework to set up API endpoints for our application. One issue that stands out was setting up Cross-site request forgery (CSRF) protection. Since we were making an AJAX request to our API endpoints, the process of using CSRF protection was a bit more complicated and involved having to retrieve a CSRF token from the browser cookie. Luckily, [Django's documentation](#) provided JavaScript code on doing this, which helped resolve this issue.

Another challenge was getting the uploading feature to work with different files (PDFS, .py files, etc). Although we defined a "file" field in our Django model, we were experiencing issues with serializing the POST request body into a SearchEntry object. To solve this, we first defined a custom serializer class ([Django documentation on this topic](#)) for the SearchEntry model. We also modified the request body that was sent to the Django backend API by defining the content type as form data as opposed to a JSON string. This was done by declaring an object of

JavaScript's FormData class, which allowed us to define key-value pairs that represented the content of the request body. From here, we simply passed this FormData object into a fetch() call to the Django backend, enabling the uploading feature to work for all media types.

As we progress into next semester, we look forward to implementing more features to SIFSearch. One feature that was discussed was giving users the ability to add their own tags to the search entries rather than selecting the ones that are already present. This will help group the entries into more specific categories that directly align with the user's needs. In addition to this, we hope to implement a feature that will enable users to edit search entries that they have uploaded rather than uploading an entirely different one. Overall, we are excited for next semester and look forward to enhancing SIFSearch's capabilities.