Scholar Gate

One stop solution for every researcher
scholargate.mukulhase.com

Team 8
Divanshu Jain (201401210)
Mukul Hase (201401144)
Hemant Kasat (201402215)
Saad Khan (20162043)
Akhil Gupta (20162026)

Github Link

https://github.com/mukulhase/Scholar-Gate/tree/master

Introduction

This system will be a one stop solution for all researchers and research enthusiasts looking for resources and possible reviews and comments on ongoing and finished researches from fellow researchers and field experts. It will provide a collaborative environment with a large open community.

Assumptions

1. Boundary of the system
    a. Does not handle online edits of uploaded documents, only versioning.
    b. Providing static public information and not facilitating contemporary social networking features such as live messaging etcetera.
2. Non-profit service.
3. There exists a party for continuous manual content moderation.
4. System is setup in a world which already has ongoing research and a large existing set of objects and works in such a way that it will support pre-existing objects which need not be internalized.
5. The user is expected to have a device running on modern (last 3 versions of Chrome, Firefox and Microsoft Edge) web-browser above with a minimum internet speed of 512 Kbps
6. Research metadata is obtained from Arxiv.

Stakeholders

1. Direct Stakeholders
    a. User: A researcher, research enthusiast or field expert.
    b. System Admin
2. Indirect Stakeholders
    a. Research Community
3. Domain Stakeholders
    a. Conference/journal committees

Functional Requirements

1. Authentication System
   a. Profile creation and validation.
2. Content Storage and Sharing
   a. Upload and Download of research works
   b. Versioning
3. Search Engine
   a. Filtering search over authors and papers
   b. Semantic searching over research work.
4. Crowd sourced review and moderation
   a. Comment and Reporting system.
5. Pre-moderation
   a. Auto-filtering of abusive and inappropriate language.

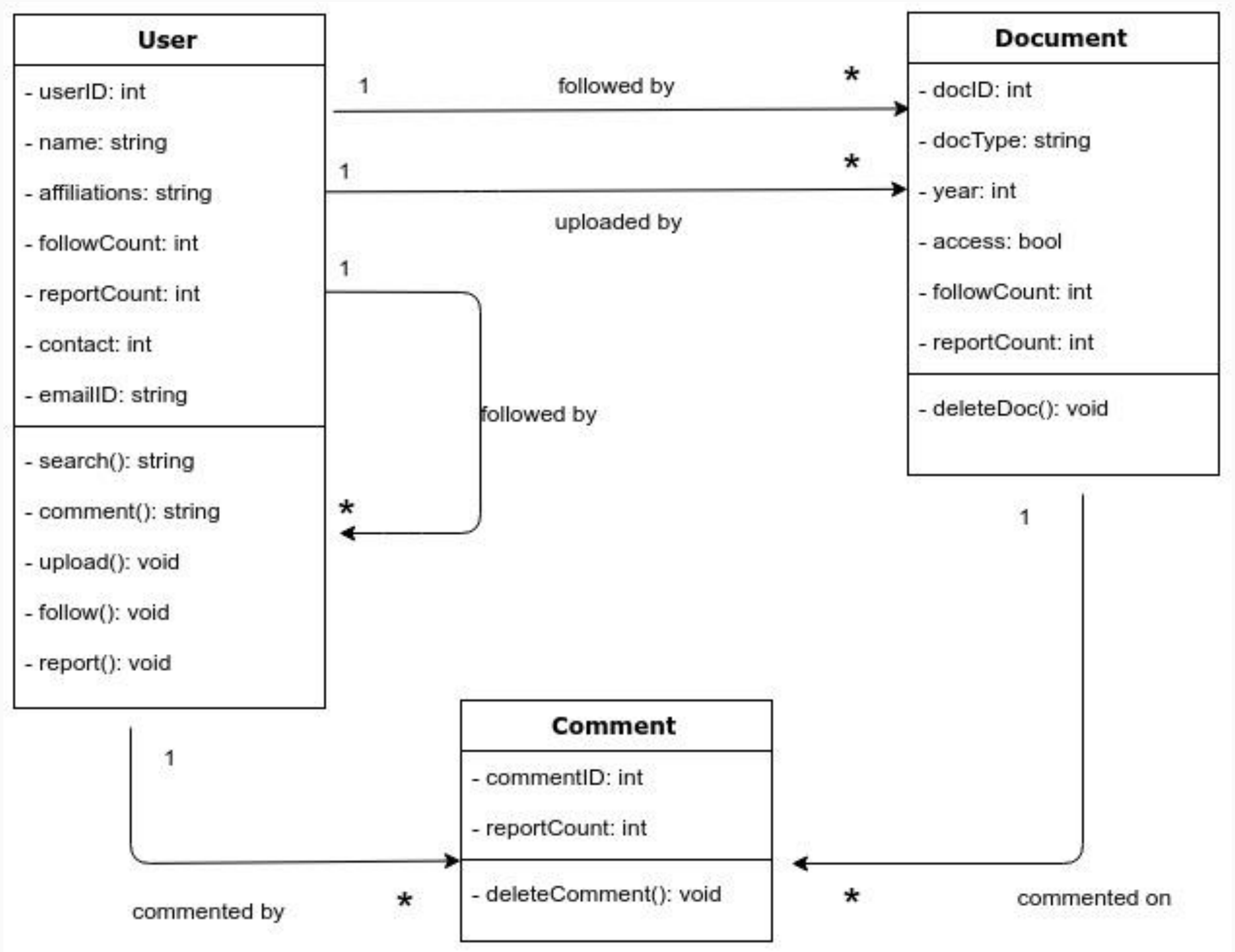
Non-Functional Requirements

1. **Performance**: Should support at least 300 concurrent users with websockets, upto 5 times more over only http, 800 document accesses per second, 100 publications added per second with negligible load on server.
2. **Security**: Minimal security requirements for authentication and permissions.
3. **Availability**: System has to be available all year, 99.95%. Globally.
4. **Capacity**: System is expected to be able to store upto a 1 million research works with yearly growth rate of 10%.
5. **Reliability**: Static serving - 1 day per year, Publish Requests/Search Requests - 1 day per month.
6. **Maintainability**: Code linting, Code smells to be automatically checked for.
7. **Documentation**: Per-Module documentation extended with Source-code generated documentation, to keep documentation up to date with current code.
8. **Usability**: Responsive design, Support for all browser versions from 2010 onwards.

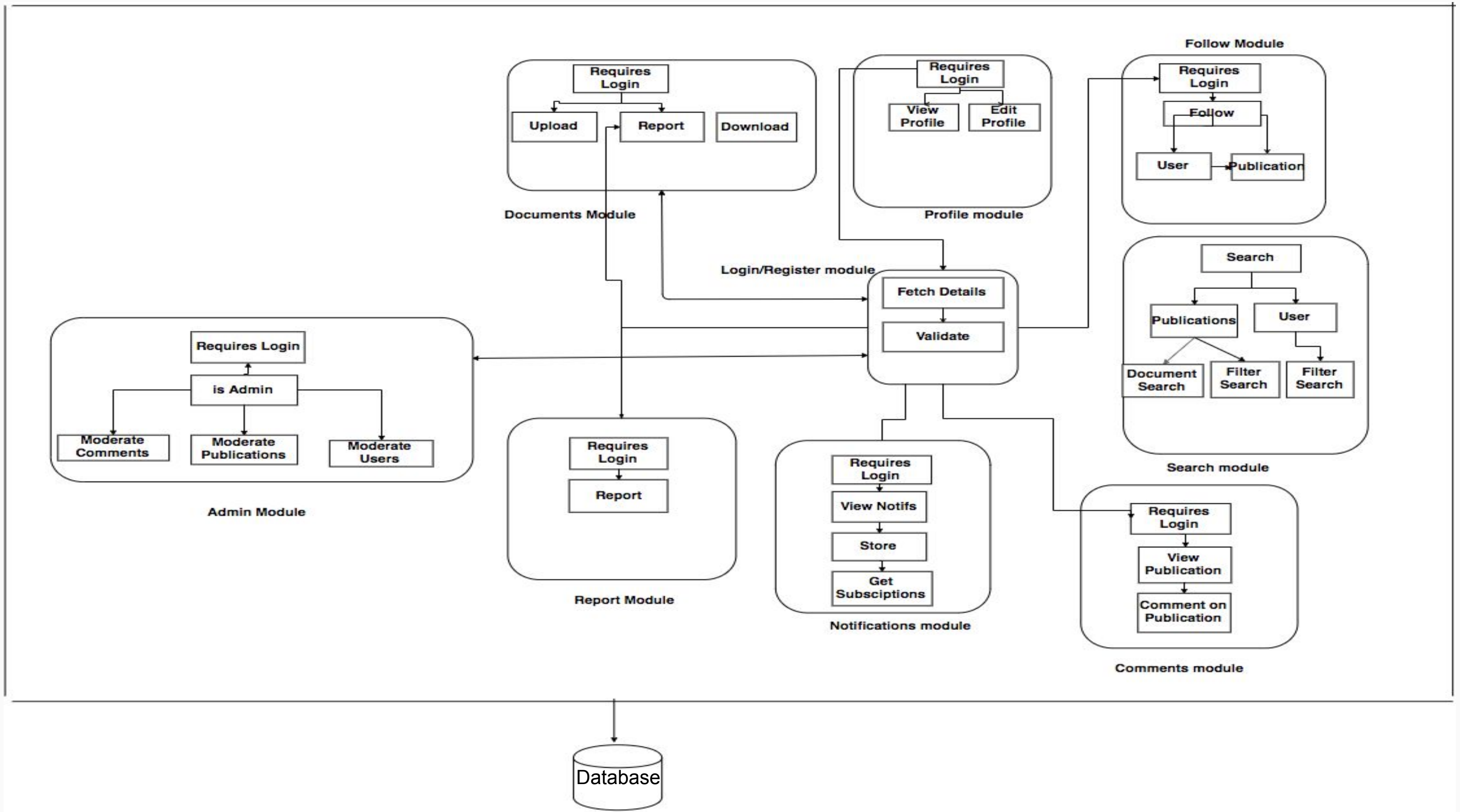
Features

1. Enable a user to perform an advanced search for research resources based on fields of study, tagged topics, conferences, ORC IDs and authors.
2. Intelligent input suggestions by scanning uploaded documents.
3. Users can comment and report the present works.
4. Provide a user a wall-like interface where they would get updates, announcements and notifications in their fields of interest.
5. Users can upload and manage research works.
6. Users can obtain reviews from the rest of the community. This would also be featured as part of notifications of the user.
7. Allow users to provide public contact information in order to facilitate networking within the community.
8. Up to date with latest data from services like Arxiv (updated bi-weekly).

Class Diagram



Detailed Architecture



Implementation

1. **Server-Client-side(SSR) logic**: Using Meteor.js for serving
2. **Frontend**: React frontend
3. **Storage**: Mongo Database Backend
4. **Graphcool**: Meteor for API and Subscriptions
5. **Document Storage**: Uploaded Research work stored on AWS S3
6. **Image and Styling**: Static files served from Cloudfare
7. **Semantic Search :** Whoosh searching library
    a. Whoosh is a fast, featureful full-text indexing and searching library implemented in pure Python
    b. Fielded indexing and search.

MongoDB Search

- MongoDB, one of the leading NoSQL databases, is well known for its fast performance, flexible schema, scalability and great indexing capabilities. At the core of this fast performance lies MongoDB indexes, which support efficient execution of queries by avoiding full-collection scans and hence limiting the number of documents MongoDB searches.
- MongoDB tokenizes and stems the indexed field’s text content, and sets up the indexes accordingly
- Text Search also includes Phrase Search, Negation Search also.
- Text search being integrated into the db kernel functionalities of MongoDB, it is totally consistent and works well even with sharding and replication.

Semantic Search Whoosh

We provide a advance search feature, Semantic Search, that helps researchers to query publications. Queries and publications are projected to a semantic vector space using tf-idf metric for terms in query and publication and it is given by:

tf-idf(t,p) = tf(t) * idf(t) where

tf(t) = (Number of times term t appears in a publication(p)) / (Total number of terms in the publication(p))

idf(t) = log_e(Total number of publications / Number of publications with term t in it).

These tf-idf vectors are used to get similarity between publication and query given by user. The ranked results(publications) based on similarity score are returned.

Whoosh

We use Whoosh for the Semantic Search. Whoosh is fast full-text indexing and searching library implemented in pure python.

Whoosh provide functionality to index static publications present already in the database and also incrementally add new publications uploaded by users to the index created. Indexing rather than just keyword search (regex) the publications enable the search to happen fast. It also provide searcher object on the index object and which is able to do many types of queries to the database of publications. It also allow user to search using field queries.

The implementation of functions required are here : https://github.com/hemantkasat/searchserver

References

- Software engineering course class slides
- Other similar sites like https://explore.researchgate.net and https://www.questia.com.
- Resource Reference : http://dblp.uni-trier.de
- Publication Data resource: https://github.com/karpathy/arxiv-sanity-preserver