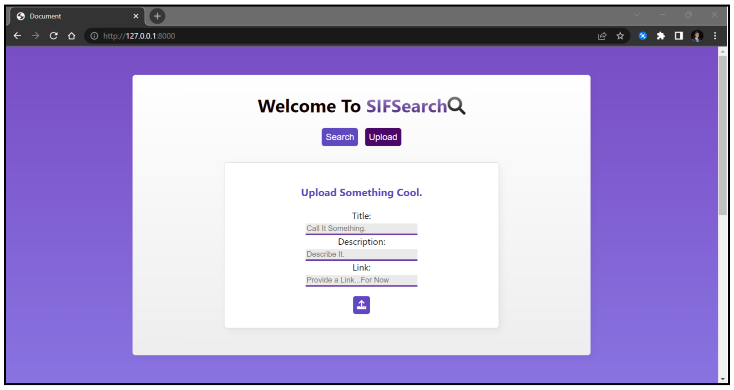
Kaushik Vejju & Shane Thakkar
Smith Investment Fund Blog Post: *SIFSearch*

Part I: Introduction

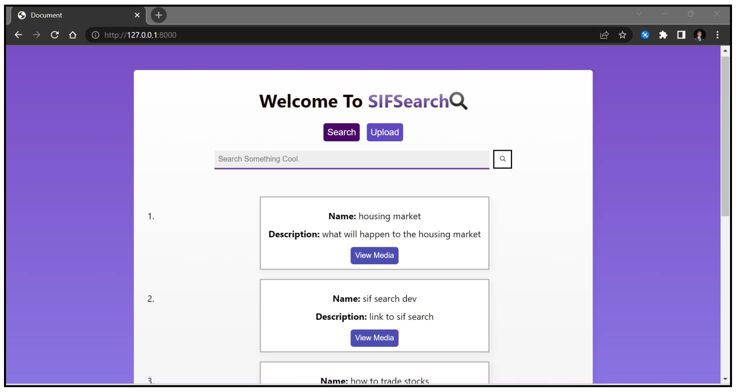
This semester, we began our work on SIFSearch, a search engine designed exclusively for the Smith Investment Fund that enables club members to upload and search for media pertaining to quantitative finance. The goal of this internal tool is to provide members with immediate access to useful resources 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



Search Page



1. **Name:** housing market
Description: what will happen to the housing market
View Media

2. **Name:** sif search dev
Description: link to sif search
View Media

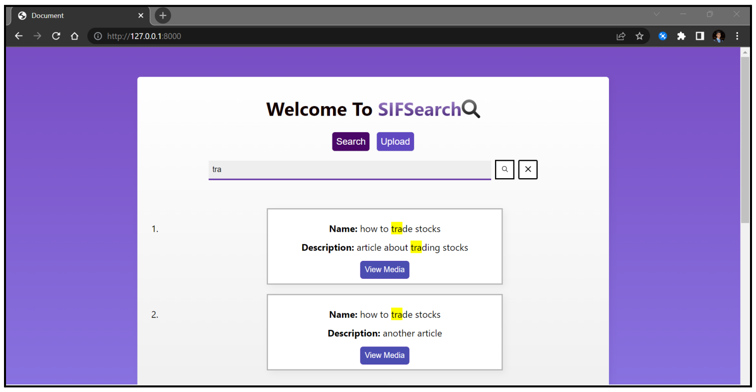
3. **Name:** how to trade stocks

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 to it. In future semesters, we aim to have the

upload form work with all file types, such as pdfs and jpegs, as opposed to only using a link to a website.

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:

a) 

tra

1. **Name:** how to trade stocks
Description: article about trading stocks

2. **Name:** how to trade stocks
Description: another article

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

Part II: Implementation of SIFSearch

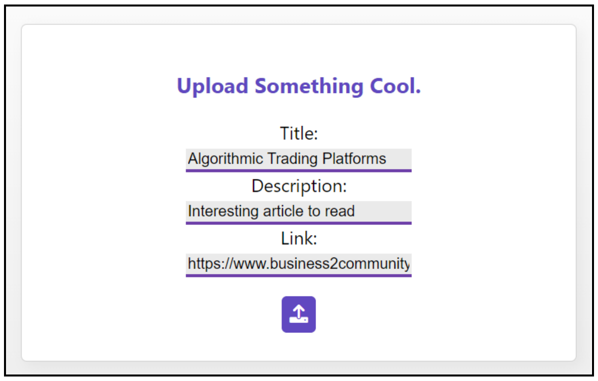
One of the most important components of SIFSearch are the entries that users are searching for. Internally, these search entries are implemented as Python classes known as models, which represent the data of this Django application. The fields of the model include the name, description, and link to a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

```
from django.db import models
class SearchEntry(models.Model):
    name = models.CharField(max_length=600)
    description= models.CharField(max_length=500)
    link = models.CharField(max_length=600)
```

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.

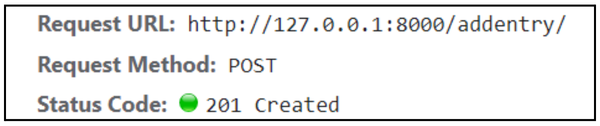
Of course, rather than us having to login to the Django Admin site and manually add SearchEntry's ourselves, we wanted to be able to do this through the UI of SIFSearch. More specifically, when the user clicks the upload button, a POST request would be sent to and handled by an API endpoint in the Django backend. This endpoint, which we called "/addentry/" , 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

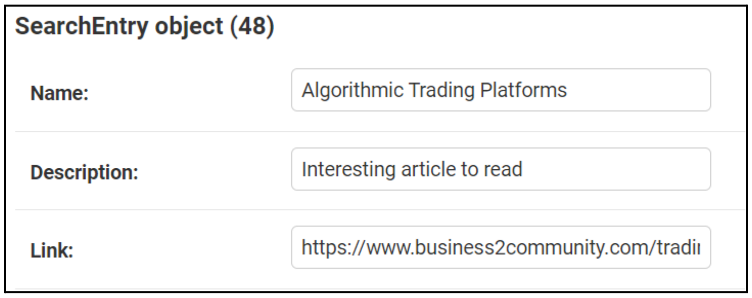

Upload Something Cool.

Title:	Algorithmic Trading Platforms
Description:	Interesting article to read
Link:	https://www.business2community

POST Request sent to "/addentry/" endpoint

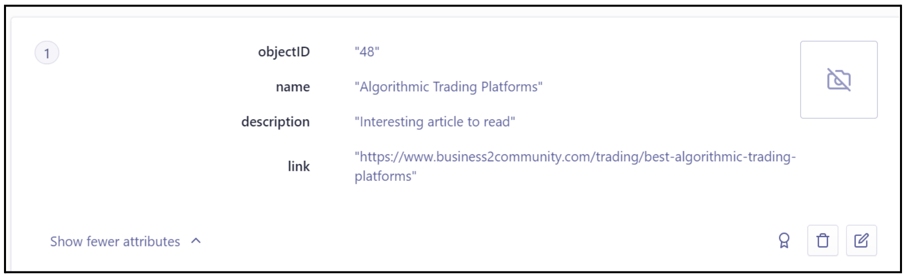

Request URL: `http://127.0.0.1:8000/addentry/`
Request Method: `POST`
Status Code: ● `201 Created`

Update on Django Admin Site


SearchEntry object (48)

Name:	Algorithmic Trading Platforms
Description:	Interesting article to read
Link:	https://www.business2community.com/tradi

Update on Algolia


1

objectID	"48"
name	"Algorithmic Trading Platforms"
description	"Interesting article to read"
link	"https://www.business2community.com/trading/best-algorithmic-trading-platforms"

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, and pagination tool. 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, the main 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 endpoint, 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 styling the front end, but more specifically incorporating the widgets provided by the InstantSearch.js library onto our user interface. During development, there were times where we needed to refactor our HTML template code in order to successfully include widgets like pagination and the hits page.
As we progress into next semester, we look forward to implementing more features to SIFSearch. One prominent feature is being able to upload all forms of media (pdfs, links, JPEG, .py files). This is easier said than done, and will require having to redefine our SearchEntry model and the template code for the upload form. Another feature we hope to include is a tag-based search. For instance, if many of the search entries have the same file type or they are articles from the same website, there can be tags that group related search entries with one another. Algolia supports this, but there is additional work on our end to have this feature implemented. Overall, We are excited for next semester and look forward to enhancing SIFSearch's capabilities.