SOFTWARE PROCESS ARTIFACTS

For

ShowMe: Related Work

Prepared By: -

Group 6

Software Factory I

Department of Software Engineering

Arizona State University

Academic Year: 2017-2018

Version: 1.8

Preface

This document is designed for evaluating the efforts put together by the team for this project. It will also help people who will be testing or using this application for their own benefits. It describes about the user requirements and functional requirements that was used to design and build the application. Each chapter tells about the functionality of this application and understand the purpose of it. In the second half of the document we have described about the design and architecture along with the technologies used. We have also included the tasks that were involved in building the application from scratch in details. If you are a researcher or a student, then it should help you in your research work, express your views on different papers and benefit from other user's ratings and comments.

Revision History

Date	Revision	Description	Author
11/20/2017	1.0	Initial Draft	Abhishek Dutta
11/25/2017	1.1	Requirements	Abhishek Dutta
11/26/2017	1.2	UI Specification	Pranjal Karankar
11/26/2017	1.3	Design information and Sprint tasks	Sachin Magar
11/27/2017	1.4	Web Server, API and Database Design	Pushkar Ladhe
11/27/2017	1.5	External Libraries	Pranjal Karankar
11/28/2017	1.6	Installation steps	Pranjal/Sachin
11/28/2017	1.7	Testing Data	Rajat Sinha
11/30/2017	1.8	Final Draft	Abhishek Dutta

Table of Contents

1. Introduction

1.1 Purpose

The main objective of this document is to illustrate the functionality, requirements, design, implementation methods and the effort involved in the ShowMe:ShowMe Related Work application. This application is mainly focused on providing an interface to the researchers for navigating through various research papers which may be related or inter related by defining the relations among them. The application is intended mainly for researchers in various fields who can go through various papers needed for their work. Students can also benefit from this application.

1.2 Scope

The application allows the user to search for a paper by the paper name or its author which is shown in the form of a graphical notation: nodes and edges. The nodes represent the research papers and the edges represent the relation between two papers. Researchers can click on a node to view the research paper. Also, they can navigate to other research papers that are related to the current one by navigating through the edges and clicking on other nodes.

It allows them to upvote/downvote a particular edge to show the relevance of the other paper in relation to the first paper. They can also comment on a particular link in case they want to make some notes about the paper or edge. Users need to login using google to reveal their identity before they update anything.

1.3 Definitions, Acronyms, and Abbreviations

AJAX	Asynchronous JavaScript and XML
MVVM	Model–view–view model
SPA	Single Page Application
XML	Xtensible Markup Language
DOM	Document Object Model
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language

SRS	Software Requirements Specification
UI	User Interface
OAuth	Open Authentication
MVC	Model View Controller
UI	User Interface

Table 1. Acronyms and Terminology

1.4 References

Books

- Software Engineering: Ninth Edition by Ian Sommerville
- The Personal Software Process (PSP SM), November 2000 by Watts S. Humphrey

2. Overall Description

2.1 Product Perspective

This is a web based application implemented in three tier architectures using MVC architecture. The application provides the researchers a platform to read through various research papers with an interactive functionality to comment and vote on the relations of the papers. User needs to authenticate themselves before they can vote or comment on any part. There is another product called “Google Scholar” that is currently available in the market which provides the web links for various papers, but it lacks any kind of interactive functionality with the user.

2.2 Product Architecture

The web application is built on a three-layered architecture. In the top tier we have our client-side application which will interact with user. In the middle tier we have our middleware where all the business logic and rules are defined. Bottom tier consists of the database where all the data and relations among data are stored. Below is the system architecture diagram.

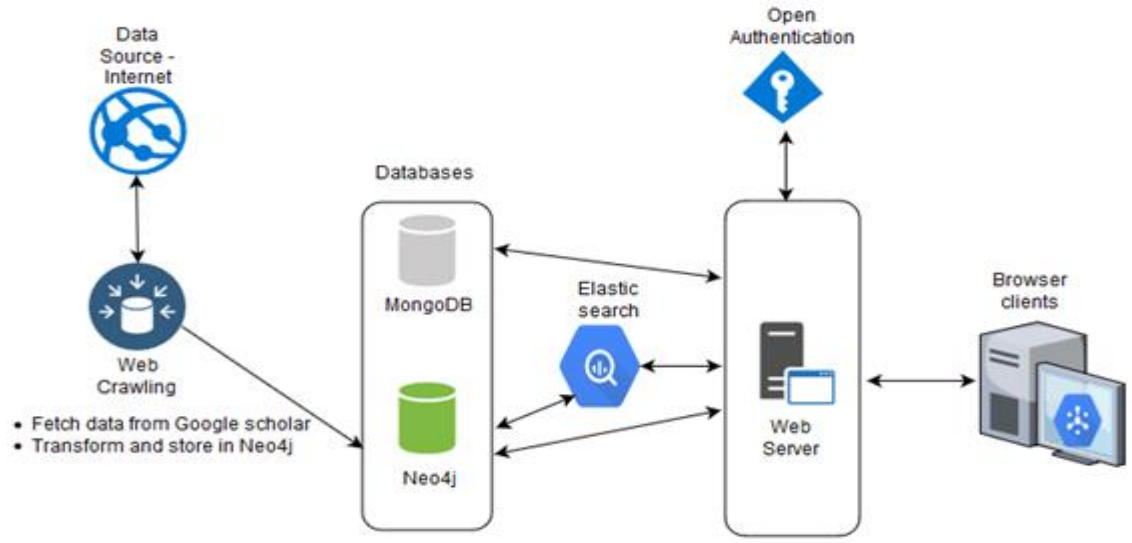


Figure 1. System Architecture

Each of the components are described below individually: -

Internet Data Source: These signify the websites which provide relevant information for the research papers, such as, the paper's DOI, author name, URL to the paper, abstract, references, etc. These websites will act as the source of information for our project. This data can be collected either manually or by using the web crawlers. Since we will need a huge volume of data, we are relying on web crawlers. For the prototype, we have manually collected data for one hundred research papers form Artificial Intelligence and Computer Graphics. Once we have a running prototype, we will be scaling the data using web crawlers.

Web Crawling: Web crawlers are the programs which collect relevant information from websites. We will develop web crawlers using python libraries - beautiful soup and scrapy, since we have some experience with using these libraries. Crawlers will fetch all the important features of the papers like - paper's DOI, author name, URL to the paper, abstract, references, etc. These crawlers will be hosted on robust systems where we can keep them running for long hours.

MongoDB: MongoDB is a document store NoSQL database. Since, our data will not have a defined schema and it will be growing at an undefined rate, we chose to use a Nosql database over a relational database. We are using MongoDB to store user authentication information along with the user comments and upvotes/downvotes for the relationship between two papers.

Neo4J: Neo4J is a NoSQL graph database. The essential components in our project which consists of the papers and the relationship between them, exhibits a graph relation and hence we decided to use this graph DB. We are using Neo4J to store information about papers along with the relationship between them. Papers are getting stored as nodes and the relation as the edges of a graph. Initially, we were planning to use only one database (Neo4J) for storing all the information but, on further research we

found out that this approach would hamper the performance of the application. Storing the user comments, upvotes/downvotes and the user authentication information would make the Neo4J dB heavily loaded with data and slow down the performance. So, we decided to use Mongo DB for storing the user comments, upvotes/downvotes and user authentication information.

Elasticsearch: Elasticsearch is an open source search engine. We are using it to facilitate the searching of papers.

Open Authentication: Open Authentication (OAuth) allows users to use their existing account like google, GitHub, yahoo etc. for authentication rather than having to create a new account. As of now, we are allowing users to create an account on our app and we are using the same account for authentication purposes. Once we have a running prototype, we will further enhance it to accommodate Open Authentication.

Web Server: A node js web server is used to serve the application. This web server also serves as an application server and contains business logic of the product. This node based server communicates with backend databases namely Neo4J and MongoDB to perform data manipulation operations. Moreover, the logic to fetch the data, creation of responses is distributed on RESTful routes with the use of Express.js.

Client-Side Framework: We are using leading client side framework - Vue.js for creation of single page web application. Vue.js facilitates to implement MVVM design pattern and provides rich user experience and faster loading time using its virtual DOM.

2.3 Product Functionality

The application provides a list of user interactive features for better experience for researchers thereby saving them a lot of time and manual effort. Below are the set of functionalities that the application offers to its end users: -

- Users can read through different papers by clicking on the nodes.
- Users can navigate back and forth to other papers through the links to different nodes.
- Users can upvote/downvote the links between different papers depending on the relevance and quality.
- Users can comment on the links to justify their vote or just to add a note for other users to benefit from it.
- Users can search for a paper.
- Users can search papers categorized based on field of study.

3. Specific Requirements

3.1 User Requirements

From a user's perspective, the application should be able to visualize relations among papers as graphs with each node representing a paper and a directed edge between two nodes if one paper cites another. Users should see the paper if the user clicks on the node and relation properties if they click on the

edge. Furthermore, they should be able to make an edge weak or strong depending on the relevance and context.

3.2 Functional Requirements

The main functional requirements gathered from the sponsor were the application should be able to visualize the papers and their relations in graphical format in the UI side. It should allow the users to navigate through the papers and enhance it by making it more interactive. Users should have the option to vote and comment on any of the edges so that they can other users can benefit it at a later stage. The application should also have search functionalities to search for a specific paper.

3.3 Non-Functional Requirements

The non-functional requirements specified by the sponsor was mainly more focused towards the look and feel of the UI using different visualization libraries. Other requirements included that valid user profiles needs to be associated with whoever wants to vote or comment. Others can view the papers without authenticating themselves. Application should not maintain any user passwords in the Database.

3.4 Software Requirements

The application is built using following tools / Software. It is required to have this software configured on the system.

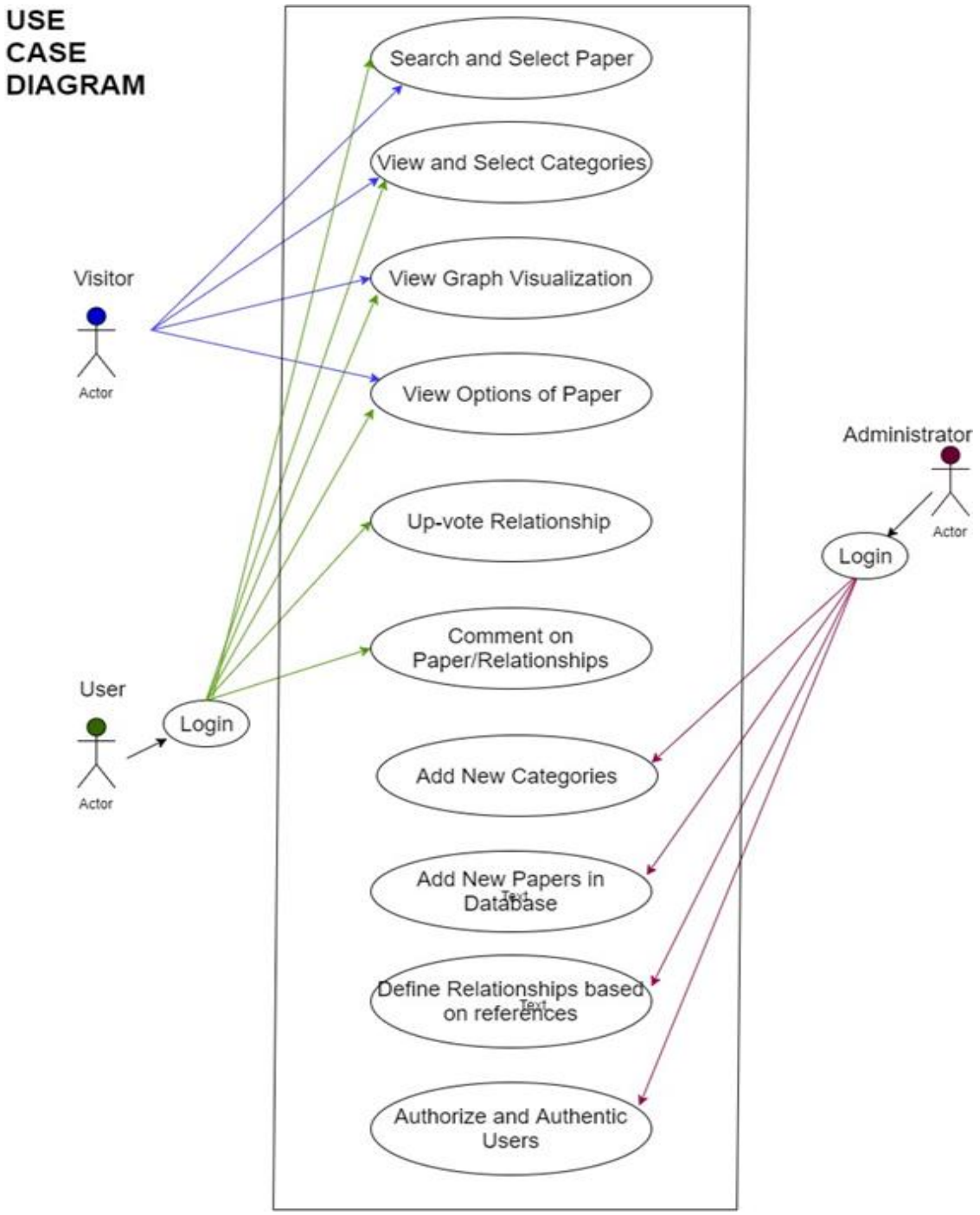
- MongoDB (3.4.10)
- Neo4J (3.4.0)
- Node.js (8.9.1)
- Vue.js (2.0)
- OAuth (2.0)
- Cytoscape

3.5 Hardware Requirements

A standard computer system should be able run this application easily. Any system with below minimum configurations are sufficient to host it.

- 4 GB RAM
- X86 or x64 bit Windows/Linux/iOS Operating system
- 50 GB Hard drive (Depends on the usage to store the data)
- 2.1 GHZ CPU

3.6 UML Diagram



Use-Case Description:

The use-case has three actors that perform specific operations and interact with the application.

User - This actor must first login to validate that they are a valid user. After logging-in, the user has the option to search for any paper. On selecting the searched paper, the user also gets to see all the papers that are related to this paper. Each of the related paper will have comments and upvotes/downvotes, that will define the strength of the relationship between these papers. The user has the option to write comments for the relation between any two papers and provide a upvote or downvote to signify the relationship between two papers. Instead of searching for a paper, the user can also simply select a category and view different papers in the category along with the relationship between them. On selection of any paper, the UI is refreshed to display the papers related to the selected paper.

Visitor - This actor does not have to login. They can search for any paper or view papers in any category, but the visitor cannot comment or upvote/downvote the relationship between two papers. The visitor can simply view the papers but cannot contribute in any manner.

Administrator - This actor's responsibility is to maintain the whole application. The administrator can add new categories for research papers along with adding papers in the categories. They can also add the initial relationship between the papers. The admin can also remove papers or the relationship between two papers. The administrator can also restrict or unauthorize users if any malicious activity is detected, to maintain the integrity of the application.

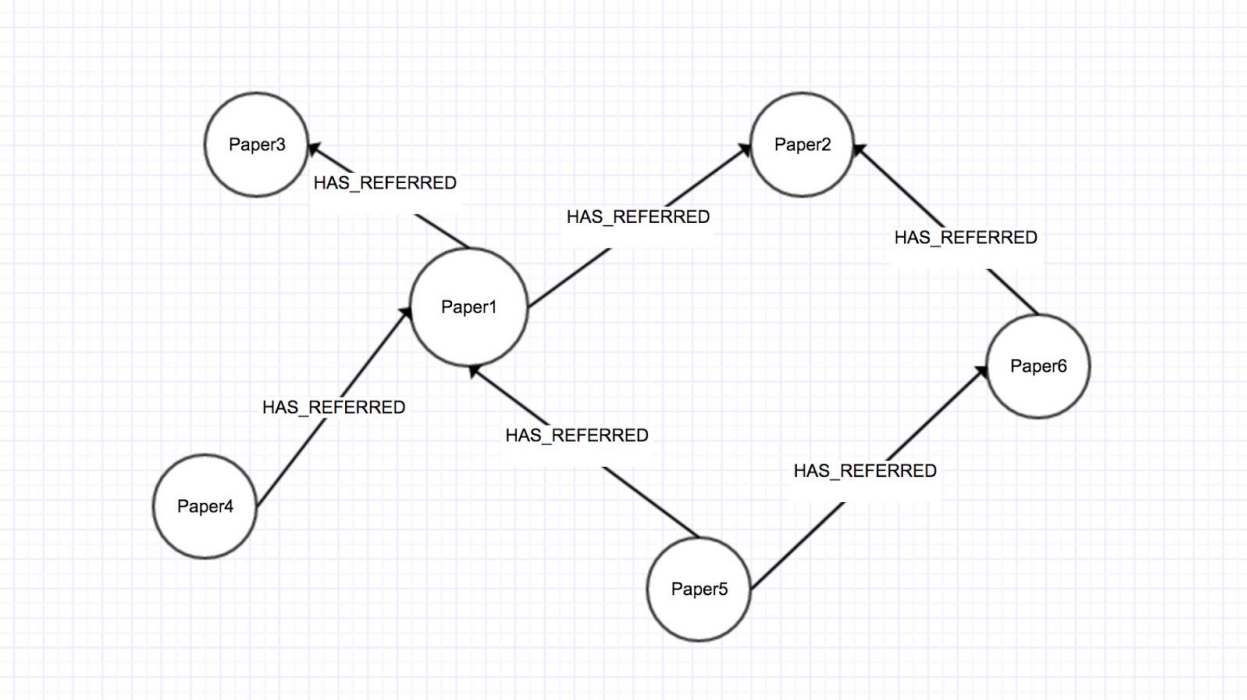
4. DATABASE DESIGN

4.1 Data Gathering

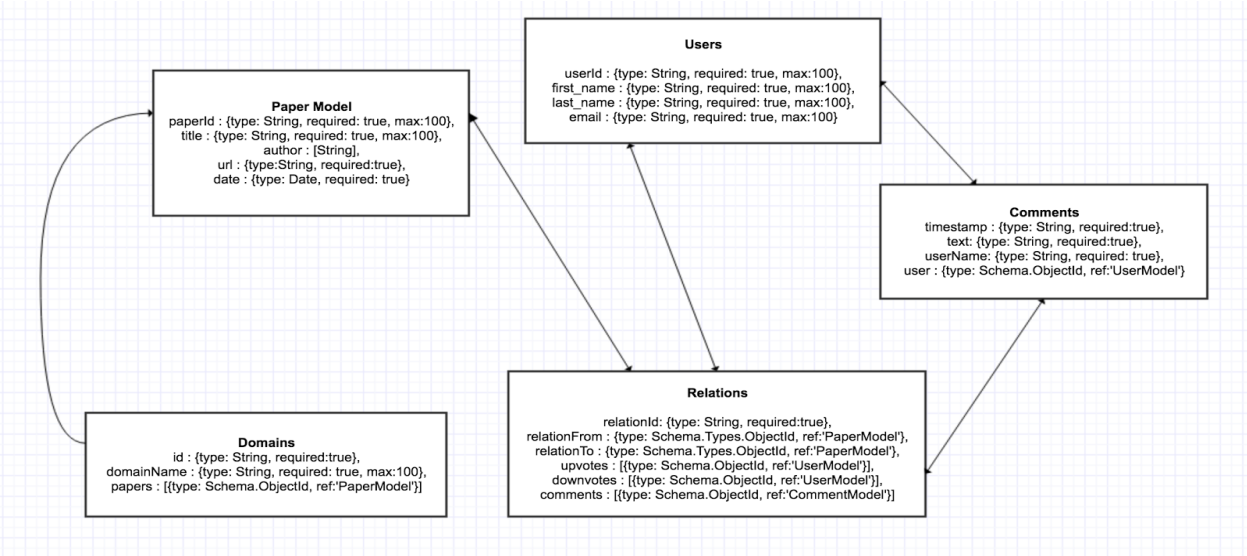
This project revolves around research paper and the relationship between these papers. Hence, collection of paper data has been the very base requirement for this project. For the purpose of the prototype, we have collected data manually for one hundred papers, divided between two categories - Artificial Intelligence/Machine Learning and Computer Graphics. This data has been extracted from multiple websites and contains information such as - title of the paper, author name, domain of the paper, referenced from (signifies the paper it was referenced from), and the link of the paper. For a future score, we plan to run a crawler that will pick all this data up and directly insert it into the database.

4.2 Neo4J Graph Database Design

The Neo4j is being used to store the graph that connects multiple research papers based on their citations. The Neo4j allows us to easily manage and retrieve graph from the database and pass it on to UI for visualization. Shown in the figure is a sample graph in Neo4j. The nodes are the research papers wherein the edges are relations between them. The graph for Neo4j has been built by gathering data from multiple sites where research papers are available.



4.3 Mongo DB NoSQL Database Design



The MongoDB is the main interaction point for our web application. The design has been made this way because retrieval and storage in MongoDB is faster than Neo4J. There are in all 5 models in MongoDB - Users Model, Paper Model, Domains Model, Relations Model and Comments Model.

The User Model stores the data for a user of the system. This allows us to manage user accounts and control activities in the application. We store the user Id, first name, last name and email of

the user. The Paper and Domains Models store data related to research papers in the system. It has the name, authors, date, URL, domain and unique Ids for identification.

The Relations Model stores relationship between two papers. It stores the relation id, relation from (paper id), relation to (paper id), upvotes, downvotes and a pointer to comments module and user module. The Comments Model stores the comments related to a specific relation. It stores the comment text and a pointer to the user profile of the user who commented.

5. Web Server

5.1 Overview

The web server is an express based server which uses http for communicating with the UI. It provides APIs that can be used to communicate with the two databases. The Web Server provides 10 APIs that allow smooth communications between database and front end.

5.2 Application Programming Interfaces (API)

The domains API allows UI to fetch all the domains available in database and show them to end user. The User API finds a user in the database and returns true if found else returns false. The Papers by Domain API fetches all the research papers available in the system for a domain. The Relations API finds a relation based on its id and returns the upvotes, downvotes, comments and the details of comments for that relation.

The add upvotes and add downvotes API adds upvotes and downvotes respectively to a relation. Similarly, the remove upvotes and remove downvotes API allows a user to cancel their upvote or downvote on a relation. We use the difference between upvotes and downvotes to find the weight of a relation.

The add comments API stores the comments posted by users to the comments module in accordance with the database design. The API also maps the comments to the user who has posted the comment, allowing us to fetch it back in relations API when requested.

The graph API is the API that enables Graph Visualization. This API fetches data from Neo4j for building the graph in UI. This API fetched data from three different queries. The API returns the node information, all incoming relations and all outgoing relations.

6. User Interface Design

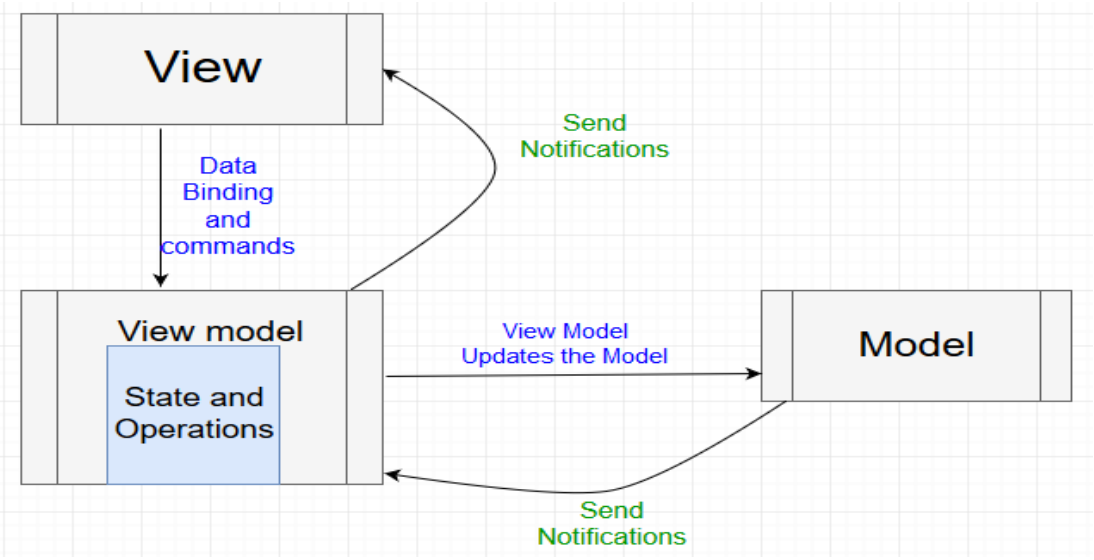
6.1 Overview

The product is designed based on the Single Page Application (SPA) architectural style of web application. Single page applications allow to fit entire application content into single page. Traditional web application performs entire page refresh for each functionality. Single page applications work on the Asynchronous JavaScript and XML (AJAX) principle and refresh only part of the page where dynamic functionality is expected. Several benefits of single page applications include, performance improvement in page loading time, user friendliness and allows dynamic functionality like filtering on the fly without any performance compromise. Single page applications can be implemented using several popular frameworks like Angular, React.js, Vue.js, Ember.js etc. This product is built using the one of the leading and trending framework - “Vue.js”. The choice of Vue.js was based upon rational decision and comparative analysis between popular frameworks. Some of the benefits Vue.js are listed below:

- Vue.js is based on Model View View-Model (MVVM) design architecture
- Vue.js uses virtual DOM for fast rendering of the content
- Provide reactive and composable view components [3]
- Maintain focus in the core library, with concerns such as routing and global state management handled by companion libraries [3]
- State management is achievable using Vuex

6.2 Client-Side Architecture

Front end design of this product is based upon Model View View-Model (MVVM) design pattern. MVVM provides separation of concerns between the user interface controls and the business logic. There are three core components in the MVVM pattern: the model, the view, and the view model. The following illustration shows the relationships between the three components.



Source: https://msdn.microsoft.com/en-us/library/hh848246.aspx

“Some of the benefits of MVVM pattern are

- Isolated unit testing
- The components are decoupled from each other.
- Components can be swapped
- Internal implementation to be changed without affecting the others
- Components to be worked on independently” [4]

6.3 View Components

This product is developed in Vue.js which is based upon MVVM principle. Hence, the application is divided into several prime components which are rendered on the DOM dynamically and delivers ultimate user experience. The prime component regarding this product are listed below.

6.3.1 List of domains

This component is responsible for sending GET request to the server for fetching all the research domains available in the database. The server sends the JSON response which includes information such as Domain Id, name and count of papers available in that domain. This response is parsed and displayed in the List of Domains component.

6.3.2 List of papers

This component is responsible for sending GET request with query parameter of domain id to the server for fetching all the papers belonging to the given domain. The server sends the JSON response which includes information about all the papers where each paper contains information such as paper id, title, list of authors, year of publication, and number of citations. This response is parsed and displayed in the form of paginated table.

6.3.3 Information about Paper

This component is responsible for sending GET request with query parameter of paper id to the server for fetching all the information belonging to the given paper in the database. The server sends the JSON response which includes information about the papers where each paper contains information such as paper id, title, list of authors, year of publication, and incoming and outgoing relationships with other papers.

6.3.4 Visualization of the relationships

This component provides network graph visualization of the relationship between the papers. It uses the same JSON response received in the “Information about the Paper” component. The network graph is built upon the information about the incoming and outgoing relationships in the JSON response.

6.3.5 Information about relationship

This component is responsible for sending GET request with query parameter of relationship/link id to the server for fetching all the information belonging to the given relationship in the database.The server sends the JSON response which includes relationship information such as relationship id, source paper for the relation, destination paper for the relationship, and user feedback for the particular relationship.

6.3.6 User feedback

This component lists out all the previous user comments on a particular relationship. It uses the same JSON response received in the "Information about the relationship" component. Moreover, this component allows users to add new comments. New comments are displayed on the screens and posted to the server as well.

6.4 External libraries

6.4.1 Axios

Axios is a NPM library used primarily for the HTTP requests to the Server. Axios is a Promise based HTTP client for the browser. It makes XML HTTP Requests to the server and automatically parses JSON responses.

6.4.2 Vue-Router

Vue-router is the official router for Vue.js. It deeply integrates with Vue.js core to make building Single Page Applications with Vue.js a breeze. "Some of the Features include:

- Nested route/view mapping
- Modular, component-based router configuration
- Route params, query, wildcards
- View transition effects powered by Vue.js' transition system
- Fine-grained navigation control
- Links with automatic active CSS classes
- HTML5 history mode or hash mode
- Customizable Scroll Behavior" [5]

6.4.3 Bootstrap

Bootstrap is one of the popular CSS library developed by Twitter. It offers of the shelf styling for frequent HTML component such as tables, buttons, navigation bars etc.

6.4.4 Cytoscape Visualization of network graph

Cytoscape is a JavaScript visualization library which specializes in visualizing molecular interaction networks and biological pathways and integrating these networks with annotations, gene expression

profiles and other state data. The network graph display requirement is well satiated by using Cytoscape library.

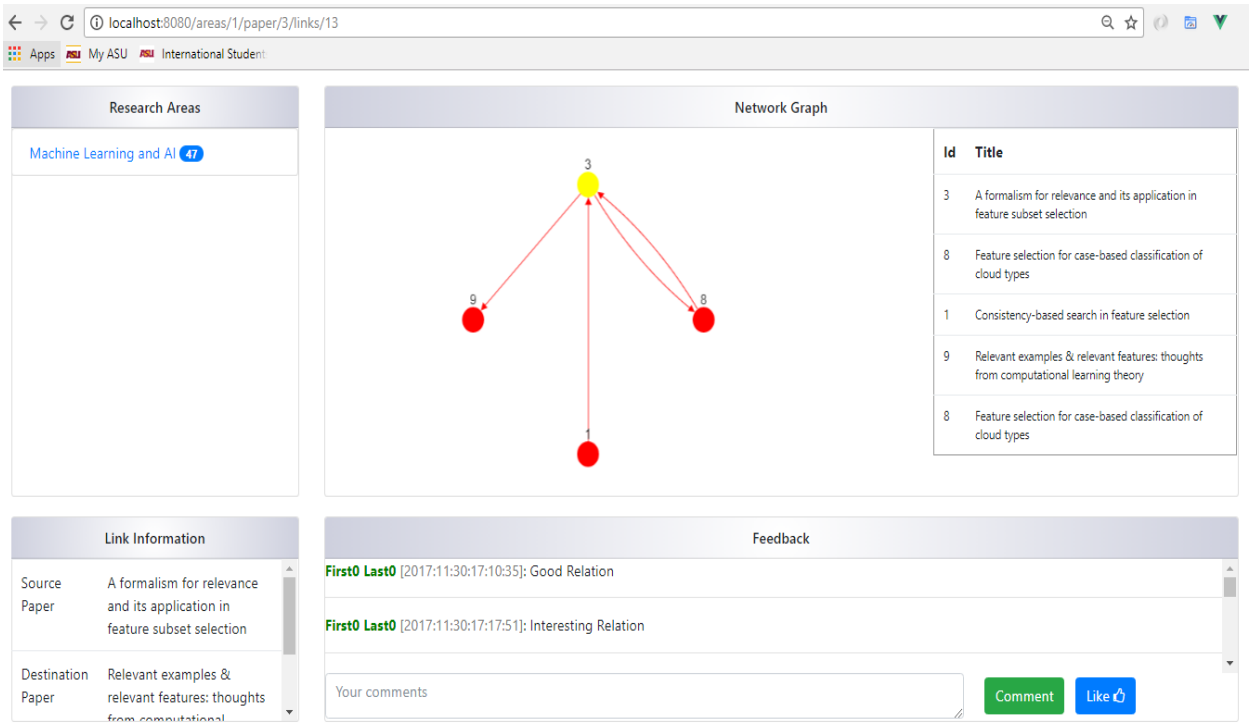
6.5 User Interface Views

Following are the UI snapshots that we designed for this project:

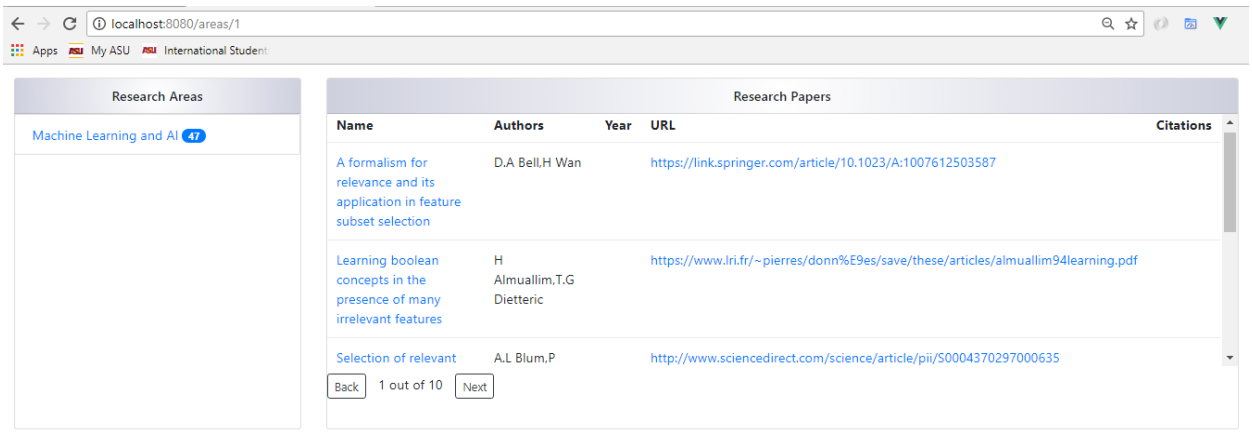
Main Page: This is the first web page viewed by a user. The user can search for a paper or select a category to view the papers in that category. This page also has a login link for the user. Users who don't login can only view but cannot make any contributions (comments, upvotes/downvotes).

Graph page: This page displays the main graph where each node of the graph represents a paper and the edges represent the relationship between each paper. An authenticated user can comment on the relationship between two nodes. They can also upvote/downvote the relationship signifying the relevance of the relationship.

On hovering the mouse over the graph node, we can check out the paper-title, author name and a link to the paper. The section on the left side of the screen contains a list of all the papers that are being displayed. And the section at the bottom of the page contains the comments made by the users on the relationship between two papers. On hovering the mouse over an edge of the graph, the number of upvotes and downvotes are displayed. If the user clicks on a node, the graph reloads with the selected node as the center and the surrounding nodes are the papers relevant to the selected paper.



View all related nodes: It's not possible to display all the papers related to a selected paper in the graph, for that very reason, we created the following page where the user can view all the papers related to the selected paper. The user can select any paper and view the information for that paper along with the graph containing the related papers.



7. User Stories/Tasks

Sprint 1:

Task: Study and compare Graph visualization libraries
We needed graph visualization libraries to display the papers and their relations in the form of graph. Once the data is fetched from Neo4J, we will use a graph visualization library to display it on the User Interface. For this we conducted research on different graph visualization libraries like Reactive Charts, eCharts, C3.js, Canvas JS, Cytoscape .js, Chart.JS etc. We shortlisted D3, Alchemy and Cytoscape for final comparison. After considering multiple points, and creating POCs, we decided to go ahead with Cytoscape since it met our requirements more closely.

Task: Study about graph databases
Main requirement of our project is to establish a graphical relationship between research papers for which we required a graph database. Since the graph databases fall under NoSQL database, we studied how does these databases work. We study distinct types of NoSQL databases like key-value stores, column stores, document stores and Graph databases. Since our main concern was Graph database, we studied them in detail.

Task: Comparative Analysis of Different Graph databases
We compared different graph databases like, SAP Hana, Neo4J, Infinite Graph, gStore, Mark Logic, Spark see, Cayley, Arango DB. We considered factors like performance, licensing and language support for our comparison. After all considerations, we decided to go ahead with MongoDB

Task: Study on getting data from google Scholar
As per initial discussion with our sponsor, we were supposed to get the data from google scholar. We tried to crawl google scholar and detected that they have a threshold for data access and they block the IP address after exceeding the threshold.

Task: Study how users will be authenticated using open authentication
We planned to implement OAuth for ease of access and user experience. We researched on OAuth 2.0 and decided to use it for our application.

Task: Integrating Middleware Server in Node Express with MongoDB (POC)
We started the development at server side with integrating our Middleware server with MongoDB at the backend.

Sprint 2:

Task: POCs on different Graph Visualization techniques
We are not completely sure on which Graph visualization technique to use for our project. So, we went with POCs on D3, Alchemy and Cytoscape and compare their results relatively. This task in continuation from Sprint 1 as we needed to be completely sure on choosing one out of them.

Task: Data Collection
To test our application prototype, we planned to build it on static data collected manually as confirmed by our sponsor. So, we did an extensive search collected 50 different research papers from various sources over the internet.

Task: UI Mockups
We made few sample mockups to pitch our UI interface design to the sponsor and other reviewers. It was done to mainly convey about our idea to other reviewers.

Task: Login/Authentication
There is different architectural flow which can be implemented for OAuth 2.0(server side and Client side). We had to research and start with the POC and compare on which would be suitable for our requirement. We planned to start with Google OAuth and later on implement with other service providers.

Task: Neo4j Setup
We started with setting up and configuring Neo4j DB at the backend where the data will be stored. It will be used to store the research papers with a graphical format which can be queried and retrieved to fetch relevant data.

Task: Node Express and Neo4j integration
This task was integrating our middleware server with Neo4j DB at the backend and query it to fetch results. It mainly started with POC phase and later on was developed on top of it.

Task: POC on Elasticsearch
We are planning to have a search functionality in our UI which will be server by ElasticSearch on the backend. We completed with research and POC on the Elasticsearch. We may not have this functionality in our prototype but will have it in our final application.

Sprint 3:

Task: MongoDB Schema
MongoDB was mainly used to store the user relation information and the activities he/she performed. It includes storing the relations, upvotes, downvotes, comments, etc. We completed with designing the schema for the DB.

Task: Hosting Infrastructure
We are seeking for a stable hosting platform where we can host our infrastructure. We researched on few of the cloud platforms such as AWS and Heroku, but they provide infrastructure with lower configuration at free cost. We might go with AWS with several accounts.

Task: Load data
We have loaded all the static data which was collected manually into the Neo4j DB. We will be using this data for our presentation.

Task: Display List of Research Domains
This component fetches all the research domains in the database and displays a list of it. It also displays the count of papers available for research domain in the database.

Task: Display Papers in a research domain
This component fetches all the research papers belonging to domain and display them in a paginated table where each paper has information such as paper title, author names, year of publications and number of citations.

Task: Display Information about the Paper
This component loads a information box where information about the selected paper is shown. Each selected paper has information such as paper title, author names, year of publications and number of citations.

Sprint 4:

Task: Name and logo for the website
We used CSS animations and HTML to display name and logo of the application on the landing page of the web application.

Task: Display visualization of paper relationships
This component fetches the information about incoming and outgoing relationships and displays them in the form of network graph. This interactive graph allows user to perform click events on the node and edges of the graph.

Task: Display Information about the relationship
This component loads a information box where information about the selected link/relationship is shown. Each selected link has information such as source paper, destination paper, number of upvotes/strength of the relationship.

Task: Upvote / Downvote relation between papers
This component allows user to upvote or downvote any relationship. The changes will reflect in the strength of the relationship.

Task: Add comments on the relation between papers
This component allows users to view the comments on the relationships and allows to post new comments on the relationships.

Task: Develop API for consuming OAuth token
This task is associated with developing API for redirecting users to google login page and consuming the token from the call back. Currently, we are not including this part in the working prototype as this is not a priority requirement, so we are building on top of static user profiles.

8. Installation Guide

8.1 MongoDB

8.1.1 Download and Installation

For Windows: https://www.mongodb.com/download-center#enterprise

For Linux: https://docs.mongodb.com/v3.0/tutorial/install-mongodb-on-ubuntu/

8.1.2 Setup and run DB

Create a new database named - showMe

8.1.3 Load data in DB

- Open terminal.
- Clone the following repository from github:
  - git clone https://github.com/shinigami1392/Show-Me-Related-Work.git
- Go to the following path: $ cd ./server/Show-Me-App/data/
- Follow the instructions given below to load data into mongoDB (Make sure that you have created a database named showMe)
  - $mongoimport --jsonArray --host 127.0.0.1 --db showMe --collection usermodels --file ./sampleUsers.json
  - $mongoimport --jsonArray --host 127.0.0.1 --db showMe --collection domainmodels --file ./sampleDomains.json
  - $mongoimport --jsonArray --host 127.0.0.1 --db showMe --collection relationmodels --file ./sampleRelations.json
  - $mongoimport --jsonArray --host 127.0.0.1 --db showMe --collection papermodels --file ./samplePapers.json

8.2 Neo4j

8.2.1 Download and Installation

Follow the following links for downloading and installation:

For Windows:

Download Link: https://neo4j.com/download/

For Linux:
Download Link: https://neo4j.com/docs/operations-anual/current/installation/linux/debian/

8.2.2 Setup and run DB

- Run Neo4J Community edition and click on Start
- Now browse to the URL - http://127.0.0.1:7474/browser/
- Run the query -: server connect
- Enter “neo4j” as the username and password
- Set the new password to “neo4j123”

8.2.3 Load data in DB

Following steps have to followed for loading the data into Neo4J database (Presuming that you have cloned the Git repo provided in section 2.1) :

- Start the Neo4J server
- Open the data file provided at the location - LOCATION HERE
- Copy the contents of the data file and run them as a query in Neo4J (The data should be loaded in the Neo4J dB now)

8.3 Node Server setup

8.3.1 Prerequisites

- Install Node using the following link**:** https://nodejs.org/en/download/

8.3.2 Installation and Configuration of server

Follow the given instructions below (Presuming that you have cloned the Git repo provided in section 2.1):

- Open terminal and go to the path of Git repo
- Run $cd .\server\Show-Me-App\
- Run $npm install
- Run $npm start. (Your server should be starting now)

8.4 Front-end setup

8.4.1 Prerequisites

Install Node using the following link: https://nodejs.org/en/download/
We need Node package manager(NPM). Latest node version comes with NPM.

8.4.2 Installation and configuration

Follow the given instructions below (Presuming that you have cloned the Git repo provided in section 2.1):

- Open terminal and go to the path of Git repo - Front End development
- Run $npm install
- Run $npm run dev. (This will serve the application on http://localhost:8080)

9. Testing

9.1 Frontend Testing

Cross Browser Compatibility

- This application is tested on different browsers including Mozilla Firefox, Google Chrome and Internet Explorer.
- The performance of the application was up to the expectation when tested against all the browsers.

Component Testing

The client side of the application is developed using component based framework- Vue.js. For each component, we performed simple functional testing in different browsers.

- *List view component* - It is a reusable component which can populates list of data binded to it. This component was rendered without any issues on all the browsers.
- *Graph Visualization component* - This component renders the graph of the papers and its outgoing and incoming relationships with other papers. This is a dynamic component and after interactive function on the graph such as click events on nodes and edges. This component was rendered without any issues on all the browsers.
- Feedback Component - This component allows users to comment on a relation. This component was rendered without any issues on all the browsers.

9.2 Backend Testing

Neo4J testing:

- Tested the data in Neo4J by executing multiple test queries. Also, tested for the validity of regions between papers.

MongoDB Testing:

- Tested the data in MongoDB by executing multiple test queries.

- Tested storage and retrieval of comments and likes.

API / Server Testing:

- All the APIs were thoroughly tested for corner conditions. Tested server of unhandled error conditions. Made sure that all 300, 400, and 500 level errors are handled and tested them. Checked if there are any orphan URLs in the system. Tested all get post and put methods.

10. Links

GitHub: https://github.com/shinigami1392/Show-Me-Related-Work

SCURM Board Taiga: https://tree.taiga.io/project/adutta14-showme-show-related-work/

Video Link : https://youtu.be/xic1KvpI9rA

11.References

[1] Software Engineering: Ninth Edition by Ian Sommerville

[2] The Personal Software Process(PSPSM), November 2000 by Watts S. Humphrey

[3] https://vuejs.org/v2/guide/comparison.html

[4]https://msdn.microsoft.com/en-us/library/hh848246.aspx

[5]https://www.npmjs.com/package/vue-router