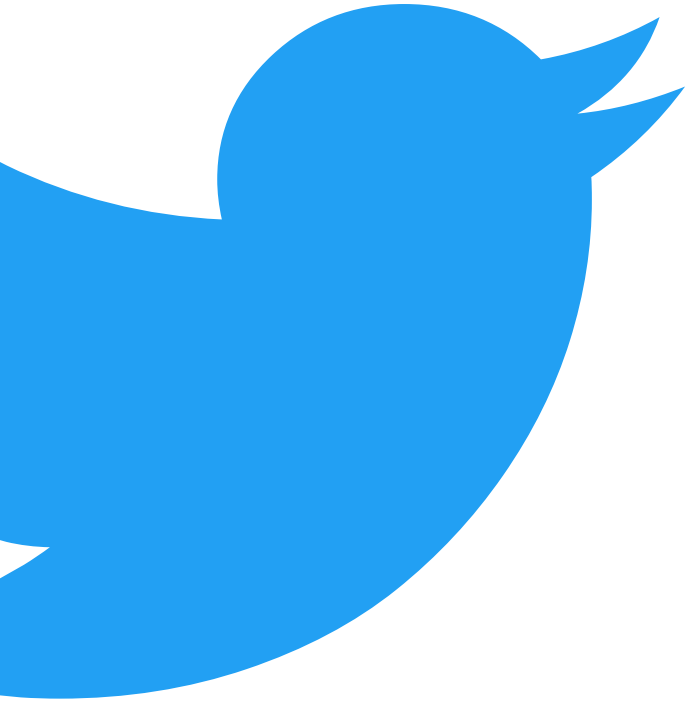


Database Management Systems

Twitter Search Application

Final Project

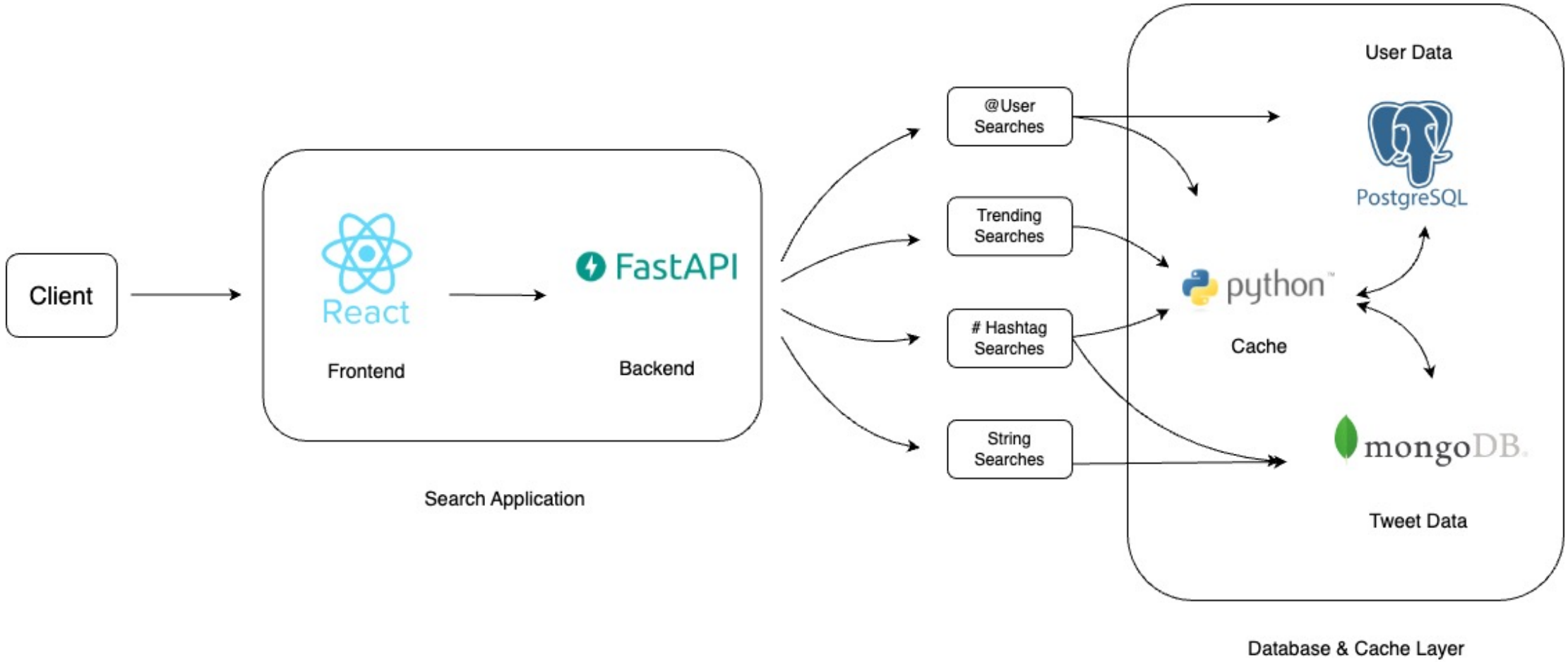
Team 31
- Sasank Chindirala

Data Curation

- The data used for our application was sourced from the corona-out-2 and corona-out-3 datasets provided.
- Each file was read line-by-line, parsed and processed before storing the needed user and tweet object models in the database.
- The individual line was itself a nested json object, that had important information inside the nested key's like "retweeted_status" and "quoted_status"


```
{
  "created_at":"Sun Apr 12 18:27:25 +0000 2020",
  "id":1249403767180668930,
  "id_str":"1249403767180668930",
  "text":"RT @nuffsaidny: wishing death on people is weirdo behavior.",
  "source":"\u003ca href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for iPhone\u003c\/a\u003e",
  "truncated":false,
  "in_reply_to_status_id":null,
  "in_reply_to_status_id_str":null,
  "in_reply_to_user_id":null,
  "in_reply_to_user_id_str":null,
  "in_reply_to_screen_name":null,
  "user":{ },
  "geo":null,
  "coordinates":null,
  "place":null,
  "contributors":null,
  "retweeted_status":{ },
  "quoted_status_id":1249315454797168641,
  "quoted_status_id_str":"1249315454797168641",
  "quoted_status":{ },
  "quoted_status_permalink":{ },
  "is_quote_status":true,
  "quote_count":0,
  "reply_count":0,
  "retweet_count":0,
  "favorite_count":0,
  "entities":{ },
  "favorited":false,
  "retweeted":false,
  "filter_level":"low",
  "lang":"en",
  "timestamp_ms":"1586716045552"
}
```

System Architecture



User Data

- An index on screen_name was created for efficient searches
- Another multi-column index was created for faster sorting of the search results.
- Username based searches were ranked on the basis of the number of followers and the number of tweets posted.

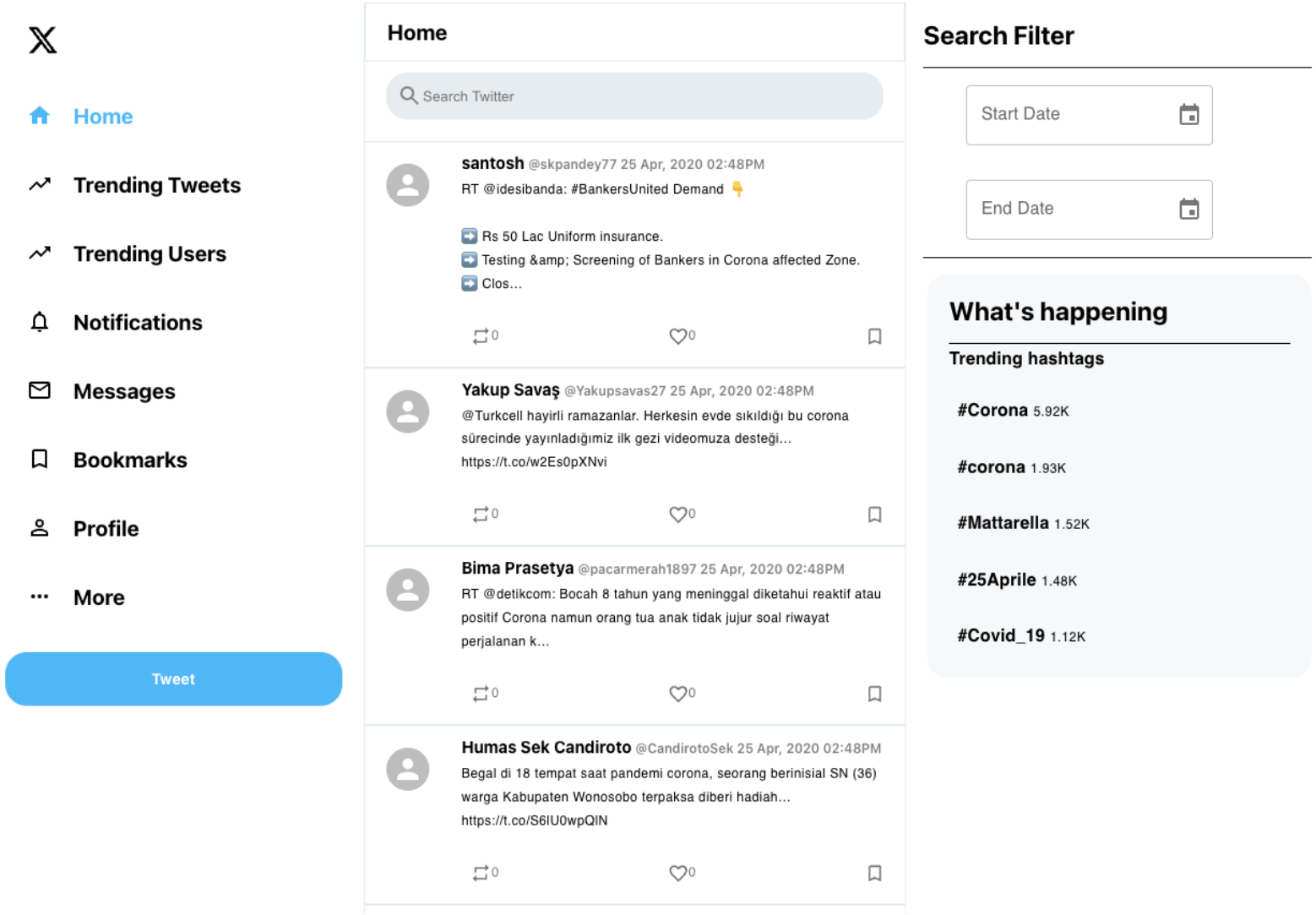
total_rows bigint	table_size text	column_count bigint
108043	35 MB	12

Column	Type	Collation	Nullable	Default	Storage
id	character varying(255)		not null		extended
name	character varying(255)				extended
screen_name	character varying(255)				extended
verified	boolean				plain
location	character varying(255)				extended
description	character varying(255)				extended
followers_count	bigint				plain
friends_count	bigint				plain
favourites_count	bigint				plain
statuses_count	bigint				plain
tweets_count	bigint				plain
created_at	timestamp without time zone				plain

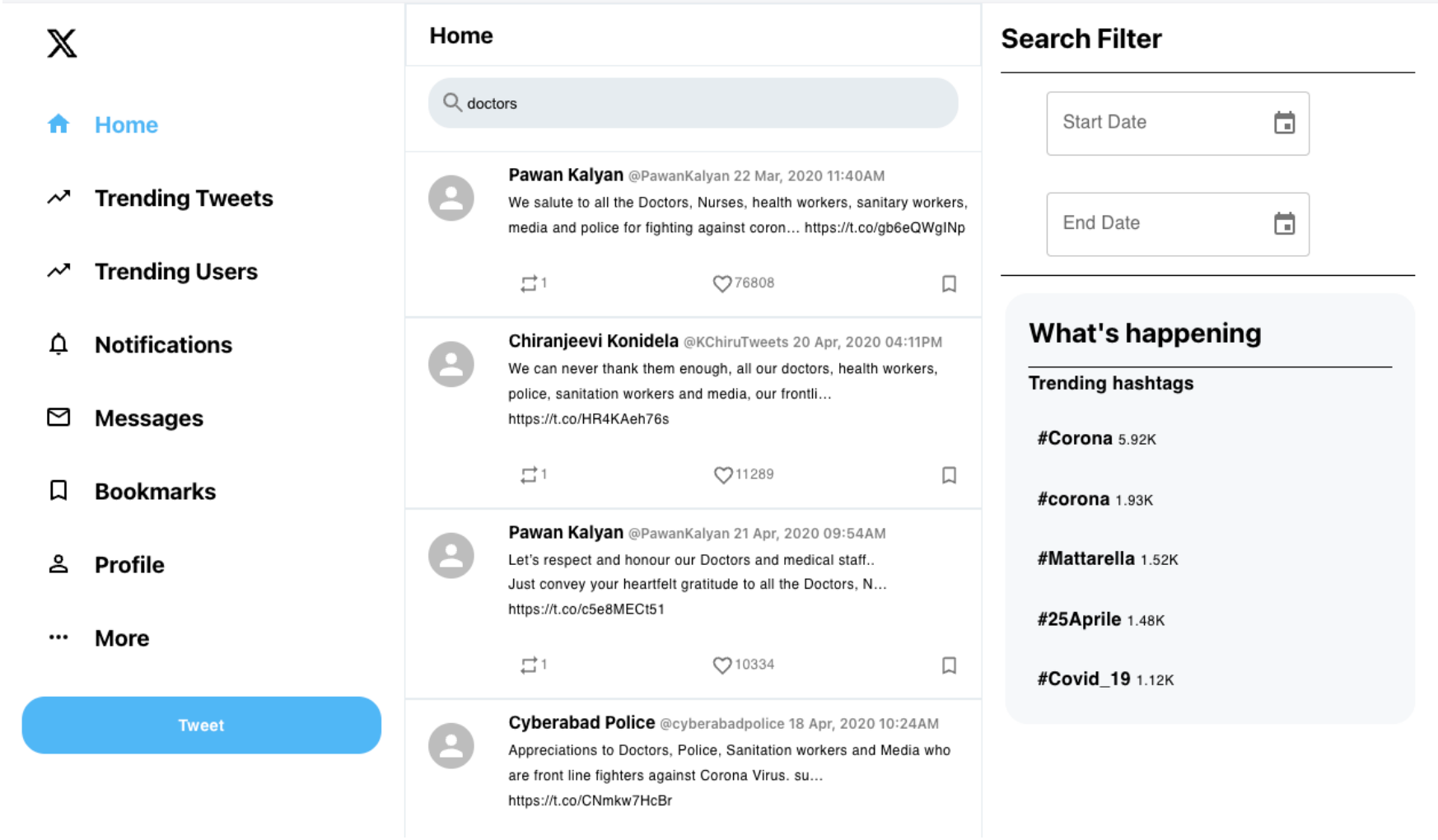
Tweet Data

- An index on the text field was created for efficient searches.
- A field of tweet_score was created for ranking the search results that assigns a weighted score to each tweet based on the number of likes and retweets that a particular tweet had.

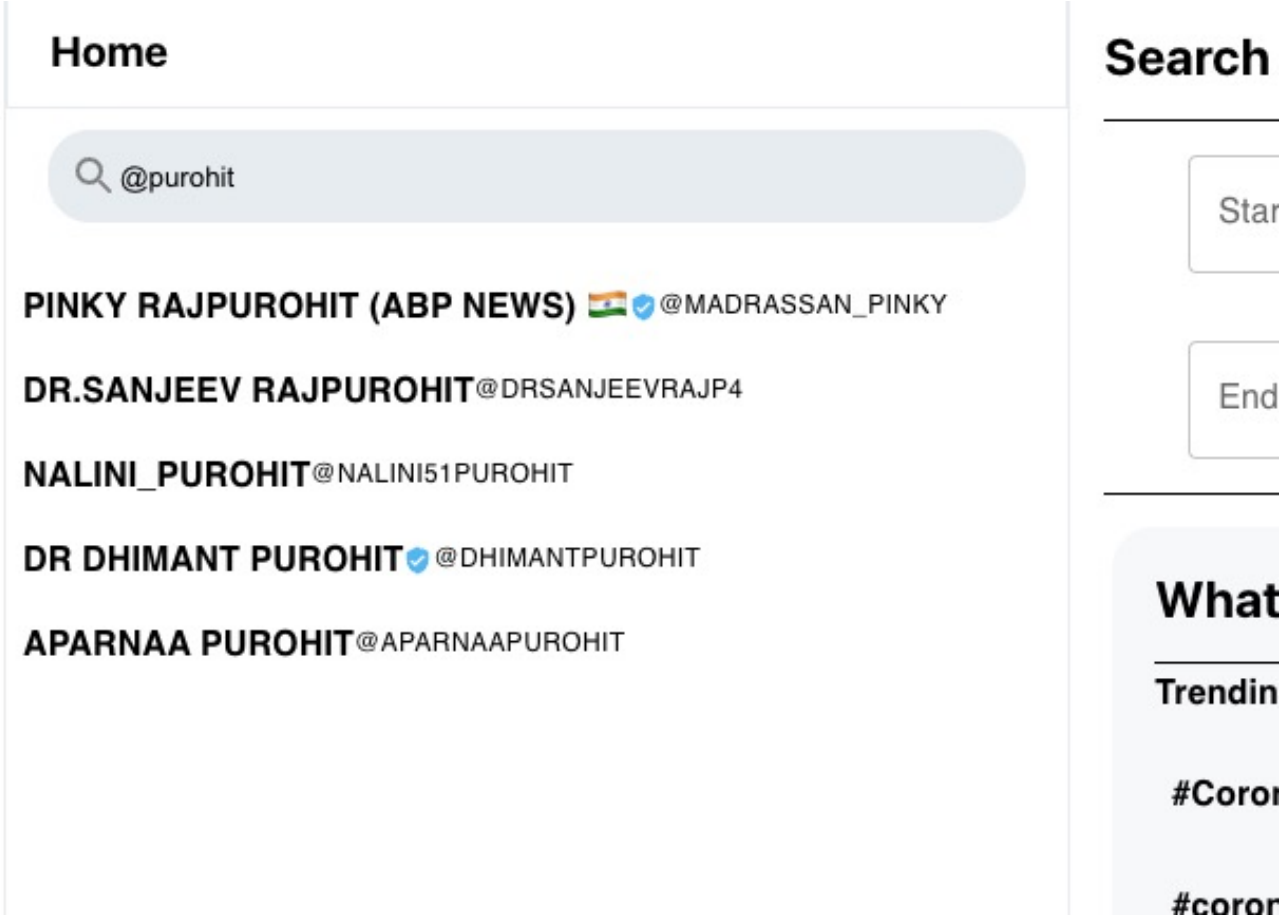
Application User Interface



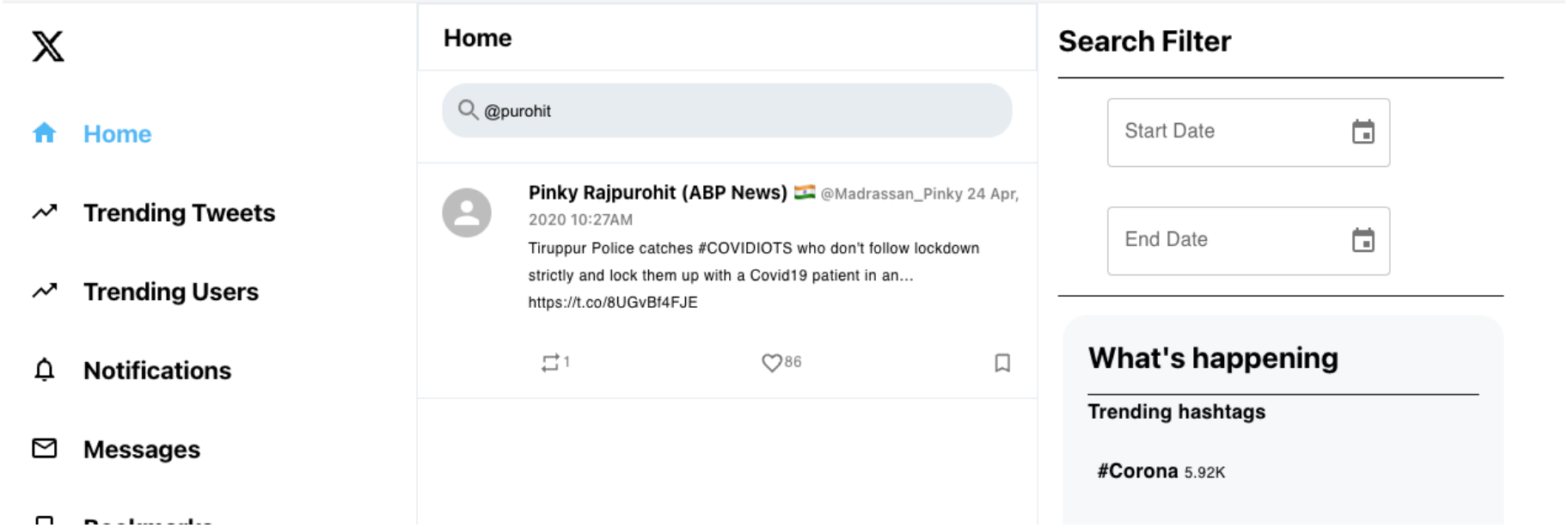
String Search



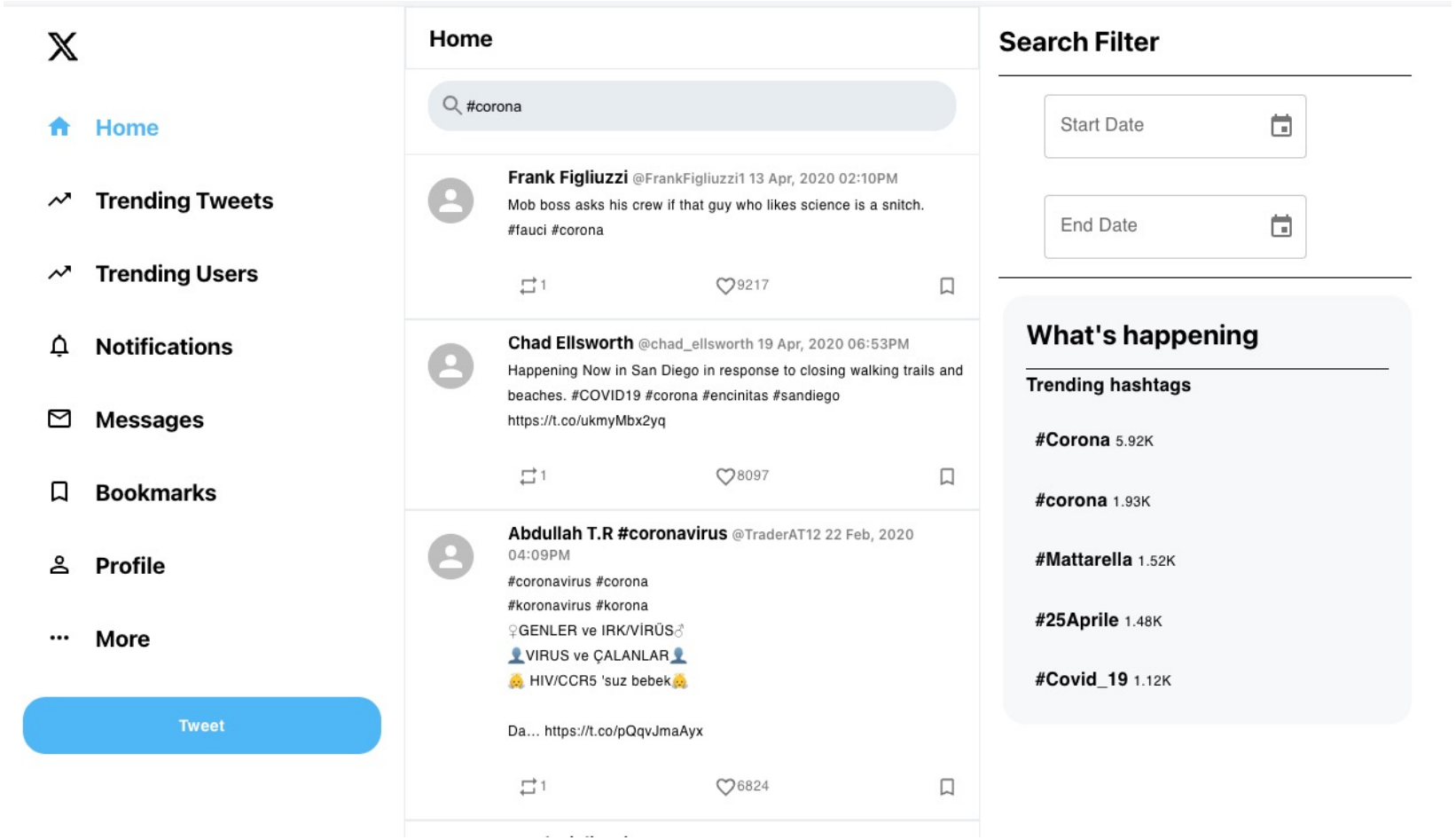
@Username Search



@Username Search – Tweet Drilldown

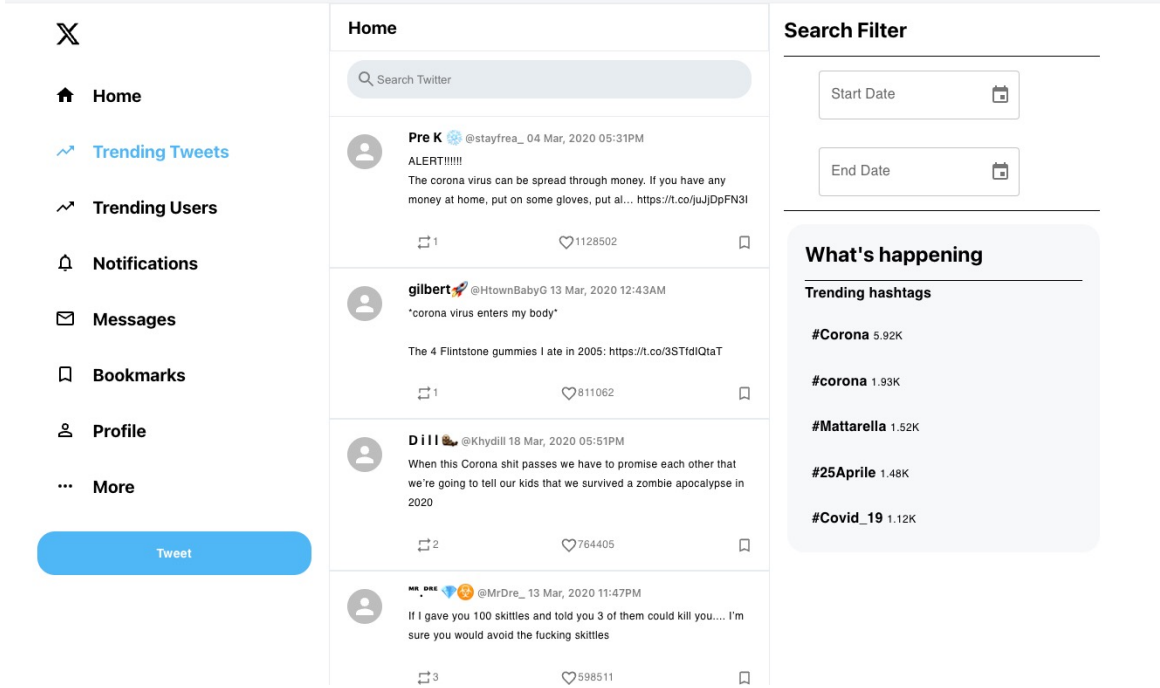


#Hashtag Search

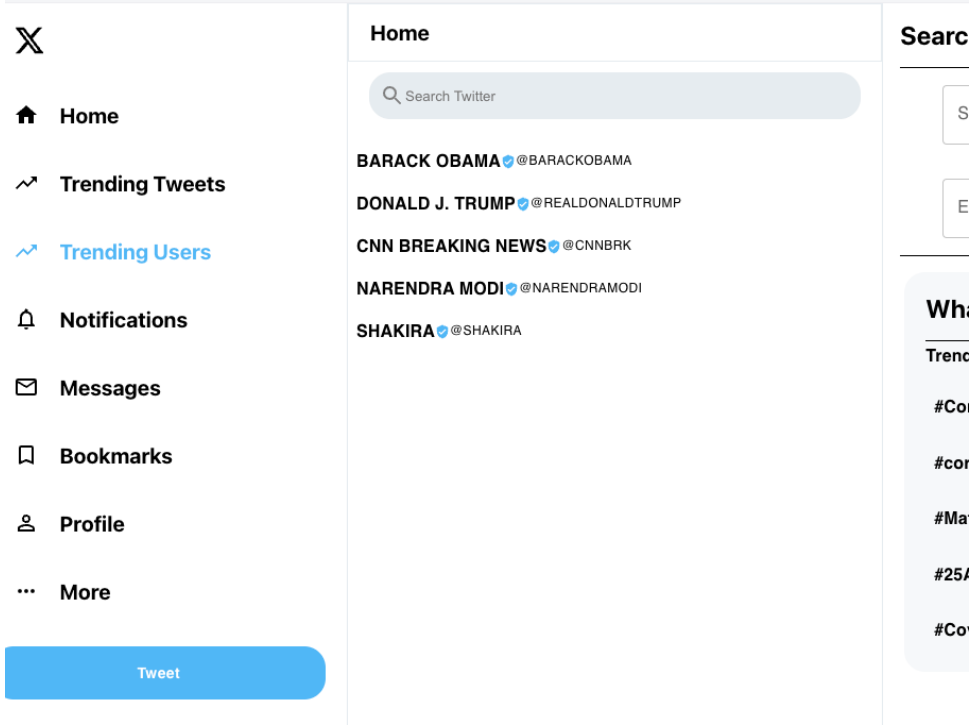


Trending Search

- Trending Tweets

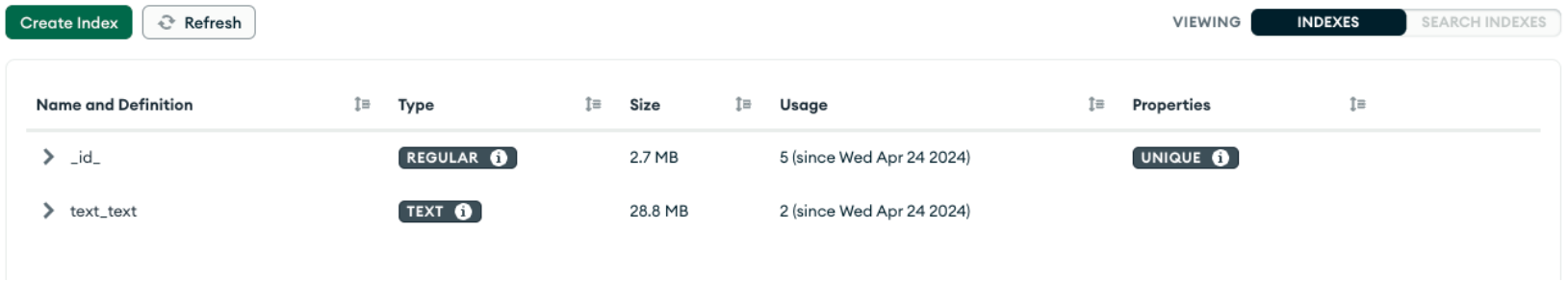


- Trending Users



Performance Optimizations

- Ranking or search relevance metrics were directly stored in the database schema to avoid on the fly computations
- Indexing
  - MongoDB indexing was done on the text field.



  - PostgresSQL can’t use a regular B-tree index on a pattern that starts with a wildcard (screen_name like %s%). So used pg_trgm extension to create a GIN (generalized inverted index) that can improve performance for **like** queries that use wildcard characters.

```
Indexes:
    "users_new_pkey" PRIMARY KEY, btree (id)
    "idx_users_ranking" btree (followers_count DESC, tweets_count DESC, verified DESC)
    "idx_users_screen_name" gin (screen_name gin_trgm_ops)
```

Performance Optimizations (Contd.)

- Cache
  - The cache class uses a **least- accessed eviction** strategy to ensure that frequently accessed keys are retained in the cache. This helps optimize the cache's memory and ensures that it can store a large amount of data within the defined limit of **10 MB**.
  - The cache class includes a **checkpoint** interval of 5 minutes, which allows the cache to regularly save its contents to disk. This ensures that in case of any failures, the cache can recover its contents and continue serving data to users without disruptions.
  - The cache class provides a **TTL** (time-to-live) feature that allows the cache to automatically remove stale data from the cache. With a TTL of one hour, any data that hasn't been accessed within a week will be automatically removed from the cache, ensuring that only the latest and relevant data is stored.

@Username Search

Without Cache

```
0.041877985000610 35 seconds
```

With Cache

```
2.7179718017578125e-05 seconds
```

#Hashtag Search

Without Cache

```
0.07761096954345703 seconds
```

With Cache

```
2.7894973754882812e-05 seconds
```

Thank You!