Senior Mentor: College Networking Platform

A project report submitted in partial fulfillment of the requirements for the degree of

Bachelor of Engineering

By

Ayush Tomar (Roll No. 8645)

Ayush Yadav (Roll No. 8648)

Sudheer Tripathi (Roll No. 8646)

Parth Shah (Roll No. 8638)

Under the guidance of

Prof. Supriya Kamoji

DEPARTMENT OF COMPUTER ENGINEERING

Fr. Conceicao Rodrigues College of Engineering, Bandra (W), Mumbai - 400050

University of Mumbai

April 30, 2022

We dedicate this work to our family and friends.
We are grateful for having their constant motivation and support.

Internal Approval Sheet

CERTIFICATE

This is to certify that the project entitled **"Senior Mentor:College Networking Platform"** is a bonafide work of **Ayush Tomar(8645) , Ayush Yadav(8648), Sudheer Tripathi(8646) , Parth Shah(8638)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Bachelor Of Engineering** in **Computers.**

Prof. Supriya Kamoji

Supervisor/Guide

Dr. B.S.Daga

Head of Department

Dr.Srija Unnikrishnan

Principal

Approval Sheet

Project Report Approval

This project report entitled by **Analysis Of Growth And Planning Of Urbanization And Correlated Changes In Natural Resources** by **Edwin Clement, Joshua Noronha** is approved for the degree of Bachelor of Engineering

Examiners

1.———————————————

2.———————————————

Date:

Place:

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Ayush Tomar(Roll No. 8645) (sign) _____
Ayush Yadav(Roll No. 8648) (sign) _____
Sudheer Tripathi(Roll No. 8646) (sign) _____
Parth Shah(Roll No. 8638) (sign) _____

Date: April 30, 2022

Abstract

Senior mentor is a very efficient and reliable web app, where seniors and juniors of the college can easily approach each other. At all levels of education, students must be able to communicate effectively. Without a well-developed communication platform, children run the risk of falling behind their peers or becoming emotionally overwhelmed or withdrawn at school. Effective communication skills are essential for the successful future career of a student.

In today's competitive and continuously changing world, effective communication skills plays a major role in matching with the pace of development. Good reading, writing and keen listening are the three most important aspects of communication skills for students.

Effective communication skills help students to listen, understand the point of view of teachers in the class. After listening and understanding what teachers are speaking about, students can ask better questions with confidence and it will help them to gain more knowledge.

This web app provides an outstanding platform specially in this covid scenario to communicate with their peers while being at home with great ease. By this app, We aim to eliminate the problems covid has created and make sure it doesn’t affect our academic background.

Acknowledgments

We have great pleasure in presenting the report on **"Senior Mentor"**. I take this opportunity to express my sincere thanks towards the guide Prof Supriya Kamoji, C.R.C.E, Bandra (W), Mumbai, for providing the technical guidelines, and the suggestions regarding the line of this work. We enjoyed discussing the work progress with him during our visits to the department.

We thank Prof. Dr. B.S.Daga, Head of Computer Engineering Dept., Principal and the management of C.R.C.E., Mumbai for encouragement and providing necessary infrastructure for pursuing the project.

We also thank all non-teaching staff for their valuable support, to complete our project.

Ayush Tomar(Roll No. 8645)
Ayush Yadav(Roll No. 8648)
Sudheer Tripathi(Roll No. 8646)
Parth Shah(Roll No. 8638)

Date: April 30, 2022

Table Of Contents

Chapter 1 : Introduction 11

1.1: Senior Mentor, the idea 11

1.2: Objectives of the project 12

Chapter 2 : Review Of Literature 13

2.1: Background of social media applications 13

2.2: How Senior Mentor is different 13

Chapter 3 : Proposed System 14

3.1: Challenges in old web application architectures
14

3.2: Modern application requirements and Frontend Frameworks. 14

3.3: MERN stack and MVC architecture 15

3.4: Proposed Flow diagram 15

3.5: Proposed database architecture 18

Chapter 4 : Software / Hardware requirements 19

4.1: Software requirements (Server, Client and Database)
19

4.2: Why we used React on client side 21

4.3: Why we used Node on server side 24

4.4:Why we used a Non Relational Database 25

4.5: Why we used an ODM 26

4.6: Why we used SocketIO library 27

4.7: Why we used Material UI 29

4.7: Why we used SupaBase 29

4.8: Why we used Redis Queue 29

Chapter 5: Implementation 39

5.1: Implementation Output 39

Chapter 6: Conclusion and Future Work 43

6.1: Application Output 43

6.2: Challenges we faced and learnings. 43

6.3: Future work 44

Chapter 7: Reference 45

List of Figures

1) Figure 1:Architecture od senior mentor 15
2) Figure 2. Use Case Diagram 16
3) Figure 3. Flow Diagram of Senior Mentor 17
4) Figure 4. E.R Diagram of Senior Mentor 18
5) Figure 5.Virtual ODM 22
6) Figure 6. Engine IO workflow 28
7) Figure 7. Chat workflow 29
8) Figure.8 Home Page 31
9) Figure.9 View a post 33
10) Figure 10. Comment on post 34
11) Figure 11. Message a User 35
12) Figure 12. Type a Message 35
13) Figure 13. View Profiles of the user 36
14) Figure 14. Filter Profiles 37
15) Figure 15. Filter Profiles 2 37
16) Figure 16. All Chat 38
17) Figure 17. New Message 39
18) Figure 18. View Profile 39
19) Figure 19. Edit Profile 40
20) Figure 20: Edit Skills 41

Chapter 1 - Introduction

The year 2020-21 has taught us lessons and has changed the entire execution process of the modern education system, although bits and pieces of our education system were already digital, the COVID-19 pandemic has pushed all of us to accelerate the offline to online shifting process. The education system has changed dramatically and e-learning has scaled up to a whole new level.With this in mind we thought of designing a system where juniors could interact with seniors, we have come up with a web application that allows students to connect with each other in every possible way, introducing Senior Mentor.

Senior Mentor is a web application loaded with all features of modern web applications, it is based on Javascript tech stack which makes it quick and responsive.

Senior Mentor is a very efficient and reliable web app, where seniors and juniors of the college can easily approach each other. At all levels of education, students must be able to communicate effectively. Without a well-developed communication platform, children run the risk of falling behind their peers or becoming emotionally overwhelmed or withdrawn at school. Effective communication skills are essential for the successful future career of a student.

In today's competitive and continuously changing world, effective communication skills play a major role in matching with the pace of development. Good reading, writing and keen listening are the three most important aspects of communication skills for students.

Effective communication skills help students to listen, understand the point of view of teachers in the class. After listening and understanding what teachers are speaking about, students can ask better questions with confidence and it will help them to gain more knowledge.

This web app provides an outstanding platform specially in this covid scenario to communicate with their peers while being at home with great ease. By this app, We aim to eliminate the problems covid has created and make sure it doesn't affect our academic background.

The objectives of our project are-

- To create a stress free and competitive environment at the same time among students, where they help each other by not only sharing their knowledge but also competing among us.
- We aim to strengthen the bonds between seniors and juniors of any particular institution, so that juniors have a beautiful opportunity to learn from their direct superiors and seniors get a sense of responsibility and motivation to work harder.
- Moreover, we have tried to integrate an event calendar as well as to keep students updated of various important dates like hackathons, exams and other competitions.
- Effective discovery tabs can aid students to interact and collaborate with people of the same skill set and interest. Moreover the discover tab might be very useful for the user to stay updated about the area of his/her interest.

Chapter 2 - Review of Literature

The current platforms made for networking for students lack appropriateness as not every student can follow the same advice posted by a student from a different college or domain. The very fact that each student needs to face a problem unique to their college and exposure says a lot about how there is no application to fill this void. Applications like LinkedIn, and Meetup.com are for professional purposes but being bombarded with advice from students not facing the same problem as you is a problem and our product aims to tackle this.

Senior Mentor is exquisite to a college where students can get advice from seniors who have been in their shoes and will also provide a more germane approach to the problems the student is facing. Often we need to have some connection with a college senior where we have to stay in contact with them so that they can help the student in the future. The problem arises when someone doesn't know where to seek advice from and also isn't able to make proper contacts to get the help they need.

Our product will help students to interact with seniors and collaborate with peers as we completely remove the effort one has to take to make contacts and you can directly contact any person you wish to seek guidance from. The chatting feature allows the users to interact without having their personal contact numbers and hence is convenient to both the entities.The profile of every student is available because of which a student can decide if he will get the direction he is looking for from a particular mentor. All these features and many more make this product much different from what is present in the market right now and makes this product a game changer which gives more information to the students.

The recommendation system we are working on will also greatly help students to discover people and content which aling to their interest and skills. This can play a great role to form a connection between people not only help educate each other with new techniques and methods but also form collaborations for different projects, start up ideas, etc.

Chapter 3 - Proposed System

Web applications today run on a combination of various frameworks to be highly responsive, Senior Mentor must have features such as live chat, group chat, posts, events, notifications, etc. Hence, if we continue to implement features with the old html-css-js way probably using jQuery for Frontend and using a template engine in backend, it would lead to a lot of spaghetti code, also using some template engine would lead to a lot of boilerplate code, we need modularization of code in Frontend, especially for the chat feature to be highly reactive, we need a frontend framework. The frontend framework we have used is ReactJs.

Modern Web Applications Architecture that is built on client side MVC architecture and utilizing server side RESTful web services to provide desktop-like rich browser interfaces and leads to reusability, simplicity, extensibility, and clear separation of component responsibilities compared to traditional web application architecture.

As the application has expanded it has led to introduction of a lot of new APIs which involve analytics, keeping separate servers for separate responsibilities has become essential. Hence, we have adopted microservices architecture. We would have separate microservices for saving contextual data and saving graphical data. The base server would serve data that does not require analytics and the analytics server would serve the data that involves discovery / analytics / recommendation features of the platform.

Below is the **high level design** of the system**,**

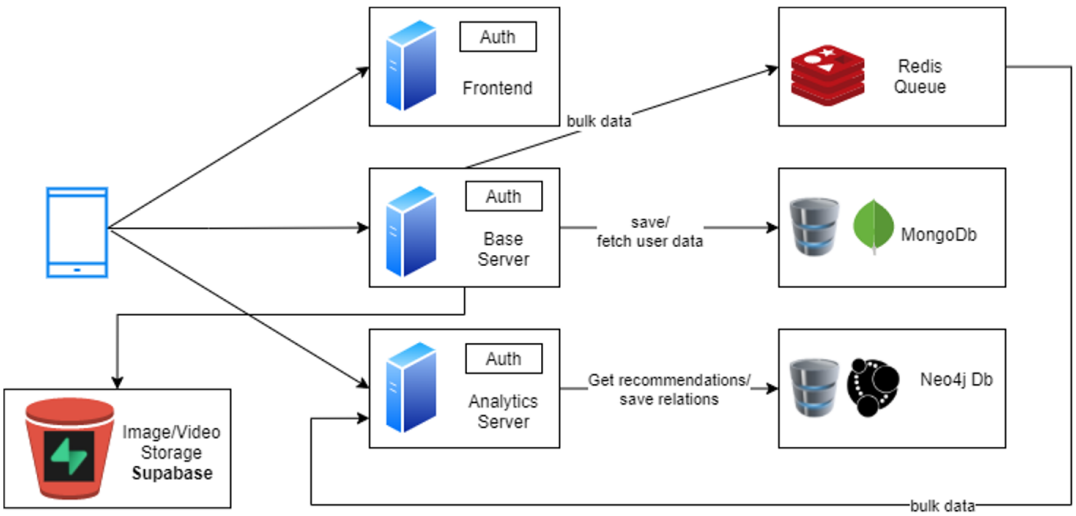

Figure 1: High Level Design of Senior Mentor

The application is based on microservices architecture and the software architecture can be divided into 3 layers.

Presentation Layer

- First is the Frontend server that serves static contents on the website. This makes the client facing side of the system decoupled from the backend of the system.

Business Logic Layer

- The core APIs reside in the Base server, which takes care of all the APIs which are related to Student i.e Login, Registration, Profile Page, Notifications, etc. It also has APIs related to Posts by Users. The Base server persists all the contextual data of the user in MongoDb and is the only service connected to MongoDb.
- The Analytics server consists of the APIs which are responsible for sending personalised data to the user. It includes personalized Events discovery, Posts discovery as well as intra college peer discovery feature. This microservice does the work of running these queries on Neo4j

Data Persistence Layer

The application primarily stores the data in two different type of databases.

- Storage of contextual data of the user is done in MongoDb. This includes a User profile for example.
- Storage of graphical data is done in Neo4j. From the data posted by the user, the platform infers whether a particular user is interested in a particular type of tag. If the user is interested in a tag, there's a direct connection between the user ID and the tag.
- Storage of photos and videos is done on a free open source storage service called Supabase storage.

The users will be of two types i.e registered and guests, the system would serve some data to the guests users too. The registered users would be mainly college admins and students. College admins would majorly post events related to the college. There would be a super admin to take care of verifying the college admins. The system would take care of recommending the events posted by college admins to users. The event recommendations would be personalised based on the user data previously known.

Since we know the users of the system and the basic working, let's look at the use case diagram below.

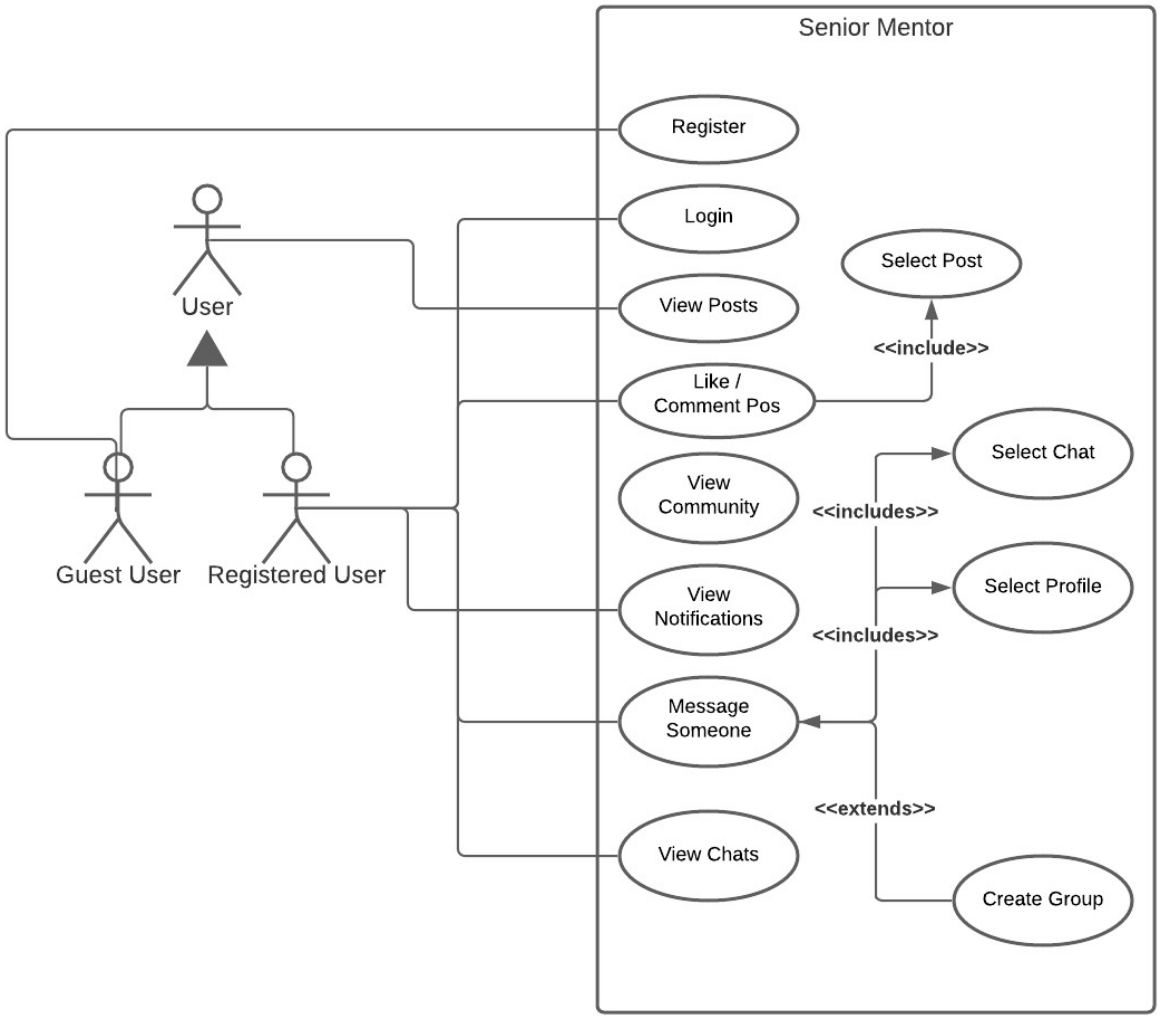

Figure 2. Use Case Diagram

When the user visits the site for the first time, he/she is considered as guest user, next once the user is registered, the user gets to know the hidden part of the platform, i.e Community and Discovery tabs.
Below is the basic flow diagram of the system.

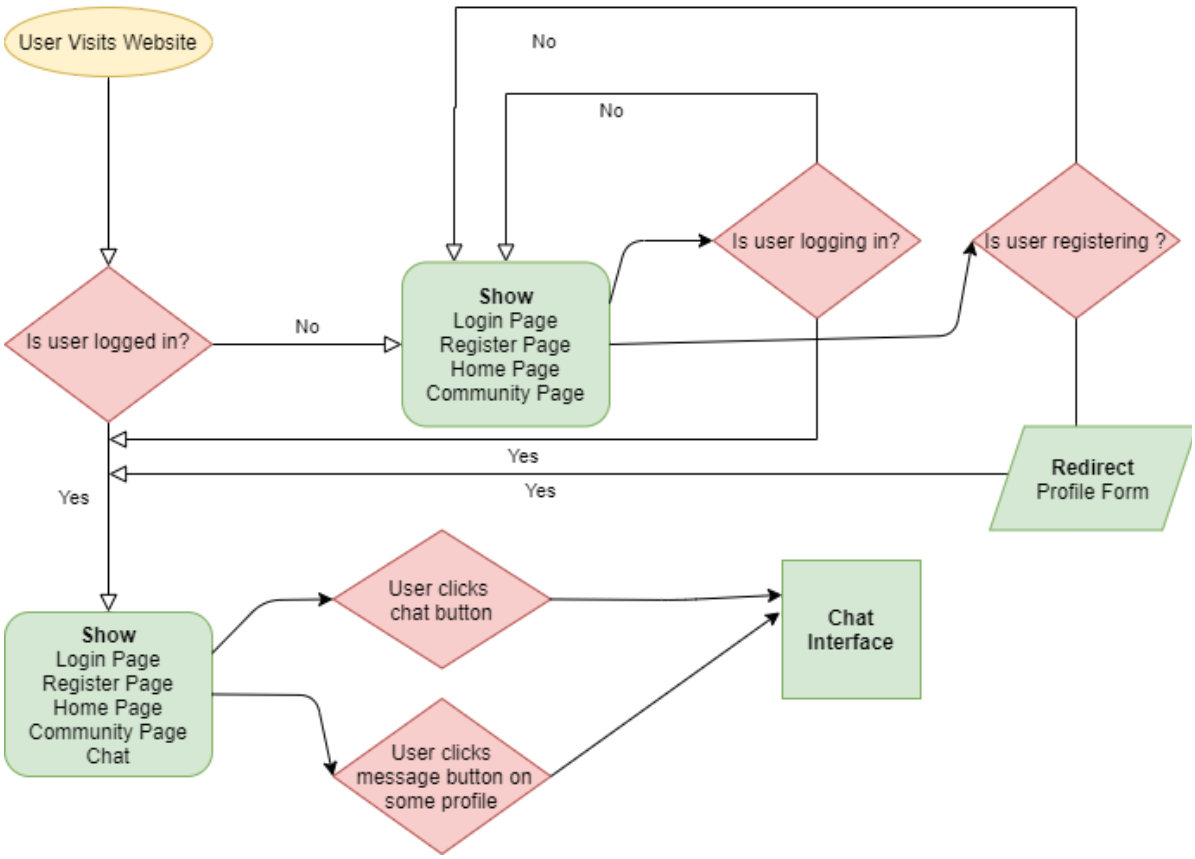

Figure 3. Flow Diagram of Senior Mentor

The database we have used is a non relational database, but so far the database design has been very similar to a relational database. We have implemented the models in MongoDb but the database architecture is close to being relational so far.

Below is what it would look like if it were a relational database,

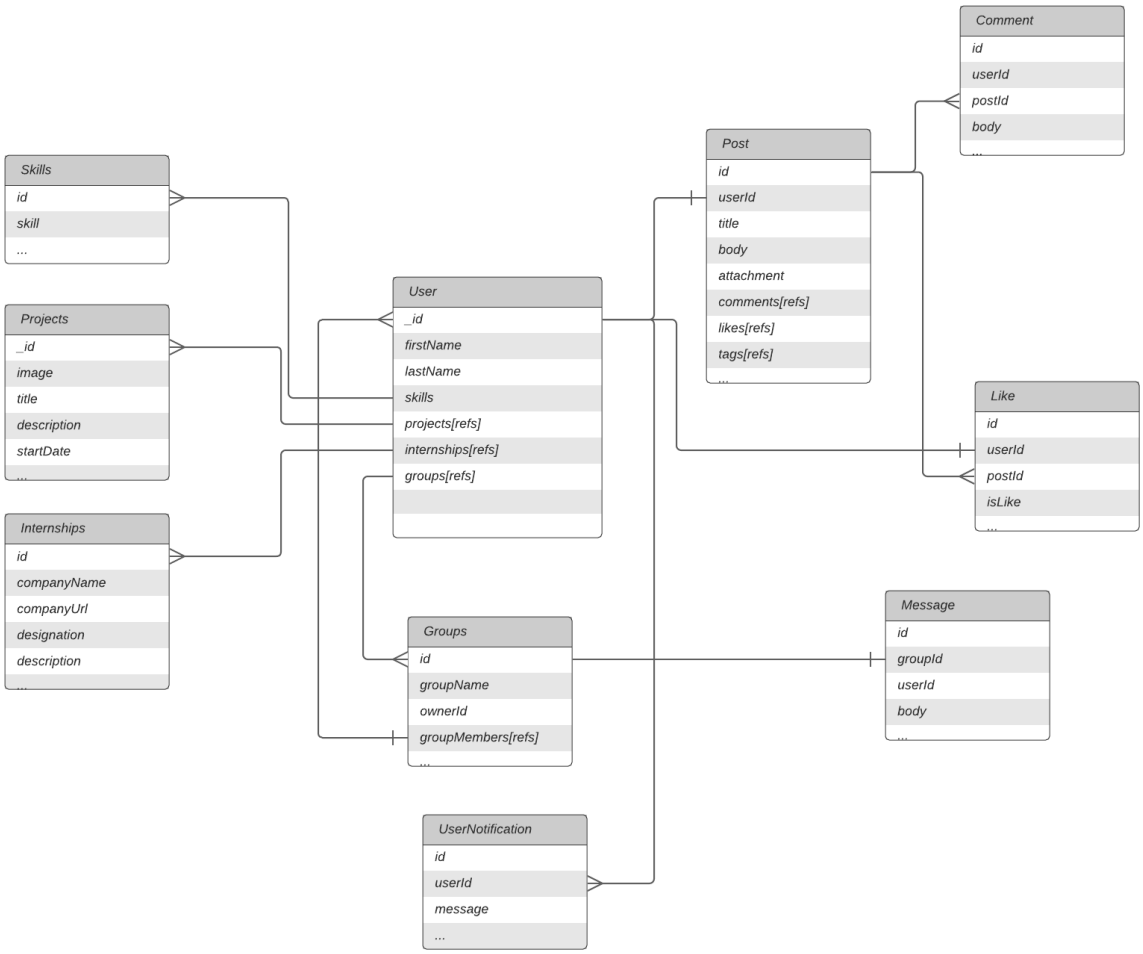

Figure 4. E.R Diagram of Senior Mentor

Chapter 4 - Hardware/Software requirements.

Software requirements - **Frontend**

- ReactJS
- Semantic UI
- React Router DOM - for managing states
- SocketIO client
- Material UI

Software requirements - **Server**

- ExpressJS
- SocketIO server
- Material UI
- JWT encoders and decoders.
- Analytics Server - Neo4j

Software requirements - **Database and Models**

- Object Document Mapper - Mongoose
- Non relational databases
    - MongoDb.
    - Neo4j
- Image/ Video Storage - SupaBase
- Redis Queue

Software requirements - **Database GUI**

- Robo 3T.

We used a frontend framework, ReactJS, why ?

Why did we need a Frontend framework ?

Front-end frameworks are a powerful tool for developing complex user interfaces. They encourage you to build out a maintainable, modular, standalone architecture that makes it easy to build your application and collaborate with other developers. Popular frameworks are backed by supportive communities, a wealth of documentation and tutorials, and offer battle-tested code that tackles common challenges that front ends pose as they scale. Frameworks allow you to tap into the most modern JavaScript features and offer tooling that makes it easy to prototype apps. Finally, they empower you with a shared language to discuss your architecture and challenges.Front-end frameworks and libraries come in many shapes and sizes—you can use a full-fledged UI framework to build out your entire front end, implement a CSS library to tighten your visual design, or use a templating engine to create reusable components.

Front-end frameworks exist because for many apps, the front end grows and strains in predictable ways. While each popular framework offers its own design philosophy, they are all attempting to solve the same general problems we encountered earlier:

- Your code should be maintainable: easy for you and others to read, test, and change.
- Complex interfaces are usually made of the same components. You should be able to create and reuse these components to easily create new pages.
- Since DOM manipulations are slow, you want to perform as few element updates as possible.
- You should be easily able to manipulate your UI based on data.
- Your UI should be consistent and intuitive; this means standardizing typography, color, buttons, inputs, and other elements.

- You shouldn’t need to reinvent the wheel and write tons of boilerplate when it comes to solving these common front-end challenges.
- You should have a common language through which to communicate your ideas and patterns with other developers.

You can build a simple frontend with just three files: HTML, CSS, and JavaScript. However, as your app scales, your files will grow along with it, filling up with inscrutable and unmaintainable spaghetti code.

Why did we use ReactJS ?

React uses a virtual DOM for client side rendering. This helps us quickly render different views instead of fetching data from server again and again. [1]

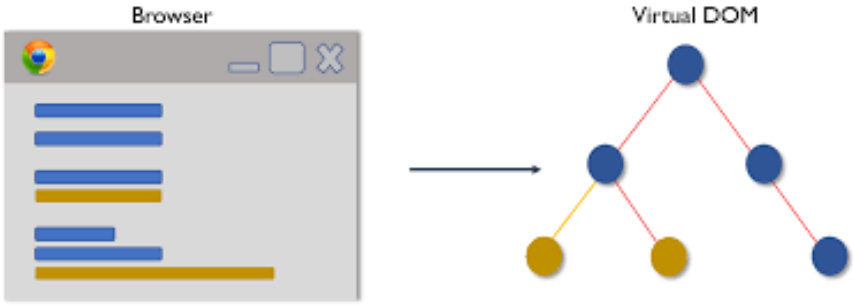

Figure 5.Virtual ODM

Traditionally, web application UIs are built using templates or HTML directives. These templates dictate the full set of abstractions that you are allowed to use to build your UI.

React approaches building user interfaces differently by breaking them into components. This means React uses a real, full featured programming language to render views, which we see as an advantage over templates for a few reasons:

- JavaScript is a flexible, powerful programming language with the ability to build abstractions. This is incredibly important in large applications.
- By unifying your markup with its corresponding view logic, React can actually make views easier to extend and maintain.
- By baking an understanding of markup and content into JavaScript, there's no manual string concatenation and therefore less surface area for XSS vulnerabilities.

React has JSX, an optional syntax extension, in case you prefer the readability of HTML to raw JavaScript.

React really shines when your data changes over time.

In a traditional JavaScript application, you need to look at what data changed and imperatively make changes to the DOM to keep it up-to-date. Even AngularJS, which provides a declarative interface via directives and data binding requires a linking function to manually update DOM nodes.

React takes a different approach.

When your component is first initialized, the render method is called, generating a lightweight representation of your view. From that representation, a string of markup is produced, and injected into the document. When your data changes, the render method is called again in order to perform updates as efficiently as possible.

We used mongoose ODM, why ?

Why use Node.js for backend?

JavaScript was initially created to “make web pages alive”.The programs in this language are called scripts. They can be written right in a web page’s HTML and run automatically as the page loads.Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.[3]

Node.js is a runtime environment that allows software developers to launch both the frontend and backend of web apps using JavaScript. Although JS underpins all the processes for app assembly, as a backend development environment, Node.js, differs from the frontend environment. It has unique APIs that support HTTP requests, file systems and other server-side features for which frontend APIs provide limited support. The following are the key points which make node js the best choice for backendAn I/O non-blocking model allows for serving multiple simultaneous requests, ensuring easy scaling and prompt execution of client requests on high-load platforms.

- An I/O non-blocking model allows for serving multiple simultaneous requests, ensuring easy scaling and prompt execution of client requests on high-load platforms.
- With a growing number of online users demanding the most relevant information and services at their fingertips, the Node.js runtime environment’s flexibility and responsiveness are key merits required by modern web apps.
- Node.js language is spoken by both frontend and backend developers, which makes for a more effective and better-coordinated working environment, and eliminates the need to explain how certain features work or what certain code means. Small projects can be handled by a single full-stack team, saving time and money.
- JavaScript programming language is very convenient and straightforward for backend development, inspiring software developers to shift away from more complex and cumbersome options. Node.js ensures that your app will not become obsolete in a few years, and it will be easy for other developers to understand your code.
- The development process is greatly accelerated by code sharing between frontend and backend.

- Node.js is an open-source technology that does not require costly licensing

Why NoSQL database?

Although relational databases have been used for a good plenty of years and have fulfilled the demands of business in the past, things are now changing. With the increasing presence of the internet and usage of social media, the amount of data being generated is quite higher in volume than it was maybe even a few years ago.

Given this staggering increase of data, managing it can be quite a task and relational databases are not quite adept at processing this rapidly. This is due to the fact that the new data coming in does not always fit into the tight schema followed by a relational database. NoSQL database, on the other hand, can easily manage huge volumes of data and the operations performed over it.

For instance, if you have a website that is popular and has at least 10,000 registered customers, and growing daily, each of these customers will follow their own life-cycle and processes. On the front end, they would be loading pages, similar items, adding products to cart etc. but on the backend, whenever an operation is performed, the data is retrieved from the database, the similar items are suggested taking into account the number of times a particular type of query was run, and so on and so forth.

The reason for slow operations could either be slow website loading speed or a slow backend that processes your data. If you have a relational database, chances are there would be innumerable rows and columns, and finding the right match would take a long time. On the other hand, if you use a NoSQL database, this problem would be significantly less.

So is this a real-time example? Amazon uses DynamoDB as mentioned initially, and Google uses BigTable, both examples of a NoSQL database. To put it simply, here are the 4 reasons to switch to a NoSQL database.

- Highly scalable
- Able to handle large volumes of data – structures, and semi-structured
- Schema-less
- Quick iterations

Why do we need an ODM ?

ODMs provide a high-level abstraction upon a **non relational** database that allows a developer to write Javascript code instead of MongoDB to create, read, update and delete data and schemas in their database.[5]
Biggest Pro is that it has the data validation built into it(requirements of what data you will allow to be added or to update your database). It will take some work to build that yourself.

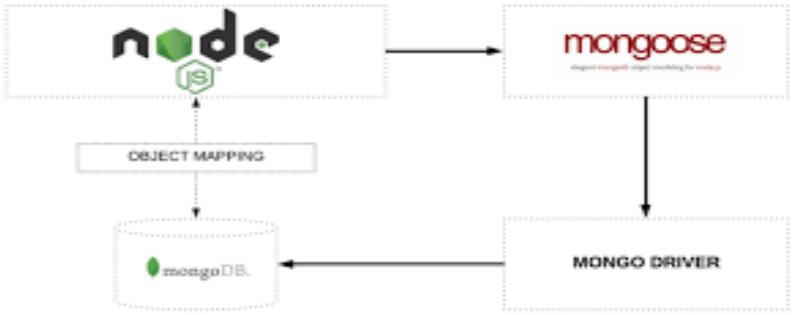

Fig 6 .Database Diagram

It will abstract away most of the mongoDB code from the rest of the application.
So Schemas in the first place helps with that.
Additionally, it implements validation and other neat features in order to make sure your schema is consistent when inserting/updating/finding documents from collections.

It also creates Model abstraction which makes it easier to work with, so it looks like you are working with just objects rather than pure data.
There are many other goodies like middleware, plugins, population, validation.
Mongoose provides optional pre and post save operations for data models. This makes it easy to define hooks and custom functionality on successful reads/writes etc. You can also define custom methods that act on a particular instance (or document). While you can achieve similar functionality with the native MongoDB driver, Mongoose makes it easier to define and organize such methods within your schema definition.
Mongoose makes returning updated documents or query results easier. A prime example can be found with update queries. While the native driver returns an object with a success flag and the number of documents modified, Mongoose returns the updated object itself so you can easily work with the results.

We used SocketIO for a live chat feature, why?

Socket.IO allows bi-directional communication between client and server. Bi-directional communications are enabled when a client as Socket.IO in the browser, and a server has also integrated the Socket.IO package. While data can be sent in a number of forms, JSON is the simplest.To establish the connection, and to exchange data between client and server, Socket.IO uses Engine.IO. This is a lower-level implementation used under the hood. Engine.IO is used for the server implementation and Engine.IO-client is used for the client.

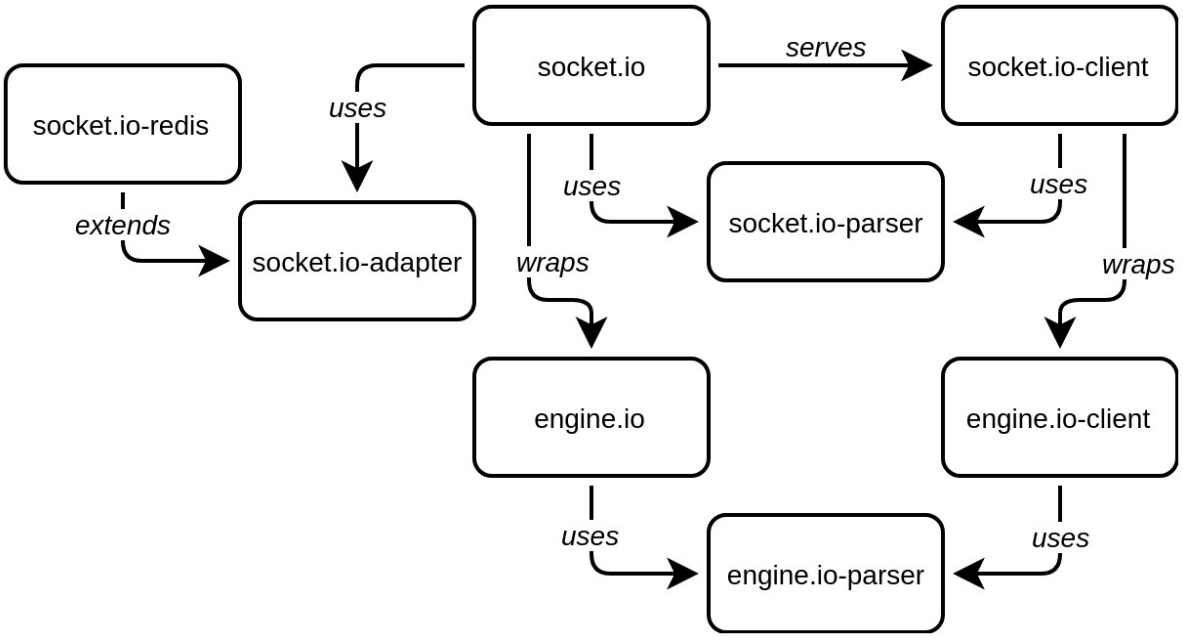

Figure 6. Engine IO workflow

A popular way to demonstrate the two-way communication Socket.IO provides is a basic chat app. With sockets, when the server receives a new message it will send it to the client and notify them, bypassing the need to send requests between client and server. A simple chat application shows how this works.

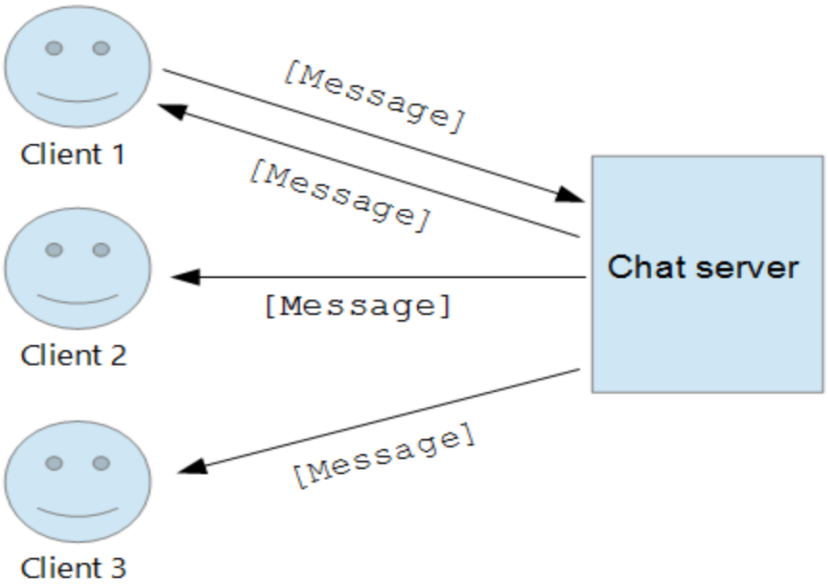

Figure 7. Chat workflow

Why use Material UI?

Material-UI,[2] the React component library based on Google Material Design, allows for faster and easier stylized web development. With basic React framework familiarity, you can build a deliciously material app with Material-UI, and its almost like cheating. *Almost.* This MIT-licensed open source project is more than just parlor tricks though and can get deep quickly. But don't let me scare you! I recently built an app with Material-UI for the first time, and by the end I was delighted. Here are 5 things I appreciated about Material-UI:

1. It's well documented

The official documentation is organized and easily navigable. The library's popularity means you have access to tons of code examples on the web if the documentation is confusing. You can also head over to StackOverflow for technical Q&A from Material-UI devs and the core team.

2. Regular updates

With the recent release of Material-UI v4 (May of 2019) and blog posts with new features and future goals posted monthly, Material-UI doesn't look like its dying down anytime soon. Along with feature updates and improvements, here are the GitHub stats from the latest blog posted November 8th:

We have accepted 182 commits from 68 different contributors. We have changed 1,157 files with 31,312 additions and 9,771 deletions.

3. Consistent appearance

Okay, this is kind of cheating because its a library, so of course the appearance is going to be consistent. BUT Material-UI is a HUGE library, and the benefit is you have some choices.

Aesthetic preferences for Material Design aside, your web project has a high chance of retaining similarity in appearance and functions all throughout.

4. Creative freedom

You don't have to have a consistent appearance if you don't want to!

I know, I know - I just said that it creates a consistent appearance, but that's out of the box. As I hinted at earlier, there is actually quite a lot of depth to the Material-UI components and the developers encourage customization. Material-UI doesn't force Material Design style on you, *it just offers it.*

One delightful component was the ThemeProvider. Placed at the root of your app, you can change the colors, the typography and much more of all sub-Material-UI components! However, this is optional; Material-UI components come with a default theme. Code magic.

5. Components work in isolation

Material-UI components are self-supporting, and will only inject the styles they need to display. They don't rely on any global style-sheets such as normalize.css! You only want to use that super cool progress spinner? Grab it! Live your best life!

So there you have it! Go try it on for size if you haven't already.

There are some great resources for starting projects on the official website.

If you have played around with Material-UI or its part of your regular build, let me know what you liked or what surprised you!

Why use SupaBase?

Supabase is an open source Firebase alternative[15]. This is a bold title, because Firebase is intended as a complete solution, with various features like authentication, file storage, serverless functions, SDK, and much more. Even though Firebase has tons of features, Supabase may be more useful because it uses open source technology. Supabase gives you the flexibility to host on your local machine, in a cloud service provider, or even as a Docker container. This means it’s restriction free, so there’s no vendor locking.

Supabase uses PostgreSQL under the hood for the database and listens to real-time changes through several tools that they build. Currently, Supabase only supports features like databases, authentication, and storage. They also have serverless functions, although these are still in the development stage.

Supabase stands out for the following reasons:

- Supabase handles the scaling for you (even though it uses an SQL database)
- Unlike Firebase, you can perform complex queries or text searches

- Data migration is straightforward in Supabase as it uses PostgreSQL, so you can import data through a .sql file

Why use Redis Queue?

Redis Queue is a python library for queueing jobs for background processing.[16] Since many hosting services will time out on long HTTP requests, it is best to design APIs to close requests as quickly as possible. Redis Queue allows us to do this by pushing tasks to a queue and then to a worker for processing.

Using Redis in conjunction with Redis Queue allows you to request input from the user, return a validation response to the user, and queue up processes in the background. All without the front end user having to wait for those processes to complete. Processes could be anything from Machine Learning models, to duplicating an image to complex simulations.

Anything that takes longer to complete than you would like to have taken place on the front end of your application belongs in the Redis Queue.

Chapter 5 - Implementation.

5.1 Guest Page

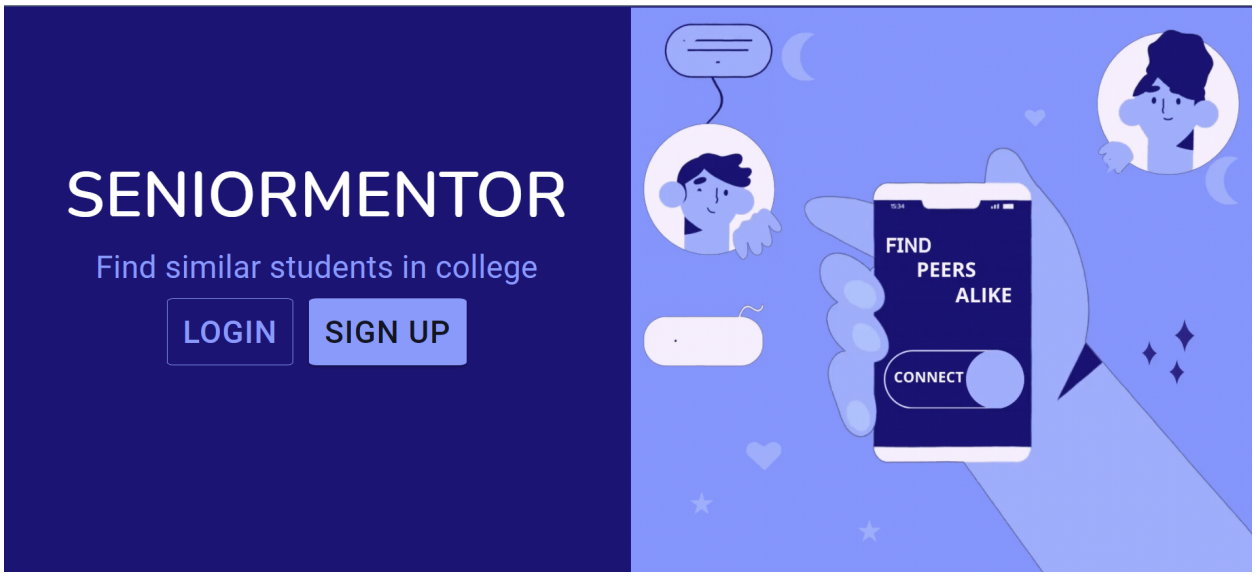

Fig 8 .Screenshot Landing Page

5.2 Sign In Page

Fig 9 .Screenshot Login Page

5.3 Navbar

Fig 10 .Screenshot Navbar

5.4 Post, Likes and Comments

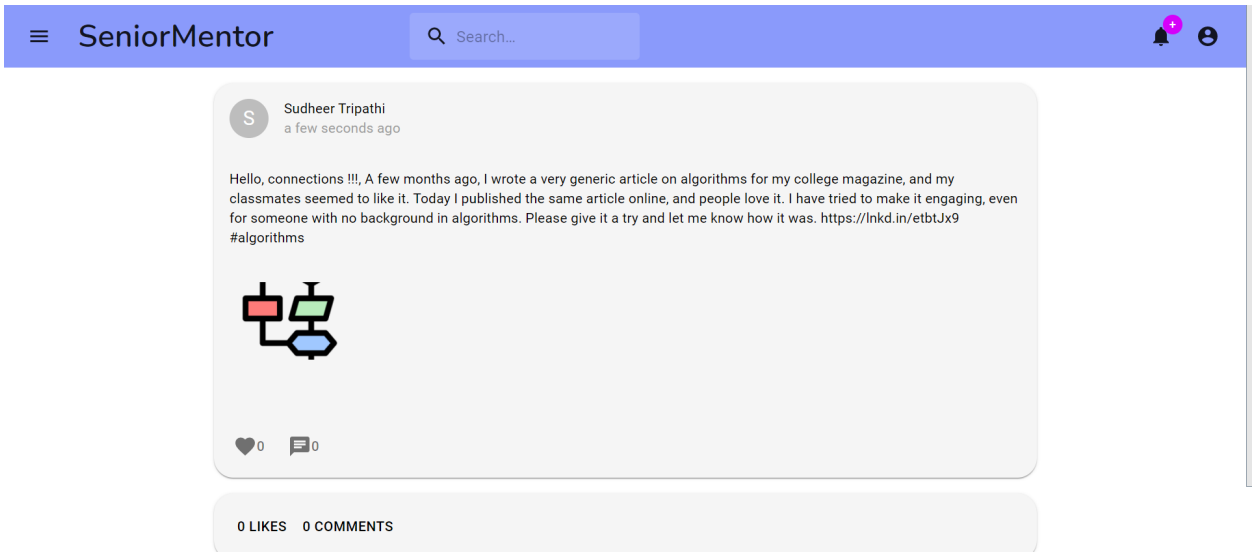

Fig 11 .Screenshot Timeline Page

5.5 Profile and Left Drawer

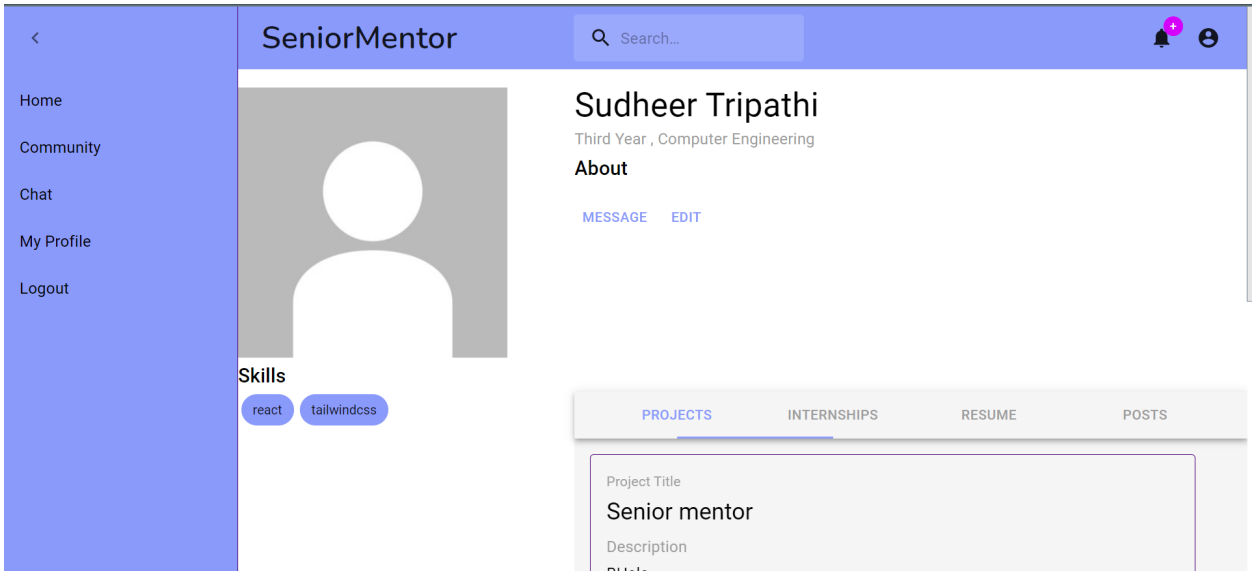

Fig 12 .Screenshot Profile Page

5.6 Chat Interface 1

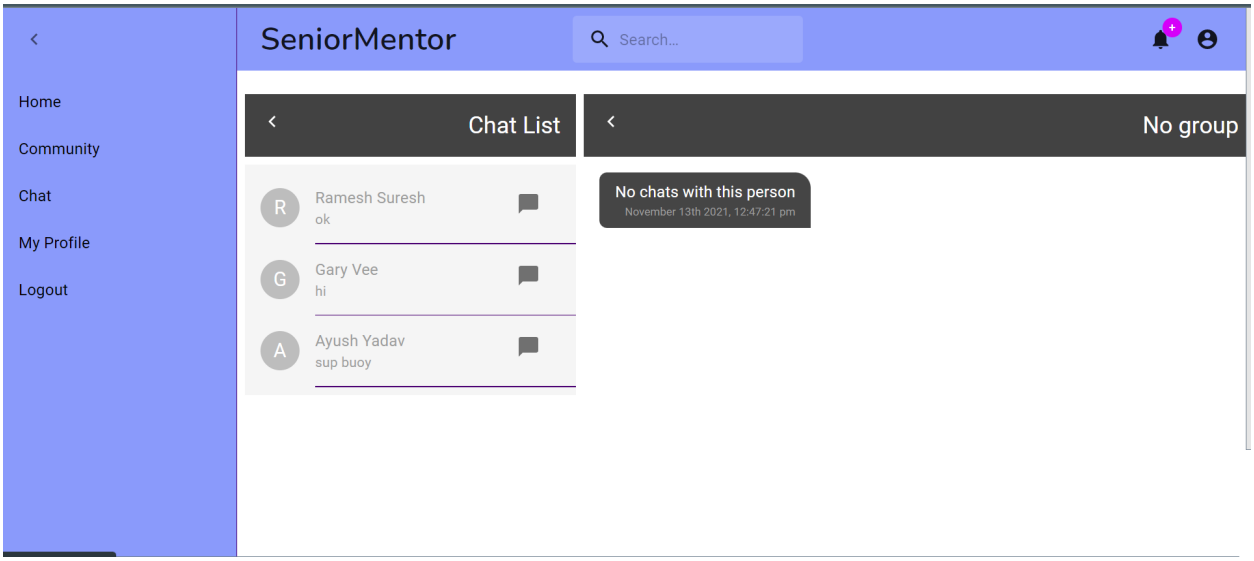

Fig 13 .Screenshot Chat Page

5.6 Chat Interface2

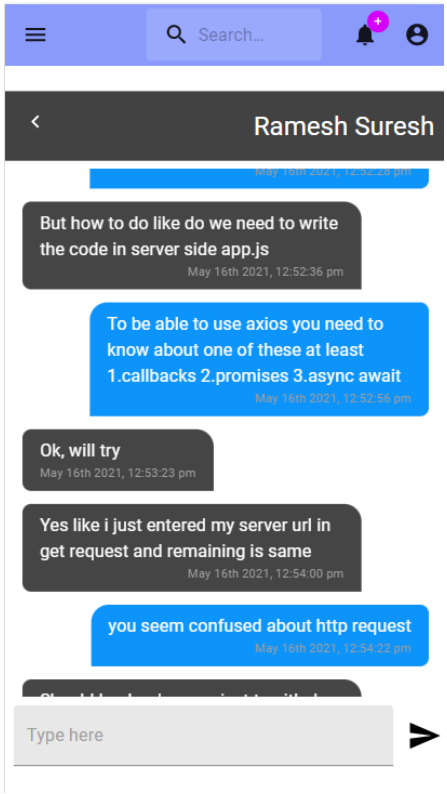

Fig 14 .Screenshot Chat Page 2

5.7 Filter Posts

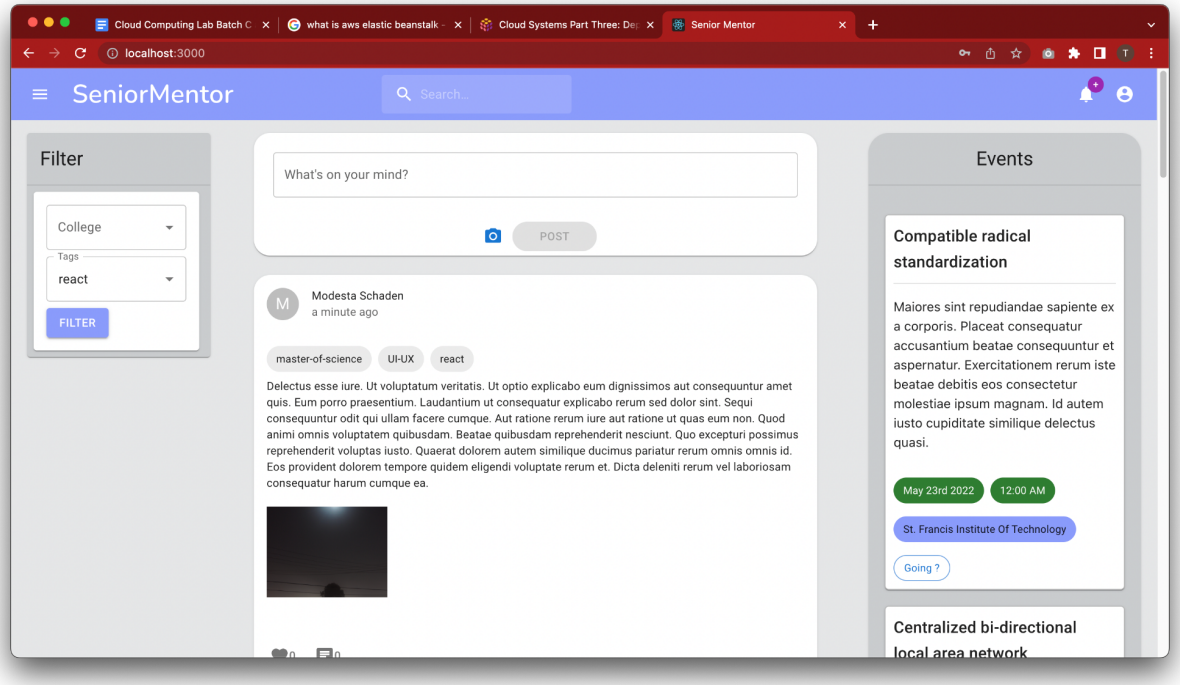

Fig 15 .Screenshot Analytics Page

5.7 Events

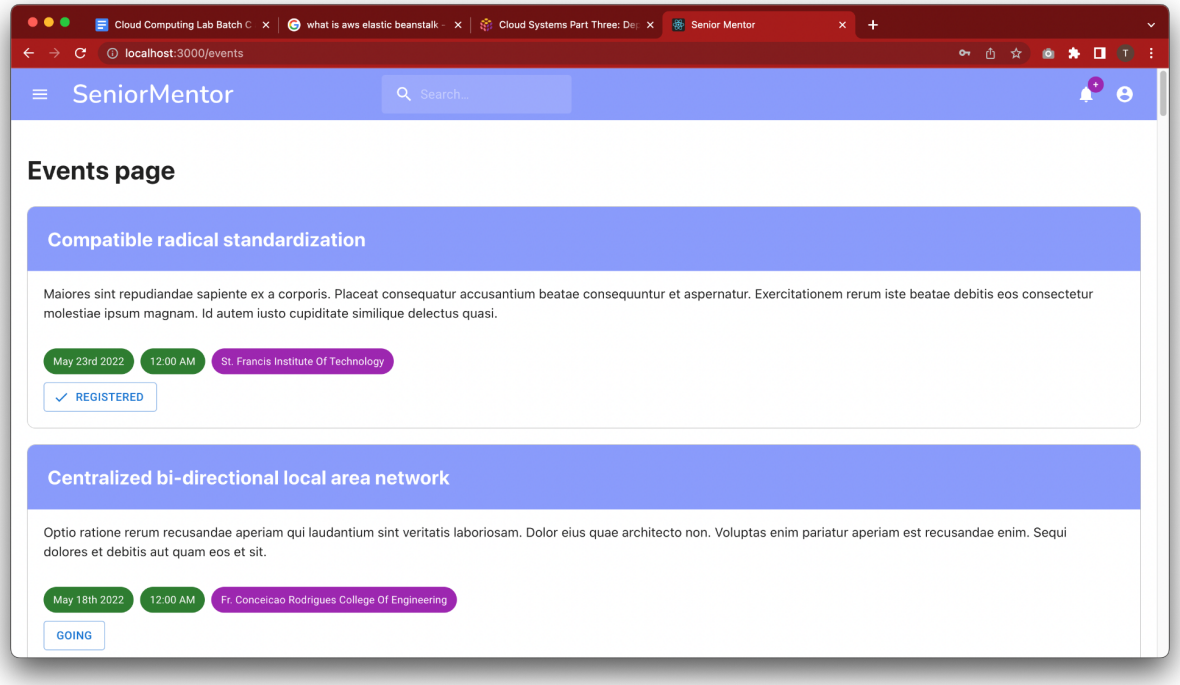

Fig 16 .Screenshot Events Page

5.8 Community

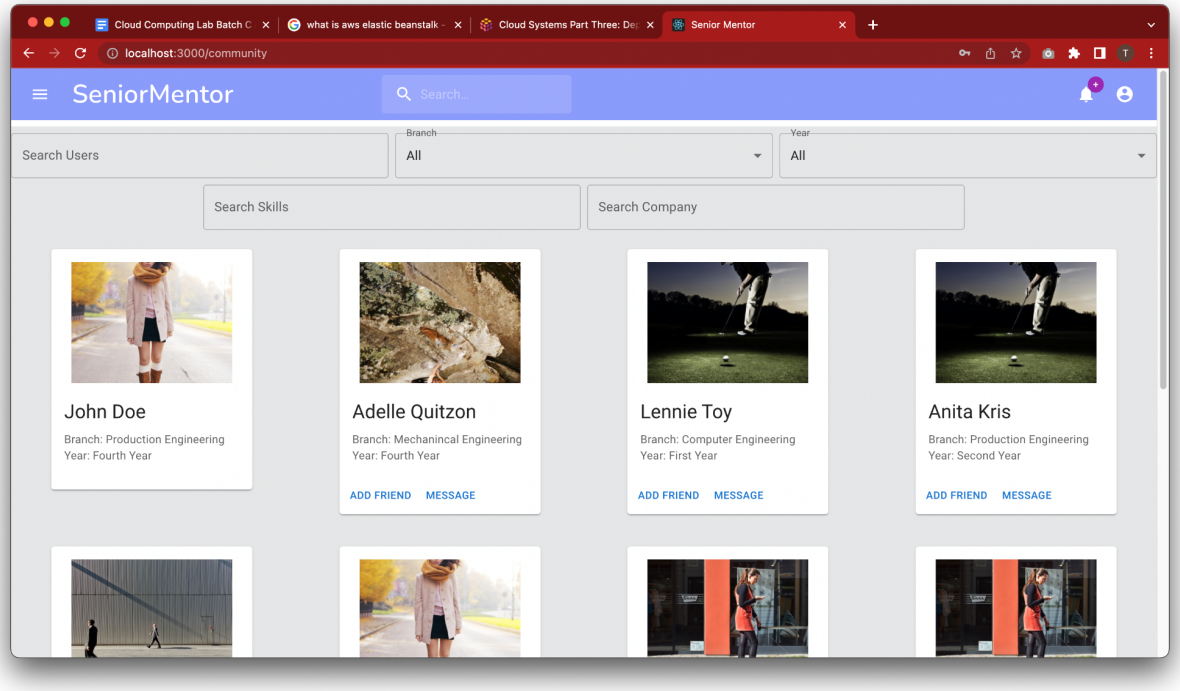

Fig 17 .Screenshot Community Page

Chapter 6 - Conclusion and Future Work.

6.1 Application Output

Currently the application is a device responsive web application. We would also roll out the Android version of the application in Future. Users would also receive relevant updates via email.

6.2 Challenges and Learnings

- UI design of the Frontend was one of the biggest challenges and there have been multiple turning points where we had to change the UI of the application.
- Managing authentication state along with the web chat sockets was the second biggest challenge and we are still facing issues on it and solving the problem.
- Designing the backend of the system was the third biggest challenge, but creating microservices with very specific responsibilities was one of the biggest learnings.
- Deciding what data to store in Neo4j and what to store in MongoDb was another major challenge.
- In case one microservice talks to another, message queues must be used, this is also one of the learnings

6.3 Future Work

Here are the tasks we are aiming to integrate in the platform in the future

- Create an analytics server which takes the data from Frontend and puts it in Neo4j and also extract meaningful analytics data from Neo4j later.
- All the Neo4j queries should reside in Analytics Server.
- Create a Discovery page in the Frontend of the website, which would help users discover events, people, posts.
- Create a robust email system that would send relevant emails to the user.
- Make the notifications on the website real-time.
- Use Neo4j for analytics

Why use Neo4j for analytics?

A graph platform like Neo4j offers an efficient means for data scientists and solutions teams to move through the stages of discovery and design.[17]

A graph platform must also offer a variety of skill-specific tools for business users, solution developers and data scientists alike. Each user group has different needs to visualize connectedness, explore query results and update information.

First, when exploring a concept, teams look for broad patterns and structures best served by global analysis. They need the ability to easily call upon packaged procedures and algorithms.

Organizations want tools to identify communities, bottlenecks, influence points and pathways. In addition, a supported library of algorithms helps ensure that results are consistent by reducing variability introduced by many individual procedures.

In the next phase of solution modeling, a streamlined process becomes extremely important as teams must test a hypothesis and develop prototypes. And the iterative, continuous nature of the above workflow heightens the need for extremely efficient tools with fast feedback loops.

Teams will be using various data sources and tools, so a common, human-friendly way to express connections and leverage popular tools is essential.

Graph Algorithms Are Part of the Neo4j Platform

Neo4j offers a growing, open library of graph algorithms that are optimized for fast results. They are part of the Neo4j platform, which also includes:

- A native graph database
- A connections-first query language
- Analytics integration
- Robust procedures
- Neo4j graph analytics.

Graph algorithms reveal the hidden patterns and structures in your connected data around pathfinding, centrality and community detection (see graphic) with a core set of tested and supported algorithms.

Neo4j graph algorithms are simple to apply so data scientists, solution developers and operational teams can all use the same graph platform. Neo4j graph algorithms are efficient so you analyze billions of relationships and get results in seconds to minutes, or in a few hours for more complicated queries that process large amounts of connected data.

Nodes In Neo4j

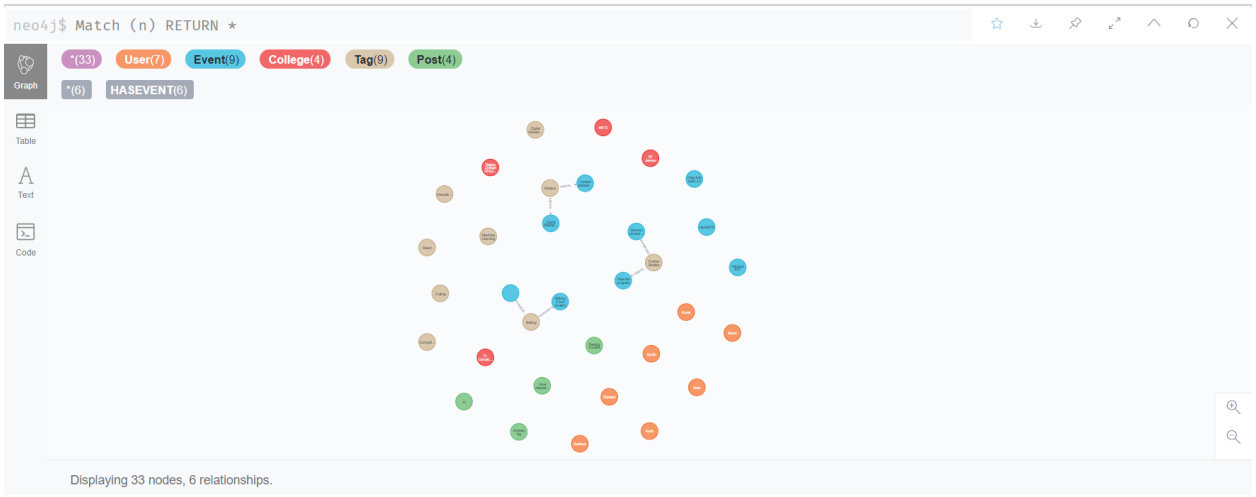

Fig 8 .Graph Diagram 1

Relations In Neo4j

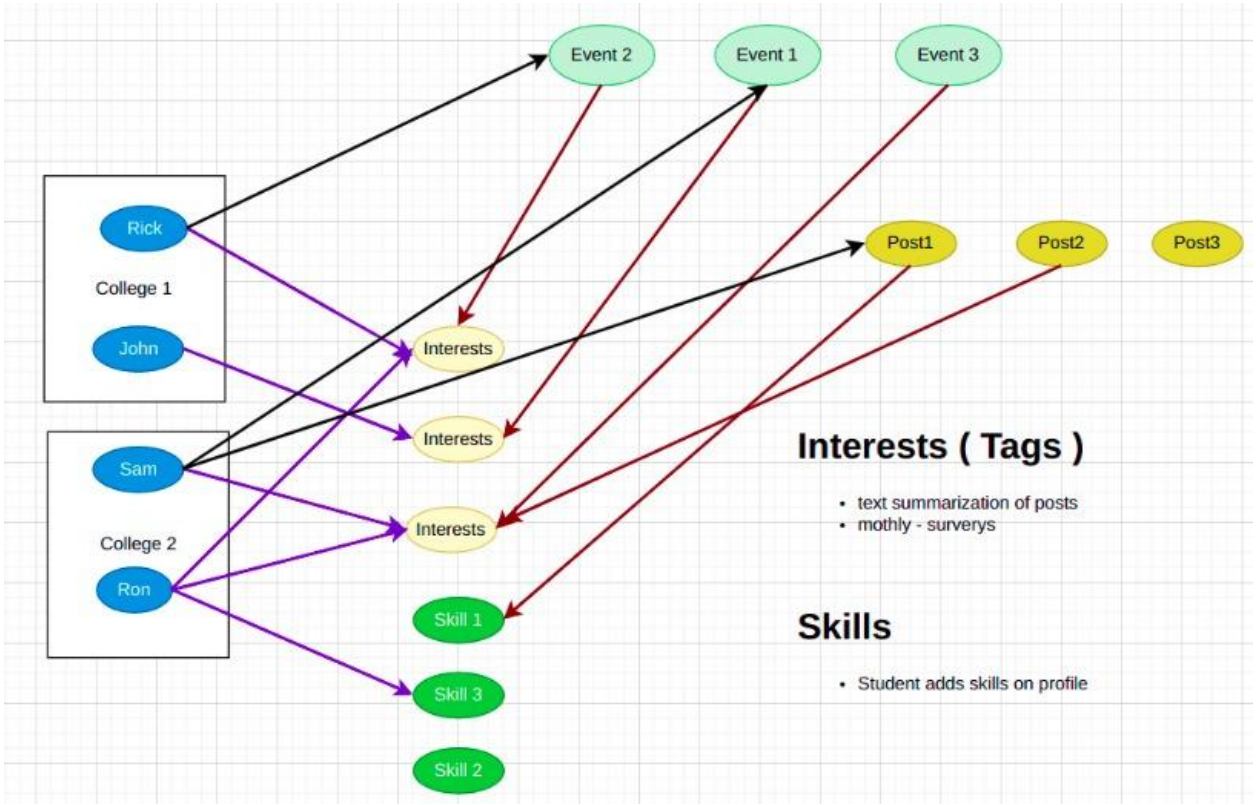

Fig 8 .Graph Diagram 2

Chapter 7 - References

01. https://reactjs.org/
02. https://material-ui.com/
03. https://nodejs.org/en/
04. https://socket.io/
05. https://www.mongodb.com/
06. https://mongoosejs.com/
07. https://reactrouter.com/web/guides/quick-start
08. https://www.npmjs.com/package/jsonwebtoken
09. https://expressjs.com/
10. https://www.linkedin.com/
11. https://twitter.com/home
12. https://www.meetup.com/
13. https://robomongo.org/
14. https://www.npmjs.com/package/socket.io-client
15. https://supabase.io/docs
16. https://redis.com/
17. https://neo4j.com/