SRH University Heidelberg

Project Report

Student Progress Tracking Tool

Authors:

Rohan Raj, 11011896

Sachin Vaidya, 11011995

Sanika Medankar, 11011861

Sarthak Manas Tripathy, 11011868

Contents

1. Introduction 1
2. Organization 1
2.1. Student / Role / Tasks (who was responsible for which tasks): 1
2.2. Organization of Teamwork: 2
2.3. Tools used for project organization: 2
3. Use Cases 3
4. Databases 6
5. Data Models 6
5.1. Mapping of requirements to data models 6
5.2 Graph Database model (Neo4j) 9
5.3. Document data model (MongoDB) 9
6. Implementation 11
6.1. Document-based Data model implementation 11
6.2. Graph Database model implementation 12
7. Queries 13
8. Evaluation 20
9. Bibliography 21

1. Introduction

The idea to track self-progress is the root thought to develop a “Student Progress Tracking Tool”. As the name suggests the basic purpose behind developing this tool is to track the progress of oneself, analyze as well as look into the aspects which need improvement like skills and connect with alumni for help. A student can start with monitoring the progress from the time he/she has applied to the university. Right from the application status to skills analysis, the monitoring is done for a particular student. He/she can also compare the additional skills required for a particular job profile compared to the available skills. Apart from the self-analysis, the student will get an overview of the course and the details of the modules and assignments to be covered as part of the course. Even after the student graduates, he/she remains an alumnus of the university. This creates a network of current students and alumni. This will be beneficial for the student to get connected with the alumni for any guidance required. The student status can also be viewed by a responsible professor. From the professor’s perspective, he/she can guide the students to the right track if required. Also, the details of the professor concerned for a particular module can be seen by the students. In case a student needs any help, he/she can fix an appointment with the professor.

This SPTT data model is a prototype implementation of an app’s backend structure which will help in building an actual app with the same use case and based on the specific requirements. For the same, the dummy data from identified data sources is collected in .csv and. JSON formats. And then inserted into document-oriented and graph databases. However, once functional the app will rely on these NoSQL databases and the data model prototype’s implementation for all its functioning.

The app is planned to be created as a Flask app using python3 in future. It will have 2 groups of users: student and lecturers. The student will have 3 different roles: applicant, student, and alumni. This app could be a step further from this project’s implementation after the model is successful.

2. Organization

2.1. Student / Role / Tasks (who was responsible for which tasks):

Tasks:

- Understanding the empathy of the students during their time at universities. Providing them with a tool that could help them plan, improve and take help from fellow students or alumni or Professors was the sole motive of the use case “Student Progress Tracker”.
- User Story Creation:
  o 15-30 minutes of individual brainstorming to think of the possible user stories.
  o Putting forward the user stories and group discussion on all the user stories that aptly suits the use case.
  o Merging of similar user stories, rejecting a few irrelevant/less important ones and finalizing a list of 10 user stories which was distributed among the team members.
  o Explaining or writing the description for each of them by the respective team member.
- Attribute Specification:
  o Creating entities and their related attributes/properties per each user story on the whiteboard.
  o Finally amalgamation of all the attributes for similar entities in a single specification sheet.
- Database Selection:
  o Identification of the NoSQL database for a specific user story with justification.
- Allocation of user stories:

o Assignment of 2 user stories implementation to each member of the team and laying of the basic structure of the database.

- Implementation of Database models:
  o Implementation of document and graph database keeping in mind the merits of each and integrating both for the implementation if required.
  o Also, the business rules and CRUD operations logic was laid out for the correct implementation as well as mimic the application level mock-up.
- Query formation:
  o Finding the right query to fulfill the purpose of user stories.

The above tasks were done by all the members and the work distribution of rest is as follows:

Student Name	**Assigned User Story**	**Sections of Report**
Rohan	6 and 7	2.3,3.1,5.3,6.1,7,8,9
Sachin	1 and 8	3,3.1,4,6,7,8,9
Sanika	2 and 5	1,2.2,3.1,5.2,7,8,9
Sarthak	3 and 4	2.1,3.1,5.1,6.2,7,8,9

2.2. Organization of Teamwork:

- Team meetings were conducted regularly to track the status of the development and all the meeting details were logged in the Trello application.
- The elements of the meeting details recorded were:
  - Date and time of the meeting
  - Number of members attended the meetings
  - Mode of the meeting and place (either in group rooms or via Skype meetings)
  - All the to do, in progress and need to be done tasks were documented in google drive
- Following is the link to the board of MoM's in trello
  https://trello.com/b/HSWvYYnc

2.3. Tools used for project organization:

- Trello — Trello is a task management app that gives you a visual overview of what is being worked on and who is working on it. This is also used to maintain all the meeting details including the Suggestion given by the professor and team members. (1)

- Google's drive — Google Drive allows users to store files on their servers, synchronize files across devices, and share files. All the files like Database dump, Demo files (JSON, CSV) which can access and modified simultaneously, all the reference documents are stored in google drive.

- Google Docs, Google Sheets, Google Slides — Google Docs is used to maintaining each user story queries and used for documentation of each step performed for creating both Neo4J and MongoDB database (Version controlling). Each member can edit with others at the same time using docs. Google Sheets is used for the Attribute's specification which gives the

details information of all the variables and attributes of the user stories. Google Slides are used for topic presentation, status report presentation, and final report presentation.

- Skype, WhatsApp — Skype allows users to communicate over the Internet by voice, by video and by Instant messaging. Skype meeting is also carried out for a virtual meeting. This is also used to share the progress of individual and getting peer review of that work. WhatsApp used to communicate when we should book our meeting time and place for project work and if there is any task pending then group members can notify each other

Tools used for the project:

- Neo4j Desktop — Neo4j Desktop is an easy and convenient way for developers to work with local Neo4j databases. Neo4j Desktop provides scripts for creating dump/backup the database and also to import the database. It also provides the scripts for checking database consistency. (2)
- Neo4j Browser — The Neo4j browser is a graphical user interface (GUI) that can be run through a web browser. The Neo4j browser can be used for adding data, running queries, creating relationships, and more. It also provides an easy way to visualize the data in the database. An alternative to this is Cypher Shell. Cypher Shell is a command-line tool that is installed as part of the product. You can connect to a Neo4j database and use Cypher to query data, define a schema or perform administrative tasks.
- Mongo Shell — The mongo shell is an interactive JavaScript interface to MongoDB. You can use the mongo shell to query and update data as well as perform administrative operations. MongoDB is the primary daemon process for the MongoDB system. It used to create a server instance. (3)

1. Robo 3T — MongoDB GUI with embedded shell, allows you to interact with your data through visual indicators instead of a text-based interface. This open source tool has cross-platform support and actually embeds the mongo shell within its interface to provide both shell and GUI-based interaction. Robo 3T also has log windows where one can see all the task or query performed by the individual. (4)

- dB Schema — dB Schema is a diagram-oriented database tool compatible with all relational and many NoSQL databases, like MongoDB. dB Schema used for interactive diagrams and pdf/HTML documentation. DbSchema is reading random documents from the database and builds out of it a virtual schema. The schema is represented as diagrams. (5)

3. Use Cases

Due to the large scope of the project each and every individual of the team had different ideas to pitch in, there were new features told from the perspective of the teammate who desired new capabilities. Since the ideas were enormous there was a need to split them into many smaller user stories so that all the specifications and requirements could be selected and care was taken to avoid stories and ideas that may be trivial or unrelated.

Here in these user stories, it was decided to address all possible scenarios a student may encounter. Starting from him being an applicant and checking the status, to him being a student and checking his module, grades, assignments till his last days in university. Some additional stories that surfaced during the discussion were about finding jobs or getting in touch with alumni. On further discussion within team ideas where chalked down on implementing these ideas into the user stories. Since

professors and students go hand in hand, the basic idea was to ensure some user stories covered the work of professor as well. Hence the stories regarding assignments, thesis, internship and many other topics that may interest the professor or lecturer were included.

User Story 1:

"A student has applied to a university and now wants to know what is the status of his application and what are his chances to get an admission."

Once an applicant has applied to University there will be an eagerness for any applicant to know his status after some time. Sometimes the applicant may even like to know the probability of his selection. In order to address this situation, the team came up with a user story to make up for it. Aspiring students can have a look at their application and follow it all the way until the end of his selection process. During the discussion, it dawned upon the team to provide statics related to previous year enrolment. Details such as the GPA, work experience, domain of their experience, certifications if available. Using all these details an applicant can have a guess at his probability of being selected. Maybe the applicant can view all the previous entries or an aggregate of some entries to set a benchmark for himself an understand his foothold.

User Story 2:

"As a student, I want to know Module distribution for a semester, the lecturer concerned & his/her skills and proficient areas. This would ease my course and plan accordingly."

A student should be able to get the details of all the possible subjects he/she will be learning in the corresponding semester/course. They can get a possible layout or broken down details about the course, topics, professor/lecturer taking the subject, amount of hours this subject will be scheduled and the number of private hours required to get you on higher ground with respect to the subject. Having a look at this he/she can plan his schedule accordingly and make up for the subject. Also, the student can find out the professor/lecturer concerned for a particular module or skills.

User Story 3:

"As a student, I can update the status of all the exercise related to each module and also able to set a reminder for the module submission."

Students can have a view of all the exercises that are available for them in the current semester. Students can have a track of their progress in respective subject/exercises and set a status of their completion. They can even set a reminder regarding last dates or submission dates for the exercise of their choice so that they are on time every time.

User Story 4:

"As a professor/lecturer, I want to view my modules, students enrolled for that module and grade their submissions to have proper accountability and see the progress of the students."

Professors/lecturers need to have a system where they can view the courses and their corresponding modules in a semester wise view. This system would help them find the modules assigned to them, help them view the students in a particular course and module. This also could help them find the defaulters who are enrolled in a module and have not made any submission post the last date of Submission. Most important of all they could view and grade any submissions done. Also, it would help find the group submissions effectively and identify a group of students for a particular assignment of the module.

User Story 5:

"As a student, I want to see the proficiency of my skills in a particular module, it will give me insights of my progress for the same."

A student can see the list of all the subjects which are to be completed as part of the course. In the case of completed modules, there should be an analysis (eg in the form of grades, inputs from professors) as which skills need improvement. Also, a student can square down on the skills he/she is strong in. (beneficial for job or internship search).

User Story 6:

"As a student, I want to fix a meeting or get in touch with the skilled expertise (can be student/prof/alumni) to get the information related to the topic or technology."

It's always a challenge for the students to find the information when they are studying a new topic or conducting research. Students don't know where to look and how to proceed. For this, the application can provide the details of whom should they contact. The contact person will be related to the student searched keyword. A student can also fix the meeting with the contact person. There can be different types of meetings scheduled:

	One to One	Group
Online Mode	Meetings between a student and contact person via Online mode. (Conversion can also be stored *)	A group meeting where a group of students and contact person can discuss online. (Conversion can also be stored *)
Offline Mode (Only logged the meetings information)	Meetings can be arranged between a student and contact person on their own.	Group can decide themselves how to arrange a meeting.

User Story 7:

"As a student, I want to know which career I should move into with my current skills, what additional skills do I need to get there."

A repository having students' skills/area of expertise, experience, the field of study, degree type and the job description (with the students' consent), will always motivate the current student to guide or suggested the career path. A student can also find jobs based on job title keyword, and the resulting job list should be correlated to the search keyword (Job title). A student will be able to view the jobs related to his profile. Students profile After choosing the job, the student can know which skill is required and which are he should focus more.

User Story 8:

"As a student, I want to have an Alumni network where I can bond with previous batches' students and get help in studies, internship/job recommendations."

Having a good network is essential for any part of the day. In order to address this, a model was introduced which takes into account the possible data a student may need to get in touch with alumni. There are many relations that arise from alumni data, some of those relations are quite essential. Using this user story some relations or connections can be retrieved, outputs such as all

possible alumni that have worked in a company or are working in a company, giving the required student the information about whom to connect with in order to proceed. This was a core idea of the team for this user story and the idea was to provide the upcoming and present students with a platform where they can easily link with the alumni. Building a network takes long duration and sometimes without a proper connection making a network may be a troublesome task. Hence this user story gives a stage to connect with alumni or at least have basic knowledge about the past students and their current employment details.

The requirements of all the above user stories are explained in section 5.1 i.e., the Mapping of requirements section.

4. Databases

MongoDB is a free and open source distributed database. MongoDB is a document database where basically it stores data in flexible, JSON like documents, meaning fields can vary from document to document and data structure can be changed over time. Since it is distributed database at its core so high scalability and horizontal scaling are built in for ease of use.

Unlike other databases, in Neo4j relationships take first priority in graph databases. This means your application doesn't have to infer data connections using things like foreign keys. A graph is composed of two elements: a node and a relationship. Each node represents an entity (a person, place, thing, category or other pieces of data), and each relationship represents how two nodes are associated. This general-purpose structure allows you to model all kinds of scenarios – from a system of roads to a network of devices to a population's medical history or anything else defined by relationships.

In the data model which the team has proposed is a combination of MongoDB and Neo4j. During the meeting and brainstorming sessions of the team, there were some benchmarks laid out. One of the main factors in benchmarking was with respect to present and future. How will the data model suffice with the current situation and in the upcoming days? What are the necessary details or conditions to be taken care of? Having these conditions in the back of our brain, the team has made certain decisions and accordingly picked the databases for the data models. For example, the student or applicant details are in MongoDB, keeping in mind that the requirements change for students over a period of time and to have flexibility with new demands NoSQL approach would justify this selection. Similarly, if the alumni database is considered, with each passing year the number of alumni increase and the relations also increase. To keep up with the relations and faster retrieval of data, having Neo4j over here makes sense.

5. Data Models

5.1. Mapping of requirements to data models

After the user stories creation and its description have been discussed and finalized, the entities and their respective properties per user story were jotted down on a whiteboard. As our use case did not demand strict schema structure, as well as the data, were to increase multiple folds in the years to come and also a need to cater a great number of users, the 2 NoSQL database: MongoDB and Neo4j were used to persist data. Also, it was decided that which data should be retrieved from which database based on their merits as well as to ease and lower application level joins for CRUD

operations. This was followed by combining all the properties (both common and distinct) for a particular entity in one attribute specification sheet.

The following are the screenshots of the attribute specification sheet:

US3	Mongo				
Variable	**Module**	**Assignment**	**AssignementStatus**	**assignmentReminder**	**Student**
Attributes	moduleId	assignmentId	dateOfSubmission	periodicity(everyday/weekly/monthly)	studentName
	blockDistribution	title	remarksFromProf	reminderType(push/email/login/popup)	studentId
	semester	lastDateOfSubmission	ratingsByProf	reminderType	
	ectsPoints	description	assignmentId	dateOfReminderSetup	
	moduleDuration	dateUploaded	submitted	reminderStatus	
	moduleDescription		studentId		
			assignmentReminder		
US4	Graph				
Variable	**Lecturer**	**Assignment/Exercises**	**Submission**	**Student**	**Module**
Attributes	lectName	assignmentId	assignmentId	studentName	courseId
	lectId	lastDateOfSubmission	studentIds	studentId	moduleId
	lectEmailId	description	submissionId	yearOfEnrollment	moduleName
		moduleId		emailId	moduleStatus
		type			
		title			

Fig. 5.1.1: User Story specific entity and attribute list

Variable	Student	Lecturer	Course	Applications	Criteria	Module	Reference	Assignment	AssignementStatus	Skills	Meeting	Job	Alumni	Certification	assignmentReminder	Applicant	LearningHours	Submission	Company
Attributes	studentName	lectName	courseId	dateApplied	gpa	moduleId	referenceMaterialId	assignmentId	dateOfSubmission	skillName	meetingId	jobId	alumniName	certificationAuthority	periodicity(everyday/weekly/monthly)	applicantName	total	assignmentId	companyId
	studentId	skills	courseName	status	workExperience	ectsPoints	literature	title	remarksByProf	skillId	timeStamp	companyId	alumniId	certificationName	reminderType(push/email/login/popup)	applications	class	studentIds	companyName
	yearOfEnrollment	linkedInLink	department	courseId	experienceDomain	moduleDuration	dataSource	lastDateOfSubmission	ratingsByProf		meetingRequestedBy	jobName	skills	certificationDate	reminderStatus	emailId	private	submissionId	companyLocation
	emailId	modules	language		certification	semester	moduleName	description	assignmentId		meetingReceivedBy	jobDescription	courseId	duration	dateOfReminderSetup	applicantId	coaching		
	courseId	emailId	courseType		sop	skills		dateUploaded	submitted		mode (if mode	jobCountry	emailId			studentId			
	skills	location	degreeType		applicantId	moduleDescription			studentId		meetingTopic	jobLocation	yearOfGraduation			alumniId			
		lectId			courseId	blockDistribution			assignmentReminder		MoMs								
					highestEducation	examType					members								
						assignment					type(O-O,group)								
						moduleStatus					conversation								
						learningHours													
						referenceMaterialId													
						courseId													

Fig. 5.1.2: Final Attribute Specification sheet

The Fig 5.1.1 shows entities as variables and properties as attributes for the US3 (user story 3) and US4 (user story 4). This also shows that which user story can be completed using which data model. The camel case nomenclature was followed for property naming convention. This naming was kept consistent for both the MongoDB and Neo4j databases. Only the required properties are recorded for a particular variable per user story, for example from figure US3's "Student" entity has 2 properties while the same entity has 4 properties in the US4.

Fig. 5.1.2 shows the complete entity and all the combined properties of the data models. The same structure was followed as above.

Then as per the data model decided per user story, the data was inserted into respective document-oriented DB (MongoDB) and graph-based DB (Neo4j).

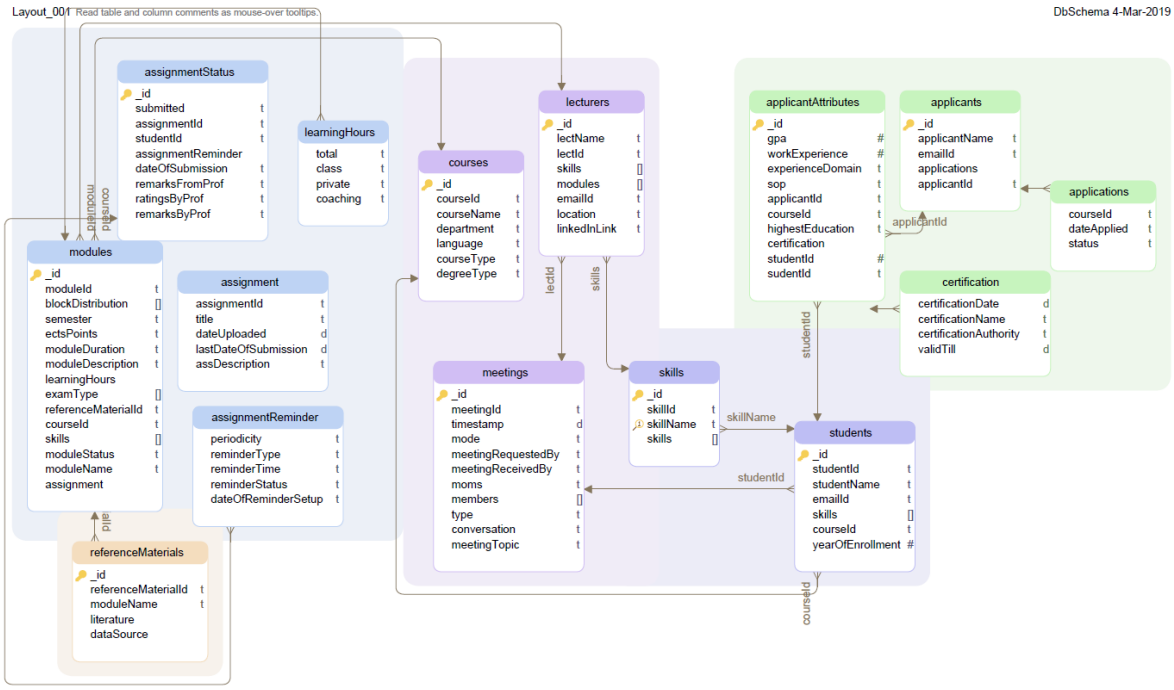

Fig. 5.1.3: MongoDB ER diagram

The structure of the data model can be understood from the following figures Fig. 5.1.3 for Mongo and Fig. 5.1.4 for Neo4j db. For example, the collection “assignmentReminder” is embedded in the collection “assignmentStatus” as both entities have a strong association and their properties are likely to be read in combination regularly, the read operation is faster in case of embedding compared to referencing which would require an application level join.

For the user stories where the nodes need to have some relations with other nodes, the graph database Neo4j was found appropriate. And the schema containing all the nodes and relationships can be seen from the Fig. 5.1.4. Sections 5.2 and 5.3 explains the details of why MongoDB and Neo4j.

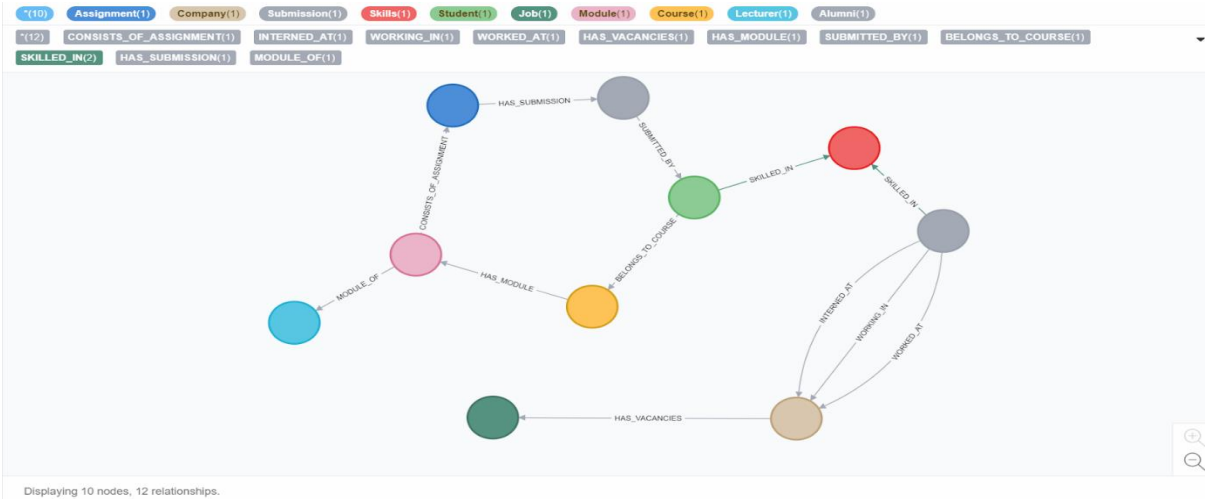

Fig. 5.1.4: Neo4j schema

5.2 Graph Database model (Neo4j)

A graph database is a database designed to treat the relationships between data as equally important to the data itself. It is intended to hold data without constricting it to a predefined model. Instead, the data is stored like we first draw it out – showing how each individual entity connects with or is related to others. (6)

There are few different approaches to what makes up the key components of a graph database. One such approach is the property graph model, where data is organized as nodes, relationships, and properties (data stored on the nodes or relationships).

Nodes are the entities in the graph. They can hold any number of attributes (key-value pairs) called *properties*. Nodes can be tagged with *labels*, representing their different roles in your domain.

Relationships provide directed, named, semantically-relevant connections between two node entities (e.g. Student *SKILLED_IN* Skill). A relationship always has a direction, a type, a start node, and an end node. Like nodes, relationships can also have properties. The use of a graph database is dependent on a particular user story

User Story 4 :
For this user story, the data is stored in graph database because when checking the assignments for a particular student the status of that assignment can be checked easily as the status details are kept as the properties of the relation. If the same details were present in the document database, the query would have been complex as to access all the assignments and then the status of the assignment inside it.

User Story 5 :
For this user story, it is beneficial to keep the data in the graph database so that the modules which are completed can be easily accessed and accordingly the skills will be updated. Also querying related to skills will be easy in the graph database as the queries can be done on relations.

User Story 7 :
In the graph database, it is easy to visualize and compare the skill required for a particular job and the skills a student is good in. There will be less number of comparisons while querying in a graph database, compared to a document database due to the relations between skills, jobs, and student. Also, the student's details like the thesis report and other details will also present so that the prof/lect can also recommend jobs to students comparing the skills used in thesis and internship.

User Story 8 :
For this user story, it is better to store the data in the graph database since it is totally related to the network of the students with the alumni of the university. Therefore the data can be easily retrieved by the queries in the graph through the nodes and the relations between alumni and the other entities of the system.

5.3. Document data model (MongoDB)

A document is a JSON object consisting of a number of key-value pairs that you define. There is no schema in MongoDB; every JSON document can have its own individual set of keys, although you may probably adopt one or more informal schemas for your data.

The key challenge in data modeling is balancing the needs of the application and data retrieval patterns. When designing data models, always consider the application usage of the data (i.e. queries, updates, and processing of the data) as well as the inherent structure of the data itself.

The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data.MongoDB allows related data to be embedded within a single document or Referencing the document to other documents.

- Embedded documents capture relationships between data by storing related data in a single document structure. MongoDB documents make it possible to embed document structures in a field or array within a document. These denormalized data models allow applications to retrieve and manipulate related data in a single database operation.

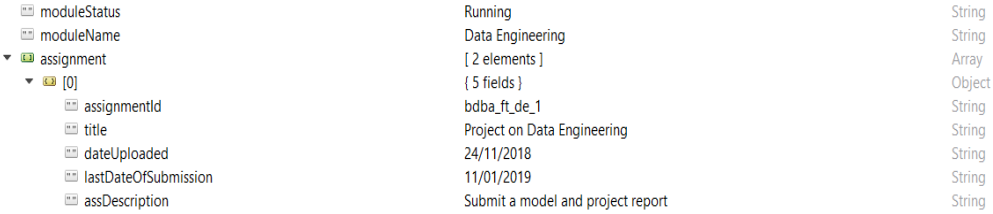

Fig. 5.3.1 Embedded document example

- References store the relationships between data by including links or references from one document to another. Applications can resolve these references to access the related data. Broadly, these are normalized data models.

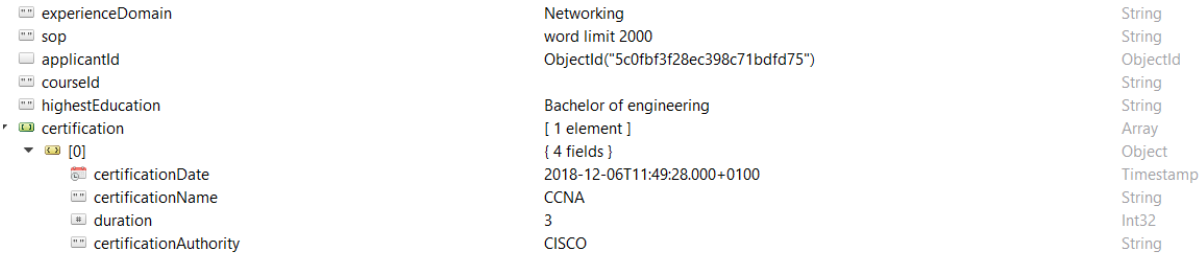

Fig. 5.3.2 Reference document example

The dynamic schema design in MongoDB has advantages of its own. This means allowing us to insert data without a predefined schema. (7) This was exploited in our user story 4 where a student can set an assignment reminder with two cases:

a) As in Fig. 5.3.3a, if submitted = “no” and lastDateOfSubmission greater than equal to the current date and time there will be an assignment Reminder added to the assignmentStatus collection document.
b) As in Fig. 5.3.3b, if submitted = “yes”, the assignmentReminder is not required anymore. So, there is no need to have null values as would have been the case in a relational database.

```
{
    "_id" : ObjectId("5c62cc9c3cb57b3683a54af5"),
    "submitted" : "no",
    "assignmentId" : "bdba_ft_de_1",
    "studentId" : 11011867,
    "assignmentReminder" : {
        "periodicity" : "monthly",
        "reminderType" : "email",
        "reminderTime" : "18:00",
        "reminderStatus" : "active"
    }
}
```

Fig. 5.3.3a With assignmentReminder

```
{
    "_id" : ObjectId("5c05776c5a5b5cfd10ea1bc3"),
    "submitted" : "yes",
    "dateOfSubmission" : ISODate("2018-11-23T11:39:24.234Z"),
    "remarksByProf" : "The work was OK needs improvement",
    "ratingsByProf" : "3.5",
    "assignmentId" : "fs_1",
    "studentId" : 11011868
}
```

Fig.5.3.3b Without assignmentReminder

6. Implementation

Dummy Data:

Data for students and alumni can be obtained from the administration department at SRH University. Since they keep track or record or all the admitted and passed out students getting this information won't be a problem. If the data from alumni need to be updated, then a survey form can be made optional. A survey will work as a mediator between the details available to the updates available. Care should be taken to add conditions such as consent is provided to use the data.

Data regarding professor are taken from SRH University website, details such as name, qualification and LinkedIn details were needed to complete the professor or lecturer details. Using the official website to get these details appeared as a better approach.

Job skill requirement was obtained by searching the jobs in LinkedIn and looking at their requirements in the skills section.

6.1. Document-based Data model implementation

The Dummy data are formatted into JSON format. The Various source from where the dummy data is collected is mentioned in above section 6. MongoImport tool import content from JSON file. For each collection, a separate JSON file is being used. JSON file structure is based on a User story. For example- In user story 1, there are 3 collections used for this story applicant.json, course.json, and applicantAttributes.json.

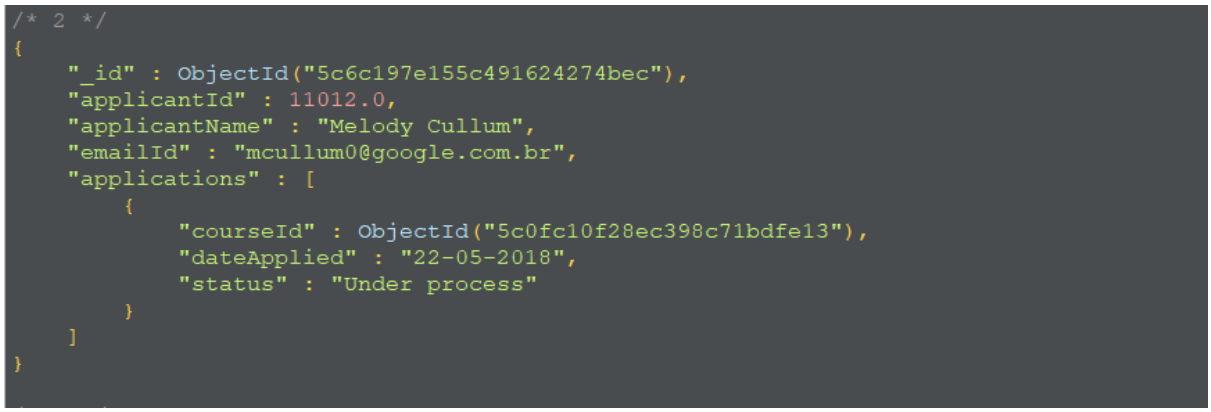

Fig 6.1.1 applicant.json

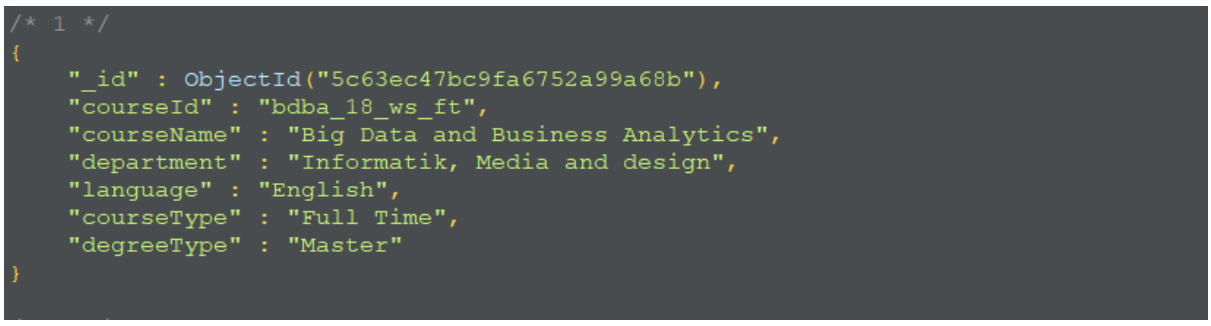

Fig 6.1.2 course.json

```
/* 1 */
{
    "_id" : ObjectId("5c0fc4f528ec398c71bdff3e"),
    "gpa" : 2.0,
    "workExperience" : 5.0,
    "experienceDomain" : "Networking",
    "sop" : "word limit 2000",
    "applicantId" : ObjectId("5c0fbf3f28ec398c71bdfd75"),
    "courseId" : "",
    "highestEducation" : "Bachelor of engineering",
    "certification" : [
        {
            "certificationDate" : Timestamp(1544093368, 1),
            "certificationName" : "CCNA",
            "duration" : 3,
            "certificationAuthority" : "CISCO"
        }
    ]
}
```


Fig 6.1.3 applicantAttributes

Sample loading of JSON file:
#mongoimport --db studentTracker --collection applicant --file applicant.json
#mongoimport --db studentTracker --collection course --file course.json
#mongoimport --db studentTracker --collection applicantAttributes --file applicantAttributes.json

6.2. Graph Database model implementation

The data sourced from various above-mentioned sources were formatted into comma separated files (assumed using some programming techniques). The files were of 2 types: the node and the relationship type. The node type file contained the entity and their corresponding properties with one mandatory unique entity_id. The relationship type file contained data to create a relationship between two or more already created nodes using node type files. As the user stories and related entities that require graph database were outlined in section 5, the graph data model was created using cypher scripts corresponding to the nodes and their relationships.

Sample node file:

moduleId	moduleName	courseId	moduleStatus
bdba_ft_fcfs	First Step Into Case S	bdba_18_ws_ft	Completed
bdba_ft_de	Data Engineering	bdba_18_ws_ft	Completed
bdba_ft_is	Information Systems	bdba_18_ws_ft	Running
acs_pm	Project Managemen	acs_18_ws	Completed
acs_sdp	Software Developme	acs_18_ws	Running

Fig. 6.2.1: module.csv

lectId	lectName	emailId
imd_110	Prof. Dr. XYZ	xyz@srh.de
ibe_123	Prof. Dr. ABC	abc@srh.de
imd_101	Prof. KRA	kra@srh.de
imd_102	Prof. ASD	asd@srh.de

Fig. 6.2.2: prof.csv

Sample Relationship file:

moduleId	lectId
bdba_ft_is	imd_110
acs_sdp	imd_110
bdba_ft_fcfs	imd_101
bdba_ft_de	imd_101
bdba_ft_fcfs	imd_102
ibe_bhr	ibe_123
ibe_bl	ibe_123

Fig. 6.2.3: lecturerModule.csv

Sample loading cypher scripts:
#Load module.cyp
CREATE CONSTRAINT ON (s:Module) ASSERT s.moduleId IS UNIQUE;
//CREATE INDEX ON :Module(moduleId);
//USING PERIODIC COMMIT 1000 //commit after every 1000 lines
load csv with headers from "file:///D://BDBA//Information_Systems//data//module.csv" AS line
CREATE (:Module {moduleId:line.moduleId})
MERGE (module:Module {moduleId:line.moduleId}) SET module.moduleName = line.moduleName, module.moduleStatus = line.moduleStatus, module.courseId = line.courseId

#Load lect.cyp
CREATE CONSTRAINT ON (s:Lecturer) ASSERT s.lectId IS UNIQUE;
//CREATE INDEX ON :Lecturer(lectId);
//USING PERIODIC COMMIT 1000 //commit after every 1000 lines
load csv with headers from "file:///D://BDBA//Information_Systems//data//prof.csv" AS line
CREATE (:Lecturer {lectId:line.lectId})
MERGE (lect:Lecturer {lectId:line.lectId}) SET lect.lectName = line.lectName,lect.lectEmailId = line.emailId

#creates relationship [:MODULE_OF]
load csv with headers from "file:///D://BDBA//Information_Systems//data//lectModule.csv" AS line
MATCH (a:Module),(b:Lecturer) WHERE a.moduleId = line.moduleId and b.lectId = line.lectId CREATE (a)-[:MODULE_OF]->(b)

7. Queries

User Story 1:

#To check the status of applicantion
>db.getCollection('applicant').find({applicantName: "Sachin Vaidya"},{"applications.courseId": 1, "applications.status" : 1})
#To check the status of all applicant for a course
>db.getCollection('applicant').find({"applications.courseId":"bdba_18_ws_ft"},{"applications.status" : 1})
#Also check the dates on which they have applied
>db.getCollection('applicant').find({"applications.courseId": "bdba_18_ws_ft"},{applicantId:1,"applications.dateApplied": 1,"applications.status" : 1})
#Check statistics of students that are selected
>db.getCollection('applicantAttributes').find({studentId : { $exists : true}}, {gpa : 1, workExperience : 1})

User Story 2:

Semester wise module details for a particular course: In this, we can display all the needed module details (like learning hours, exam type, ECTS Points) in the module collection.
Case 1 : Get details of all the modules covered in a particular course
> db.modules.find({courseId : "bdba_18_ws_ft"},{"moduleName":1,"_id":1, "ectsPoints" : 1, "semester" :1}).sort({semester : 1}).pretty();
Case 2 : Also can find the modules and its details of particular semester

> db.modules.find({courseId : "bdba_18_ws_ft", semester : "1"},{"moduleName":1,"_id":1, "ectsPoints" : 1}).pretty();
Who is the professor concerned for a module: The needed details can be displayed by setting the attributes to 1 in the query
> db.lecturer.find({modules : "bdba_ft_is"},{"lectName" : 1}).pretty(); # will display only professor name.
> db.lecturer.find({modules : "bdba_ft_is"}).pretty(); # will display all the details of the professor.
Professors concerned for a particular skill :
> db.lecturer.aggregate([{$match : {skills : "MongoDB"}}]).pretty();
OR
> db.lecturer.find({skills : "MongoDB"},{"lectName" : 1}).pretty();
Which are the current running/completed modules
> db.modules.find({moduleStatus : "Running"},{moduleName : 1}).pretty(); # will display only the module name.
> db.modules.find({moduleStatus : "Completed"},{moduleName : 1}).pretty();
How much time to be spend on a particular module for self study
> db.modules.find({moduleName : "Data Engineering"},{"learningHours.private" : 1}).pretty();
Also to display learning hours distribution for a particular module :
> db.modules.find({moduleName : "Data Engineering"},{learningHours: 1}).pretty();
Details of the reference material for a particular module
> db.referenceMaterials.aggregate([{$match : {moduleName : "Data Engineering"}}]).pretty();
Subjects which are overlapping in a block
> db.modules.find({courseId : "bdba_18_ws_ft"},{moduleName : 1, ectsPoints : 1,blockDistribution : 1}).sort({blockDistribution : 1}).pretty();
Or Subjects in a particular blocks
> db.modules.find({$and :[{courseId : "bdba_18_ws_ft"},{blockDistribution : "Block 2"},{blockDistribution : "Block 3"}]},{moduleName : 1}).pretty();

User Story 3:
-Extract the student id and courseId for the Student "Sanika"
db.getCollection('students').find({"studentName":"Sanika"},{ "courseId":1, "studentId":1})
studentId: 11011870, coursed: "bdba_18_ws_ft"
#Exercises
#All the exercises
db.getCollection('modules').find({courseId:"bdba_18_ws_ft"},{"moduleName":1,"assignment":1})
#Exercises upcoming like the one equal to or greater than today's date
db.getCollection('modules').find({courseId:"bdba_18_ws_ft","assignment.lastDateOfSubmission":{"$gte":new Date()}},{"moduleName":1,"assignment":1})
#Status update
db.getCollection('assignmentStatus').find({"studentId":11011870,"assignmentId":"bdba_ft_de_1"},{"submitted":1})
Case-1: If submitted=”no”
There would be no submission node created in Graph database for the student and for that assignment.
Case-2: If submitted=”yes”
The details of submission can be fetched from the Graph databases using the following query:
MATCH(n:Student)<-[:SUBMITTED_BY]-(m:Submission) where m.assignmentId = "bdba_ft_de_1" and n.studentId="11011870" RETURN n,m
#Reminder
Sample Reminder:

{"_id" : ObjectId("5c10f79ac9c7309b25916264"),"submitted" : "no","assignmentId" : "fs_1","studentId" : 11011869, "assignmentReminder" : {"periodicity" : "everyday", "reminderType" : "pushNotification", "reminderTime" : "10:00" }}

-Insert/fetch/update the assignmentReminder

Fetch:

All

db.getCollection('assignmentStatus').find({"studentId":11011870})

Only the reminder array

db.assignmentStatus.find({ "studentId": 11011870}, { "assignmentReminder": 1 })

Update:

Case-1:(update an existing reminder)

db.assignmentStatus.update({ "studentId": 11011870,"assignmentId": "bdba_ft_de_1"},
{ $set: { "assignmentReminder" : {"periodicity" : "monthly", "reminderType" : "email", "reminderTime" : "18:00","reminderStatus":"active"} }})

Case-2:(Remove a reminder)

db.assignmentStatus.update({"studentId": 11011870,"assignmentId":"bdba_ft_de_1"}, {$unset: {"assignmentReminder" :1}})

#schedule events for date check and update status

Case-3:(Change the status of reminder to "*inactive*" if lastDateOfSubmission<Today's date)

#find the assignmentIds lastDateOfSubmission<Today's date

db.modules.find({"assignment.lastDateOfSubmission":{"$lt":new Date()}},{"assignment.assignmentId":1,"assignment.lastDateOfSubmission":1})

#update the reminder using assignmentId of all students who have set the reminder

db.assignmentStatus.update({ "assignmentId": "bdba_ft_de_1"}, { $set: {"assignmentReminder.reminderStatus":"inactive"} },{multi:true})

User Story 4:

#For a module taken by a particular lecturer, find the number of submissions done against the assignments of that module.

MATCH (a:Lecturer)<-[:MODULE_OF]-(b:Module)-[:CONSISTS_OF_ASSIGNMENT]->(c:Assignment)-[:HAS_SUBMISSION]->(d:Submission) where a.lectName = "Prof. KRA" and b.moduleName = "Data Engineering" RETURN a.lectName,b.moduleName,c.assignmentId,d.submissionId,size(d.studentIds) as submission_count

##Count the number of students enrolled in the course and number of submissions done, find defaulters, regular submitters

#student enrolled for the particular course

MATCH(a:Student{yearOfEnrollment:"2018"})-[:BELONGS_TO_COURSE]->(b:Course) where b.courseId="bdba_18_ws_ft" with collect(a.studentId) as list RETURN list, size(list) as count

#submission count for Lecturer and particular module

MATCH (a:Lecturer)<-[:MODULE_OF]-(b:Module)-[:CONSISTS_OF_ASSIGNMENT]->(c:Assignment)-[:HAS_SUBMISSION]->(d:Submission)-[r:SUBMITTED_BY]->(e:Student) where a.lectName = "Prof. KRA" and b.moduleName="Data Engineering" WITH d.submissionId as x,e.studentId as y RETURN x,count(*)

#find defaulters

MATCH(b:Course)<-[:BELONGS_TO_COURSE]-(a:Student{yearOfEnrollment:"2018"}) where b.courseId="bdba_18_ws_ft" WITH collect(a.studentId) as p MATCH (a:Lecturer)<-[:MODULE_OF]-(b:Module)-[:CONSISTS_OF_ASSIGNMENT]->(c:Assignment)-[:HAS_SUBMISSION]->(d:Submission)-[r:SUBMITTED_BY]->(e:Student{yearOfEnrollment:"2018"}) where a.lectName = "Prof. KRA" and

b.moduleName="Data Engineering" WITH p,collect(e.studentId) as y RETURN filter(k IN p WHERE not k in y)
#check evaluation of student's submission query
MATCH (a:Lecturer)<-[:MODULE_OF]-(b:Module)-[:CONSISTS_OF_ASSIGNMENT]->(c:Assignment)-[:HAS_SUBMISSION]->(d:Submission)-[r:SUBMITTED_BY]->(e:Student) where a.lectName = "Prof. KRA" and b.moduleName = "Data Engineering" RETURN a.lectName,b.moduleName,c.assignmentId,d.submissionId,e.studentName,r

User story 5:

List all the modules to be completed as part of a particular course
MATCH (c : Course{courseId : "bdba_18_ws_ft"}) - [r:HAS_MODULE] -> (m : Module) RETURN m.moduleName #Will return only the module names which has to be completed in this course
List all the modules and the assignments in the modules for a particular course
MATCH (c : Course{courseId : "bdba_18_ws_ft"}) - [r:HAS_MODULE] -> (m : Module) - [r1 : CONSISTS_OF_ASSIGNMENT] -> (a : Assignment) RETURN m.moduleName, a.description
Which subjects need improvement for student named 'Sarthak'
MATCH (s:Student {studentName : 'Sarthak'}) <- [r:SUBMITTED_BY] - (a:Submission) <- [r1:HAS_SUBMISSION] - (b:Assignment) <- [r2:CONSISTS_OF_ASSIGNMENT] - (m:Module) WHERE (r.ratingsByProf > '2.1') RETURN (m)
For a student how many modules have I completed/not completed : Since the submission node will only be created after the submission is done therefore the query is done on the SUBMITTED_BY relationship
MATCH (s:Student {studentName : 'Sarthak'}) <- [r:SUBMITTED_BY] - (a:Submission) <- [r1:HAS_SUBMISSION] - (b:Assignment) <- [r2:CONSISTS_OF_ASSIGNMENT] - (m:Module) RETURN (m.moduleName)
List the students who are good in particular skills. This can be useful for combining it with the list of jobs so that it is easy to map suitable students for a particular job profile.
MATCH (s : Student) - [r : SKILLED_IN] -> (s1 : Skills {skillName : 'Python'}) RETURN (s)
Student entity will get updated with the list of skills that were covered in a particular module as and when a student completes that module (Combined Query)
First, find the modules completed by a student. This can be queried in the graph database from the relation generated.
MATCH (s:Student {studentName : 'Sarthak'}) <- [r:SUBMITTED_BY] - (a:Submission) <- [r1:HAS_SUBMISSION] - (b:Assignment) <- [r2:CONSISTS_OF_ASSIGNMENT] - (m:Module) RETURN m.moduleName

Result:
"Data Engineering"
"The first Step into Case Studies"
It is assumed that since a student has completed a particular module, he/she will have the particular skills associated with that particular module. The module has the skills' names stored.
> db.modules.find({moduleName : 'Data Engineering'},{skills : 1}).pretty();
{"_id" : ObjectId("5c7bd5431d97108b2f290378"),"skills" : [Neo4j","Python"]}
Using the skills mentioned in the above list there will be relation created between the student node and skill node with the label: SKILLED_IN
MATCH (s2 :Skills{skillName : 'Python'}) MATCH (s :Student {studentName : 'Sarthak'}) CREATE (s) - [r3 : SKILLED_IN] -> (s2) RETURN (r3)
Since this is the combined query between graph and document database, which can also be done by python script so that the fetched data can be stored in variables and then used for different queries.

The above can be done using python script. In this case all the queries will be executed in one python script. Following are the commands to connect with mongodb and neo4j databases:

```
from neo4jrestclient.client import GraphDatabase
from pymongo import MongoClient
# For mongodb
client = MongoClient('localhost:27017')
dbName = client.studentTracker
# For neo4j
db = GraphDatabase("http://localhost:7474", username="neo4j", password="Graph@12")
module = dbName.modules
q1 = {"moduleName" : "Data Engineering"}
r1 = module.find(q1)
skillIds = []
for i in r1:
        skillIds = i['skills']
        print(skillIds)
```

Output : ['Neo4j', 'Python']
Further use this array “skillIds” to create a relationship between student and skills node.

User story 6:

```
# Find the skill expertise based on skill keyword
MATCH (s1:Student)-[r1:SKILLED_IN]->(s:Skills) where s.skillName=~".*ython.*" RETURN
s1.studentId as Id LIMIT 25 union MATCH (p:Alumni)-[r:SKILLED_IN]->(s:Skills) where
s.skillName=~".*ython.*" RETURN p.alumniId as Id LIMIT 25 union MATCH (l:Lecturer)-
[r2:SKILLED_IN]->(s:Skills) where s.skillName=~".*ython.*" RETURN l.professorId as Id LIMIT 25
#Sample output
```

"Id"
11018534
20170001

```
# Find the skill expertise based on module teaches by professor\lecturer
MATCH p=(a:Module)-[r:MODULE_OF]->(l:Lecturer) where a.moduleName=~".*Case.*" RETURN
l.lectId LIMIT 25
#Before booking the meeting, check skill expertise is available or not
db.meetings.find({"timestamp" : {"$gte": ISODate("2013-10-
01T00:00:00.000Z")},"meetingReceivedBy":"11011869"})
# Book the meeting
Case1 : One to one and online
db.meeting.insert({“meetingId" : "meet1",”timestamp" : ISODate("2019-01-23T11:39:24.234Z"),
"mode" : "online","meetingRequestedBy" : "11011868","meetingReceivedBy" : "11011869",
"moms" : "","members" : [ "11011868",  "11011869"],"type" : "one-to-one","conversation" : "If
online and chat session","meetingTopic" : "Graph algorithm"})
Case 2: Group and online
db.meeting.insert({"meetingId" : "meet2","timestamp" : ISODate("2019-02-
23T11:39:24.234Z"),"mode" : "online","meetingRequestedBy" : "11011870","meetingReceivedBy" :
"11011869","moms" : "","members" : [ "11011868",  "11011870", "11011869",  "11011871"],"type"
: "group", "conversation" : "If online and chat session", "meetingTopic" : "Redis Sharding"})
```

# Update the meeting time
db.meeting.update({"meetingId":"meet33"}{$set:{"timestamp" : "03/03/2019 12:30")}})
#Cancel the meeting
db.meeting.update({"meetingId":"meet33"}{$set:{"deleted" : "yes")})

User story 7:

The user story is implemented using the data extrated from the data model as csv files (8) and then the python script (9) is used to show the similar jobs results for the students' job search .

#Input files are Job.csv and student.csv

All imports

```
import pandas as pd
import numpy as np
import ast
from scipy import stats
from ast import literal_eval
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
```

jobs = pd.read_csv('./input_data/jobs.csv', delimiter=',',encoding='utf-8', error_bad_lines=False)
users = pd.read_csv('./input_data/users_student.csv' ,delimiter=',',encoding='utf-8')

1. Based on job title and description

```
In [40]: jobs['Title'] = jobs['Title'].fillna('')
         jobs['Description'] = jobs['Description'].fillna('')
         #jobs['Requirements'] = jobs['Requirements'].fillna('')

         jobs['Description'] = jobs['Title'] + jobs['Description']
```

Convert a collection of raw documents(title + description of job) to a matrix of TF-IDF features, fit_transform-Learn vocabulary and idf and return term-document matrix.

```
In [41]: tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
         tfidf_matrix = tf.fit_transform(jobs['Description'])
```

Job.csv has total 10000 samples and Total term document calculated is 515311 words

```
In [42]: tfidf_matrix.shape
Out[42]: (10000, 515311)
```

cosine kernel, computes similarity as the normalized dot product of tfidf_matrix.

```
In [43]: # http://scikit-learn.org/stable/modules/metrics.html#linear-kernel
         cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
In [57]: def get_recommendations(title):
             idx = indices[title]
             #print (idx)
             sim_scores = list(enumerate(cosine_sim[idx]))
             #print (sim_scores)
             sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
             job_indices = [i[0] for i in sim_scores]
             return titles.iloc[job_indices]
```

Output:

```
In [75]: get_recommendations('SAP FI/CO Business Consultant').head(10)

Out[75]: 6051                    SAP FI/CO Business Consultant
         5868                      SAP FI/CO Business Analyst
         5351    SAP Sales and Distribution Solution Architect
         1                          SAP Business Analyst / WM
         4796       Senior Specialist - SAP Configuration - SD
         5159                        SAP Basis Administrator
         5075                              SAP FI Consultant
         5117                      SAP Integration Specialist
         4728           SAP ABAP Developer with PRA experience
         4785                    Data Management Functional BA
         Name: Title, dtype: object
```

2. Get similar profile

For finding similar profile, multiple attributes are taken into considered like degreeType, major, totalYearexperience.

```
In [14]: users['degreeType'] = users['degreeType'].fillna('')
         users['major'] = users['major'].fillna('')
         users['totalYearsExperience'] = str(users['totalYearsExperience'].fillna(''))
         users['degreeType'] = users['degreeType'] + users['major'] + users['totalYearsExperience']

In [16]: tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
         tfidf_matrix = tf.fit_transform(users['degreeType'])

In [17]: tfidf_matrix.shape

Out[17]: (1000, 1880)

In [18]: # http://scikit-learn.org/stable/modules/metrics.html#linear-kernel
         cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

Output: The output will be all the similar user's Id (student is the user). userId can used to find the student information.

```
In [22]: def get_recommendations_userwise(userid):
             idx = indices[userid]
             #print (idx)
             sim_scores = list(enumerate(cosine_sim[idx]))
             #print (sim_scores)
             sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
             user_indices = [i[0] for i in sim_scores]
             #print (user_indices)
             return user_indices[0:11]

In [76]: print ("-----Top 10 Similar users with userId: 72------")
         get_recommendations_userwise(72)

         -----Top 10 Similar users with userId: 72------

Out[76]: [72, 8375, 7721, 8123, 88, 7733, 4896, 2814, 98, 257, 6389]
```

User Story 8:

#Find alumni based on a particular skill
MATCH (a:Alumni)-[r:SKILLED_IN]->(b:Skills{skillName:"Neo4j"}) return a,b,r
#Find alumni skills and the company they have been employed at
MATCH (a:Alumni)-[r:SKILLED_IN]->(b:Skills) MATCH (a1:Alumni)-[r1:WORKING_IN]->(b1:Company) return a1,b,b1,r,r1
#Find the companies alumni have got internship
MATCH (a1:Alumni)-[r1:INTERNED_AT]->(b1:Company) return a1,r1,b1
#Find the companies where alumni have interned and worked in the same company

```
MATCH (a:Alumni)-[r:WORKED_AT]->(b:Company) MATCH (a1:Alumni)-[r1:INTERNED_AT]-
>(b1:Company) WHERE a.alumniName = a1.alumniName return a,r,r1,b
#If student wants to know alumni who have a same skills as him
MATCH (a:Alumni)-[r:SKILLED_IN]->(b:Skills) MATCH (a1:Student)-[r1:SKILLED_IN]->(b1:Skills) where
b.skillName = b1.skillName return a,b1, a1, r, r1
#Find out the companies where alumni have worked and are working
MATCH (a:Alumni)-[r:WORKING_IN]->(b:Company) MATCH (a1:Alumni)-[r1:WORKED_AT]-
>(b1:Company) where b.companyName = b1.companyName return a, b, r1, r
```

8. Evaluation

The team has strived hard to ensure that all the available use cases were implemented and worked up to the expectation of the user stories. Sometimes expectations don't meet reality and some of the user stories haven't come out as expected.

Successful:

User Story 2 -Successful implementation of the data model is done with the attributes Course, student, modules, assignments, submission. The goal of the user story is satisfied.

User Story 3- The goal to add/view the status of submission and set/update assignment reminders were successful as the students could do it. The student can add and check the status of submission and also, he/she can check if it has been evaluated or not and find the grades.

User Story 4 -The lecturers/professors can view their modules, know the students enrolled for the module and grade/view their submissions. This also helps them to look out for the defaulters and contact them asking the reason for failing the submissions via email or any other mode of communication.

User Story 6 - As per the project scope, this story is correctly modeled. All the expectations are fulfilled. However, there is a scope of enhancement for selecting the content/skill experts. This can be handled by logic and then stored the result back in the data model. The result will be used in a query for selection of content/skill experts.

There is a possibility to use Redis for chat session as we can push the BLOB with a Long key into Redis very fast and also read out of Redis just as fast.

Partially completed/Unsuccessful:

User Story 3 - Once, the timestamp of lastDateOfSubmission is crossed, an automated scheduled event could update the reminderStatus as inactive and the student would stop receiving reminder alerts for an assignment whose submission date has passed. (10)

User Story 5 - This user story satisfies the said goal but could have some enhancements to strongly achieve the motive. Text/Sentiment analysis (Natural language processing): The purpose this user story is to find the proficiency indicator for a particular module. For this purpose, we had considered two parameter ratings and remarks by the professor. The rating is in numerical format and remarks in the text. Therefore, an algorithm can be used to combine both the parameter to give a single indicator. This is not accomplished in this user story. (11)

User Story 7 - Recommendation for the job based on the user profile is not fulfilling all the requirements. Searching similar job profile and similar user are implemented using Cosine Similarity algorithm in its primitive form. (12)

Future Scope:

Text summarization can be an enhancement which may be implemented in the future. Text summarization is distilling the important information from the source. The part that can be concentrated here for text summarization can be Statement of Purpose, extract only the required keywords and it can help in better analysis. Since the number of applicants is generally high, summarization makes the selection process easier. The most important part of summarization will be reduced reading time and effectiveness of hitting the required key points from the statement of purpose. (13)

9. Bibliography

1. [Online] https://wpcurve.com/trello-for-project-management/.

2. [Online] https://neo4j.com/docs/operations-manual/current/installation/neo4j-desktop/.

3. [Online] https://docs.mongodb.com/manual/mongo/.

4. [Online] https://scalegrid.io/blog/how-to-connect-your-mongodb-deployments-to-robo-3t-gui-at-scalegrid/.

5. [Online] https://www.dbschema.com/mongodb-tool.html.

6. [Online] https://neo4j.com/developer/graph-database/.

7. [Online] https://docs.mongodb.com/manual/core/data-modeling-introduction/.

8. Kaggle. [Online] https://www.kaggle.com/c/job-recommendation/data.

9. [Online] https://github.com/Agarka/Job-Recommendation-Engine/blob/master/Job_recommendation_engine.ipynb.

10. [Online] https://thecodebarbarian.com/node.js-task-scheduling-with-agenda-and-mongodb.

11. [Online] https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b.

12. [Online] https://docs.google.com/presentation/d/1WVKsxtuPchFBn5foV0bCIa-f6DKy-i_-7MDue69OedU/pub?start=false&loop=false&delayms=3000&slide=id.g1ac72648e1_7_89.

13. [Online] https://medium.com/jatana/unsupervised-text-summarization-using-sentence-embeddings-adb15ce83db1.

14. [Online] http://learnjava4enterprise.blogspot.com/2016/12/embeeded-vs-reference-approach-in.html.