Analysis Me





# Catalog

1. **Introduction**
2. **Motivation**
3. **Related work**
4. **System design**
5. **System implementation**
6. **Analysis and discussions**
7. **Conclusions and future work**
8. **Contribution of each team member**
9. **Introduction:**

In this project, we aim to design and develop a software system focusing on collecting, aggregating, classifying, analyzing, reporting, and visualizing data related to Service Oriented Software Engineering (SOSE) research community.

As the research community data has many relations among authors and papers along with many networks. Hence decided to use graph database which is ideal for this type of data. For in browser visualization is created using alchemy js which is built on d3 js for visualizing multi depth networks.

We have built a web platform where users are provided with login facility, allowed to follow authors and researchers in the research community. Users are allowed to access the mashup services which is created using workflow tools along with recommendation services based on users interest and activity.

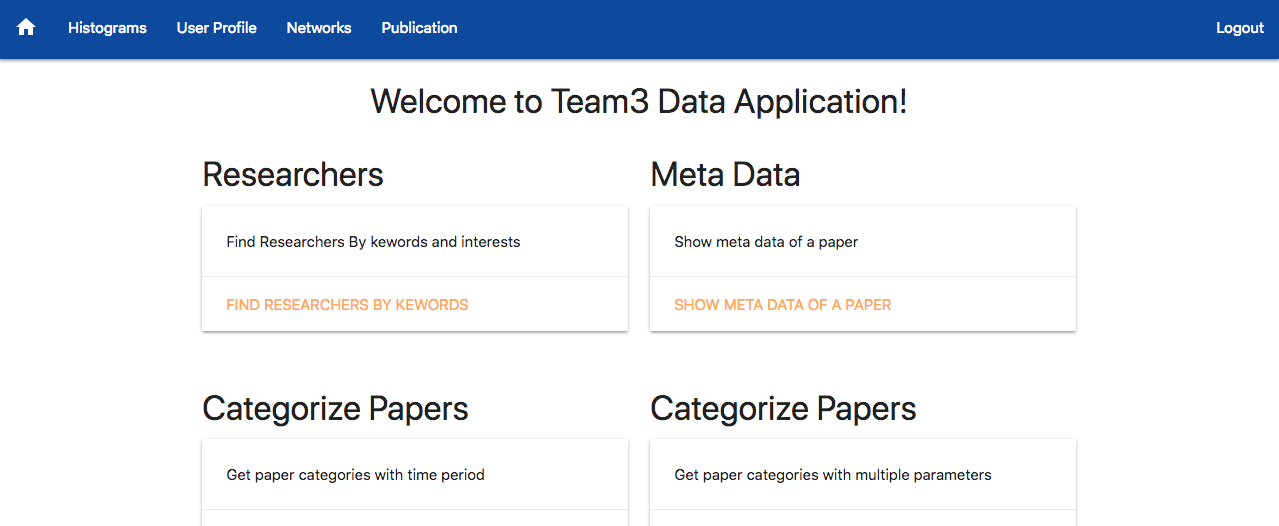


Fig. 1 Home Page.

**1.1 System Benefits:**

* Providing a platform where users can search in different ways to get insights of Research Community Data.
* Built a system where it can categorize papers based on the input parameters.
* Used in browser visualization library alchemy.js for visualizing networks of co-authors and relations with the papers.

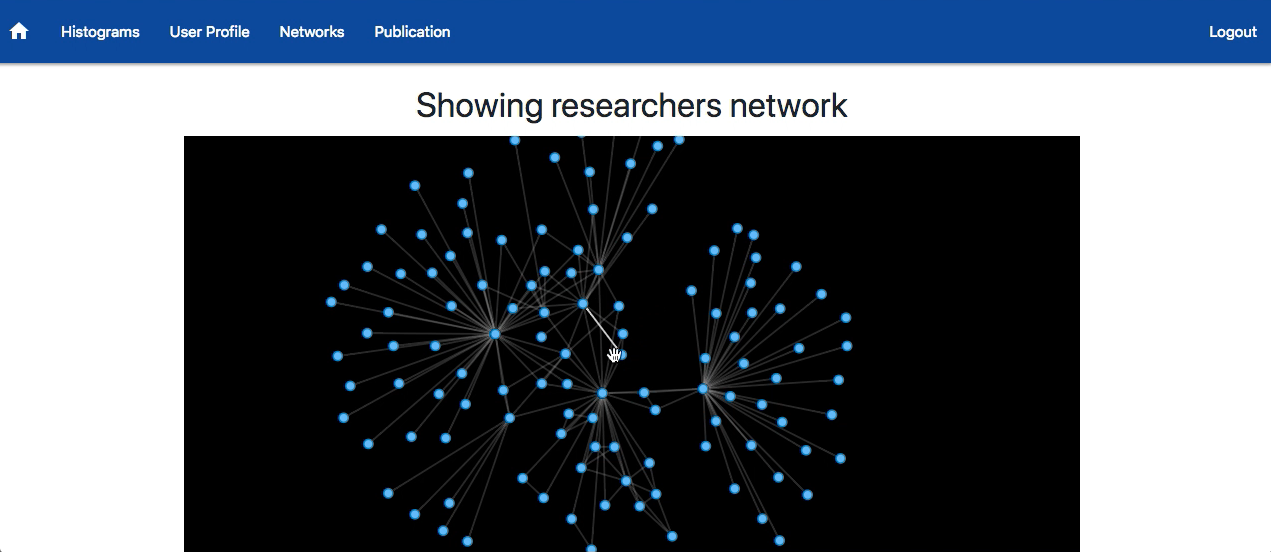


Fig. 1-1 Researchers Network.

1. **Motivation:**

Getting insights of the data is the major problem that most of the industries are currently facing. The amount of time and money that the industries are spending for this problem is tremendous. So we took this opportunity to tackle this problem by slicing and dicing the data in various ways and providing use cases to make good insights of the data. So the motivation of the the project is to provide a platform with standards where users can get good insights of the data.

**2.1 Usability:**

We designed the project in such a way that it is user friendly and very easy to understand the working of the whole system.

**2.2 Extensibility:**

We also ensured that in future to add new features in an easy way without any complexity.

**2.3 Scalability:**

We designed our project in such a way that it is easily scalable.

1. **Related work**

We found Google Scholar is related to our work for google scholar is a web based application for users to search with keywords to find papers or authors, which is know for the powerful searching engine precise searching results.

But for scholar searching engine like Google scholar, they could just focus their functions on keywords searching. For our job, we built an application with multiple selections for users who could do the searching as their requirements and get the intuitive results.

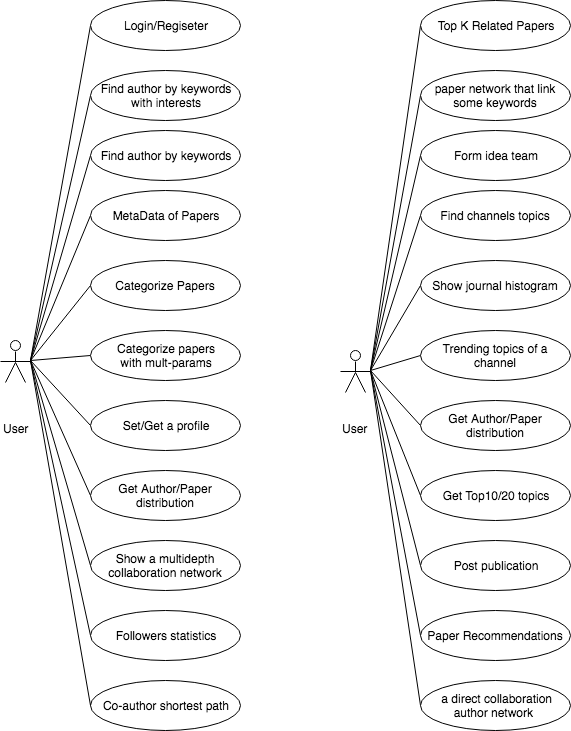
1. **System design**

In this section, we will introduce the system design from several aspects, including use case design, class design and deployment design.

**4.1 Use Case Design**

Based on the requirement given by professor, we defined the use cases below. We identify all users whether he is or not an author to be a user.

The use case diagram we designed is shown in picture 4-1.

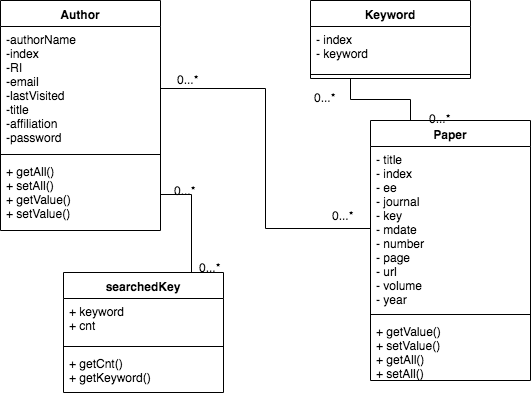


Fg. 4-1 Use Case Diagram

**4.2 Class Design**

Based on the use cases we identified above, we defined 4 classes entities to satisfy the requirement. The 4 classes including Author, Keyword, Searchedkey, Paper. Each class is corresponding with at least one use case.

The class diagram is shown in below picture 4-2:



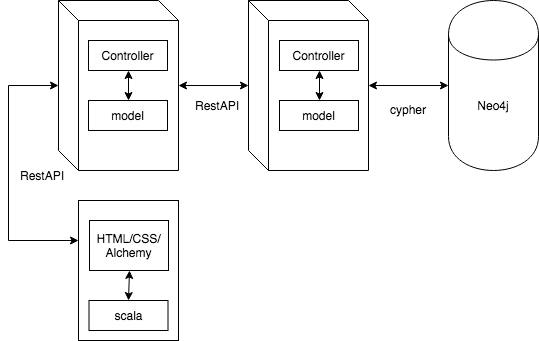
Fg. 4-2 Class Diagram

**4.3 Structure Design**

The structure of the whole system and the deployment view is shown in below picture 4-4:

In the client side, it contains frontend page logic controls, including page view and page logic. In the Server side, it contains backend business logic controls, including controller and repository, they are used to response the request from frontend and provide specific service.

FrontEnd/Client BackEnd/Server



Fg. 4-3 Deployment View

We use play framework to deal with both the frontend and the backend. Meanwhile, we use scala and Alchemy.js for viewing. Also, we use Neo4j and cypher to operate the database, it’s allowed us to binding the data from Neo4j and model.

1. **System implementation**

This section will present how we apply the frameworks and tools to build the system.

**5.1 Play Framework**

Play is a high-productivity Java and Scala web application framework that integrates the components and APIs you need for modern web application development. Play is based on a lightweight, stateless, web-friendly architecture and features predictable and minimal resource consumption for highly-scalable applications because of its reactive model.

Play Framework is used heavily in the system, which means the system is overall built on the Play Framework. Reasons why we choose the play framework is that it encapsulates the MVC allows us to focus on the business logic and flow.

**5.1.1 The Scalability of Play Framework**

Play Framework is unique in its architectural design when it comes to scalability. Unlike traditional thread-based web frameworks that are often blocked on I/O where each thread in the server receives a request, processes it, calls I/O or database, and wait for the reply from I/O or database before returning to send the response back to the frontend, Play framework employs what’s called “event-based” architecture. In an event-based server, there are dedicated threads that receive requests and process them from the frontends. Upon sending an I/O or database call, the thread immediately returns to receive and process new requests without waiting for the reply. Waiting for the I/O response is delegated to another group of threads whose main roles is to receive response from I/O calls and send them back to frontend. In this sense, the “event-based” play framework is non-blocking on I/O and database calls. As a result, complex APIs that require intensive I/O or query computation don’t block other incoming requests from the frontend, making Play scalable.

**5.2 neo4j**

In this project, we choose neo4j as our database, which is a graph database for connected data. In that the research data we are working on include a great amount of relationships. The traditional relational database are not suitable for this scenario that usually it costs too much to apply JOIN statement to fetch results, while neo4j can drastically improve performance on connected data. We utilize cypher, a neo4j graph query language, to describe patterns in graph.

**5.3 alchemy.js**

Alchemy.js is an easily customized graph drawing application for web development. It depends on jQuery, D3 and lodash. We choose it that it is a good way to display basic stuff compared to other graph display libraries and it is easy for developers to get to know how to use it.

**5.4 Taverna**

We picked up Taverna as the workflow management tool to formalize the process in order to better utilize the built services and big data. With the support of Taverna, we are able to track the data history and formalize the procedure and understand and repeat it.

The Taverna suite is is an open source domain independent [Workflow Management System](https://taverna.incubator.apache.org/introduction/what-is-a-workflow-management-system), which is written in Java. Taverna automates experimental methods through the use of a number of different services from a diverse set of domains.

Taverna obtains the characteristics of user-friendly, open development, engagement, independence, shared ownership, sustainability, which is the best fit of our project.

**5.5 Tableau**

Tableau is business intelligence software that helps people see and understand their data.

1. **Analysis and discussions**

During the development of the project, we notice following discussions and analysis:

**6.1 More Microservices for scalability**

There are certain APIs that require complex queries, therefore more compute power. If those APIs are invoked with high frequency, it can lead to scale issues. The general approach is to decouple the complex queries from the rest of the code and create an independent microservice with a separate JVM. As a result, the main service can be relieved of the strong load.

**6.2 Read Performance**

Most of the APIs are read operations. This gives us a huge incentive to implement cache to improve read performance. For each category of API, we can use some variations of a LinkedHashMap to serve as the cache for that category of API. And our cache will have the replacement policy of Least Recently Used since LinkedHashMap has built in functionality to remove the least recently used entry.

1. **Conclusions and future work**

**7.1 Periodic Statistics Improvement**

Some of our APIs have to do with computing periodic statistics. Every time these APIs are invoked, we need to aggregate a range of numbers. To improvement performance, we don’t want to recompute the range aggregation everytime the API is invoked. The idea is to have dedicated background threads that run periodically and aggregate the results in advanced and save them in a stable storage or memory.

**7.2 High Availability and Disaster Recoverability**

Any modern web application will need to have high availability and disaster recoverability. The idea of achieving high availability is to remove single point of failure by introducing redundancies. We would want to have multiple servers load balancing each other so failure from one server will not impact the availability of the service. A system that’s disaster recoverable is a system with replica of stable storage in case of data center shutdown. So, our web application will need to have a replica of database constantly in sync with the main active database.

**8. Contributions of each team member**

|  |  |  |
| --- | --- | --- |
| Team Member | Basic Requirements | Advance Requirements |
| Sri Harsha Sanne | 5,6,15,16 | 2,7,8,9,10 |
| Tingfang Pan | 4,7,10,12 | 4,5,11 |
| Bowen Zhang | 3(initial architectural setup) , 13, refined 15, 16 | 3, 6 |
| Hao Tang | 1,2,3,9,11 | 1,14,15, Workflow Design |
| Xinyuan Chen | 8,14 | 12,13 |