NOTCH

A Project Report Submitted

In Partial Fulfilment of the Requirements

For the Degree of

MASTER OF COMPUTER APPLICATION

By

MANIKA GOEL

University Roll No. 1900290149056

Under the Supervision of

Mr. Ankit Verma

Assistant Professor

KIET Group of Institutions

Submitted to

DEPARTMENT OF COMPUTER APPLICATION

Affiliatedto

DR. A. P. J ABDUL KALAM TECHNICAL UNIVERSITY

LUCKNOW

JULY,2021

DECLARATION

I hereby declare that the work presented in this report entitled **“Notch"**, was carried out by US. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution.

I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, We shall be fully responsible and answerable.

MANIKA GOEL

University Roll No.-1900290149056

TRAINING CERTIFICATE

AI | MOBILE APPS | ENTERPRISE SOLUTIONS

Office No. - 727, The Ithum
Tower, Sec-62, Noida
U.P., Postal Code - 201301

Date: 08th July 2021

To Whom It May Concern

This is to certify that Manika Goel a student of KIET Group of Institutions successfully completed her training period from 04th January 2021 to 08th July 2021 with reference to the partial fulfillment of the requirements of the MCA of Dr. APJ Abdul Kalam Technical University.

All the necessary guidance and hands on experience were provided by Zimozi for the establishment of this Training.

We wish her the very best in all her future endeavors.

For Zimozi Solutions Pvt. Ltd.

Priyanka Bijolia
HR Head

+91-120-4158818 | hello@zimozi.co | www.zimozi.co

CERTIFICATE

Certified that **MANIKA GOEL (Univ. Roll No.-1900290149056)** have carried out the project work having **"Notch"** for Master of Computer Application from Dr.A.P.J.Abdul Kalam Technical University (AKTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

Manika Goel (Univ. Roll No -1900290149056)

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Date:

Mr. Ankit Verma
Assistant Professor
Department of Computer Application
KIET Group of Institutions, Ghaziabad

Signature of External Examiner **Signature of Internal Examiner**

Dr. Ajay KumarShrivastava
Head, Department of Computer Application
KIET Group of Institutions, Ghaziabad

ABSTRACT

This project is a mobile application project. This project is all about creating the list based on interest and category. The list contains items that carries the post which may be photos or videos as well as text regarding that category. For example, if we have the category travel and if we added our interest as travel then we will be seeing post that are based on travel and of all the users that are posting the posts related to travel. So the maintenance became very easy.

- Easy accessibility.
- It makes searching records easier and faster.
- The user need not to type in most of the information.
- On the whole it liberates the user from keeping lengthy manual records.
- Everyone wants his/her work to be done by computer automatically and displaying the result for further manipulations.
- So this project is about providing convenience.

ACKNOWLEDGEMENTS

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor **Mr. Ankit Verma**, for his guidance, help and encouragement throughout our project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Ajay Kumar Shrivastava**, Professor and Head, Department of Computer Application, for his insightful comments and administrative help at various occasions.
Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Manika Goel

Univ. Roll No. 1900290149056

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 PROJECT DESCRIPION

The objective of this application is to let that how a normal person know how and which to post and place or to check their interests and list of feeds and videos related to it can use this application easily, it is flexible like data can be deleted enter or updated easily.

“Notch” is project is a mobile application project. This project is all about creating the list based on interest and category. The list contains items that carry the post which may be photos or videos as well as text regarding that category. For example, if we have the category travel and if we added our interest as travel then we will be seeing post that are based on travel and of all the users that are posting the posts related to travel.

We are making this project with the help of NodeJS, database used is MongoDB. Creating this application and some feature we need:

1.1.1 Node.js

Node.js isan open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.

1.1.2 Firebase

Firebase is a platform developed by Google for creating mobile and web applications. It was originally an independent company founded in 2011. In 2014, Google acquired the platform and it is now their flagship offering for app development.

1.1.3 Firebase Token

When a user or device successfully signs in, Firebase creates a corresponding ID token that uniquely identifies them and grants them access to several resources, such as Firebase Realtime Database and Cloud Storage. You can re-use that ID token to identify the user or device on your custom backend server.

1.1.4 Typescript

It is a strict syntactical superset of JavaScript and adds optional static typing to the language. TypeScript is designed for the development of large applications and transcompiles to JavaScript.

1.1.5 VS Code

Visual Studio-Code is a source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

1.1.6 AWS

Amazon Web Services (AWS) is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis.

1.1.7 MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

1.1.8 AWS Lambda

AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services.

1.1.9 Workflow and Approvals

It is a visual design to automate the business processes. The interface provides simple drag and drop options to make this design. It helps in creating a flexible approval process with deal discounts and expense Presidency etc.

1.1.10 Email Integration

Salesforce can integrate to an existing email platform. This helps in providing flexibility to the existing team with no additional learning curve.

1.1.11 Files Sync and Share

This feature provides the sales team the power to easily share various files, discuss them and update them as needed. Also receive alerts when something in the file changes.

1.1.12 Reports and Dashboards

Dashboards offer a real-time picture of the business at a glance. With this, anyone can create detailed reports which can be accessed from anywhere.

1.1.13 Sales Forecasting

This feature helps in getting a real time view of the forecast of a sales team. It provides multi-currency support and an in-line editing mode to manage the sales forecast well.

1.1.14 Territory Presidency

This feature is used to create multiple territory models, preview them before rollout, and continually optimize and balance territories throughout the year.

1.2 PROJECT SCOPE

The following documentation is a project the "Notch". It describe the drawbacks of the old system and how the new proposed system overcomes these shortcomings.

The list contains items that carries the post which may be photos or videos as well as text regarding that category. For example, if we have the cataegory travel and if we added our interest as travel then we will be seeing post that are based on travel and of all the users that are posting the posts related to travel.

It is basically for Institutes with the new trend of managing the stuffs.

- It is time saving as it doesn't involve manual process for facing difficulties due to heavy rush and safe from infectious place
- It is very user friendly.
- User across India come to the portal and register themselves.
- It is eco-friendly as well, as it does not involve usage of papers.
- Errors are almost impossible as it requires less human interaction.
- Accuracy in work.
- Easy & fast retrieval of information.
- Decrease the load of the person involve in existing manual system.
- Access to any information individually.
- Work becomes very speedy.
- Easy to update information

- Easy availability.

1.3 IDENTIFICATION OF NEED

User need identification and analysis are concerned with what user needs rather than what he/she wants. The list contains items that carries the post which may be photos or videos as well as text regarding that category. For example, if we have the category travel and if we added our interest as travel then we will be seeing post that are based on travel and of all the users that are posting the posts related to travel. This step intended to help the user and analyst understand the real problem rather than its symptoms. The user or the analyst may identify the need for a candidate system or for enhancement in the existing system.

An analyst is responsible for performing following tasks:

- Studied strength and weakness of the current system.
- Determined “what” must be done to solve the problem.
- Prepared a functional specifications document.

These modules are developed with the aim of reducing time, reducing manpower so that everything can be easily maintained and. The volume of work and complexity are increasing year by year. This system reduces complexity and time. Also provide availability 24*7.

1.4PROBLEM STATEMENT

In the existing system all the work is done manually. The list contains items that carries the post which may be photos or videos as well as text regarding that category. For example, if we have the cataegory travel and if we added our interest as travel then we will be seeing post that are based on travel and of all the users that are posting the posts related to travel. This is chance of committing errors and it will take more time to perform or checkout any information. There are so many limitations in the existing system. So the existing system should be automized.

- In the traditional system, if you wish to analyze any record you have to turn pages many time.
- Existing systems are time consuming as it requires too much planning and so much human involvement.
- As it involves much human involvement, the cost of the system automatically gets increased.
- Existing systems require paper use, which isn’t good for the environment.

1.5 HARDWARE / SOFTWARE USED IN PROJECT

1.5.1 HARDWAREREQUIREMENT

Hardware	Configuration
Processor	Intel(R)core(TM)i5-7200UCPU @2.50GHz
Ram	4GB
Monitor	Normal

1.5.2 SOFTWAREREQUIREMENT

Software	Configuration
OperatingSystem	Windows
Language	NodeJs
DataBase	MongoDB

1.5.3SOME REQUIREMENTS

Performance Requirements:
To achieve good performance the following requirements must be satisfied

- Scalability: The ease with which a system or component can be modified to fit the problem area.
- Portability: The ease with which a system or component can be transferred from one hardware or software environment to another.
- Security: It is the ideal state where all information can be communicated across the internet / company secure from unauthorized persons being able to read it and/or manipulate it..
- Maintainability: The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment.
- Reliability: The ability of a system or component to perform its required functions under stated conditions for a specified period of time.
- Reusability: The degree to which a software module or other work product can be used in more than one computing program or software system.

Safety Requirements:

In case scenarios where data integrity can be compromised, measures should be taken to ensure that all changes are made before system is shutdown. The user must have a registered account to use all facility of the web application.

1.5.4 OTHER REQUIREMENTS

1.5.4.1 MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

10gen software company began developing MongoDB in 2007 as a component of a planned platform as a service product. In 2009, the company shifted to an open-source development model, with the company offering commercial support and other services. In 2013, 10gen changed its name to MongoDB Inc.

BENEFITS OF SALESFORCE.COM

A. MongoDB supports field, range query, and regular-expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions. Queries can also be configured to return a random sample of results of a given size.

B. Fields in a MongoDB document can be indexed with primary and secondary indices.

C. MongoDB provides high availability with replica sets. A replica set consists of two or more copies of the data. Each replica-set member may act in the role of primary or secondary replica at any time. All writes and reads are done on the primary replica by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary.

D. If the replicated MongoDB deployment only has a single secondary member, a separate daemon called an arbiter must be added to the set. As a consequence, an idealized distributed MongoDB deployment requires at least three separate servers, even in the case of just one primary and one secondary.

E. MongoDB scales horizontally using sharding. The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards.

F. MongoDB can be used as a file system, called GridFS, with load balancing and data replication features over multiple machines for storing files.

G. The aggregation framework enables users to obtain the kind of results for which the SQL GROUP BY clause is used. Aggregation operators can be strung together to form a pipeline – analogous to Unix pipes. The aggregation framework includes the

$lookup operator which can join documents from multiple collections, as well as statistical operators such as standard deviation.

H. MongoDB supports fixed-size collections called capped collections. This type of collection maintains insertion order and, once the specified size has been reached, behaves like a circular queue.

1.5.4.2 NodeJs

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm,[6] unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.
Nodejs has certain features like listed below.

- Integrated
- Easy to use
- Asynchronous and Event Driven
- Hosted
- Single Threaded but Highly Scalable
- Easy to test
- License
- No Buffering

1.6 PROJECT SCHEDULE

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, costs and schedule. These estimates are made within a limited time frame at the beginning of a software project and should be updated regularly as the project progresses. In addition, estimates should attempts to define “best case” and “worst case” scenarios so that project outcomes can be bounded.

The first activity in software project planning is the determination of software scope. Function and performance allocated to software during system engineering should be assessed to establish a project scope that is ambiguous and understandable at Presidency and technical levels. Software scope describes function, performance, constraints, interfaces and reliability.

During early stages of project planning, a microscopic schedule is developed. This type of schedule identifies all major software engineering activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into detailed schedule. Here specific software tasks are identified and scheduled.

Scheduling has following principles:

1. Compartmentalization: the project must be compartmentalized into a number of manageable activities and tasks.
2. Interdependency: the interdependencies of each compartmentalized activity or tasks must be determined.
3. Time allocation: each task to be scheduled must be allocated some number of work units.
4. Effort validation: every project has a defined number of staff members.
5. Defined responsibilities: every task that is scheduled should be assigned to a specific team member.
6. Defined outcomes: every task that is scheduled should have a defined outcome.

1.6.1 Pert chart

Program evaluation and review technique (pert) is a project scheduling method that is applied to software development.

Pert provide quantitative tool that allow the software planner to-Determine the critical path-the chain of tasks that determines the duration of the project;Establish "most likely" time estimates for individual tasks by applying statistical models; and

Calculate "boundary times" that defines a time "window" for a particular task.

Pert chart(program evolution review technique) for project-

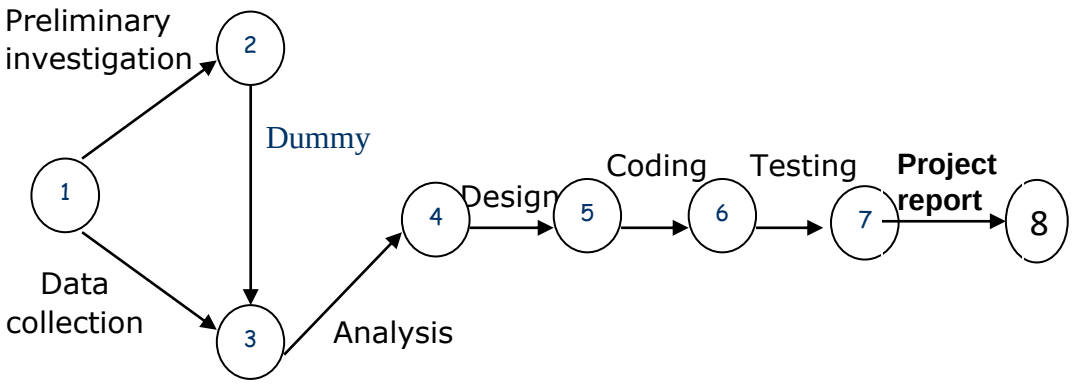


Figure 1.1 Pert chart

1.6.2 Gantt Chart

When creating a project schedule, the planner begins with a set of tasks (the work breakdown structure). If automated tools are used, the work breakdown is input as a task network. Effort, duration and start dates are input are each task network. As a consequence of this input, a timeline chart also called a Gantt chart is generated. A timeline chart is developed for entire project.

Gantt chart for project:

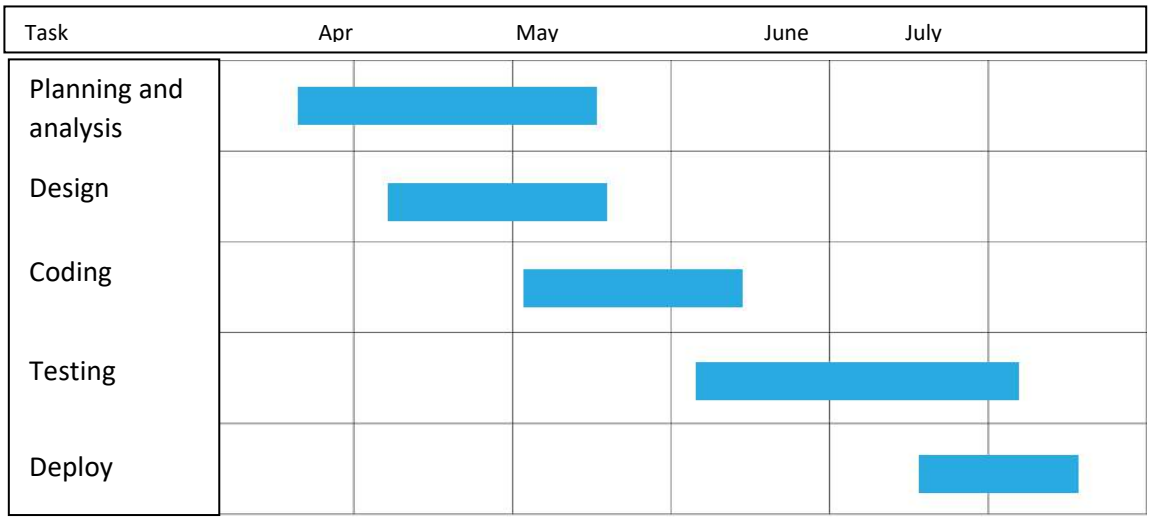


Figure 1.2:Gannt chart for project

Here horizontal bars indicate the duration of each task.

CHAPTER 2

LITERATURE REVIEW

ABSTRACT

Our software is planned for the developing up an Institute Presidency Application that will help to manage all data. This project is a mobile application project. This project is all about creating the list based on interest and category. The list contains items that carry the post which may be photos or videos as well as text regarding that category. For example, if we have the category travel and if we added our interest as travel then we will be seeing post that are based on travel and of all the users that are posting the posts related to travel. So the maintenance became very easy.[16]

Node.js isan open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.[18]

KEYWORDS:- NodeJs, MongoDB, Amazon Web Services, Typescript ,VS Code, Firebase.

2.1 INTRODUCTION

The objective of this application is to let that how a normal person know how and which to post and place or to check their interests and list of feeds and videos related to it can use this application easily, it is flexible like data can be deleted enter or updated easily.[2]

“Notch” is project is a mobile application project. This project is all about creating the list based on interest and category. The list contains items that carry the post which may be photos or videos as well as text regarding that category. For example, if we have the category travel and if we added our interest as travel then we will be seeing post that are based on travel and of all the users that are posting the posts related to travel.

We are making this project with the help of NodeJS, database used is MongoDB.[4] Creating this application and some feature we need:

So, the maintenance and maintenance of institute became very easy, it provide:-

- Easy accessibility.
- It makes searching of data easier and faster.
- Easy availability.
- liberates lengthy manual records.

So our project is providing convenience in different aspects.[17]

2.2 RELATED WORK

2.2.1 Book directory

This is the most basic project that can be created using the Node.JS framework. Building a book directory is essential for developing a REST API. To approach this, you can look at some endpoints in the directory using the following four methods – GET, POST, PUT, and DELETE.[1] This is the most basic project that can be created using the Node.JS framework.

2.2.2 Real-time chat apps

Node.JS is perfect for developing real-time messenger applications. Everything from sending messages to displaying them to noting the timestamps and displaying receipts can be seamlessly handled using the Node.JS framework – and all of this without the user having to refresh the page. If you've used any chat platform – Facebook, WhatsApp, and Instagram – you'll know exactly what we're talking about. [6]

The powerful Event API present in Node.JS emits "listener" events that are emitted by event handlers. Because of this powerful functionality, Node makes it easier to implement push notifications and server-side events in IM and other real-time chats.

2.2.3 Basic Users System

This is yet another easy project that is excellent for getting real-life experience of working with Node.JS. Despite being basic, the project will help you practice some essential skills that will definitely come in handy throughout your career. While building a basic user system, you'll explore concepts like – setting up databases, performing migrations, adding new users, building login endpoints, authenticating users, getting the users' data, and more.[7]

Try to generate a JWT token for the user that the API will return in case of login and registrations. You must also remember to hash the password before you save it in your database.

2.2.4 Complex Single-Page Applications

Think of single-page applications like desktop applications instead of static web pages. Using Node.JS, you'll be able to add a lot of dynamicity and smoothness to your single-page application, thereby replicating the feel of a proper desktop application. Single-page applications are extremely useful in the industry and are utilized for creating social networking platforms.[8]

2.2.5 Real-time Collaboration Apps

A real-time collaboration application should provide features like audio and video conferencing, document sharing, project management, and collaborative working on different documents. Think of Trello, Slack, and Google Docs – that is what we mean by real-time collaboration applications.

The event-based and asynchronous architecture offered by Node.JS is ideally suited for building such collaboration apps. In any real-time collaboration application, many I/O requests and events are happening

concurrently. With Node.JS, all these events are handled seamlessly without stressing the server even a bit.[9]

2.2.6 Email Sender

The last idea in our list of Node.JS project topics is an email sender. Using Node.JS, you can create an application to send across emails. In doing so, you'll be familiarizing yourself with the process of sending emails using Node.JS as the medium, and that's an instrumental skill that'll come in handy for any complex application you create in the future. For this project, you can explore the Nodemailer plugin for sending emails. is one of the hot cloud computing tech. This is a very well-documented plugin that is easy to use and understand.[10]

In this we define the projects structure, feature, criteria for success the main objective is to create the desired project. our system is based on cloud.

2.3.2 Implementation And Evaluation

The implementation of the system is done with the help of salesforce and apex.by using of force.com cloud platform. With the help of apex language.by the using of force.com pages leveraging and functionalities the information on our account of NodeJS.[1]

With the help of the custom domain, user can use this domain to access the website.

Figure 2.2 flow of website

2.3 DESIGN

2.3.1 Design

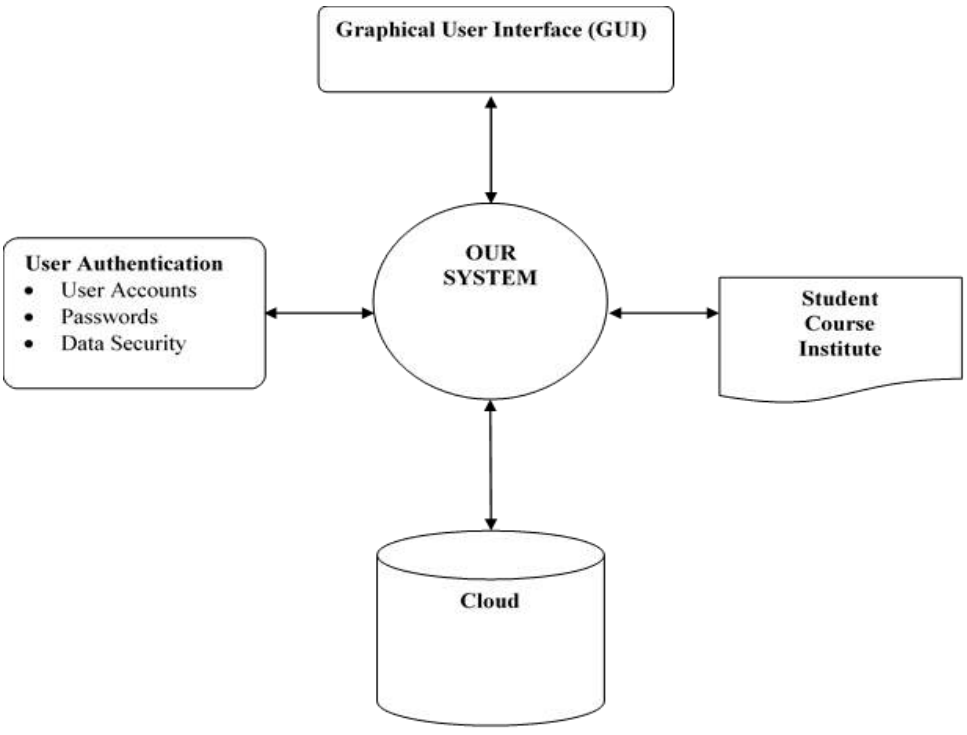


Figure 2.3 Design

2.3.2.1 NodeJs

Node.js is an open-source, cross-platform, back-end JavaScript runtime

environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm,[6] unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

2.3.2.2 MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL). 10gen software company began developing MongoDB in 2007 as a component of a planned platform as a service product. In 2009, the company shifted to an open-source development model, with the company offering commercial support and other services. In 2013, 10gen changed its name to MongoDB Inc.[19]

MongoDB supports field, range query, and regular-expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions. Queries can also be configured to return a random sample of results of a given size.

2.3.3 GUI

The graphical user interface is one important factor by which the user can easily access the website to understand what he need to do in order to use the software effectively.[3]A easy and effective GUI is good for both user as well as developer.
Some of the GUI are as follow:

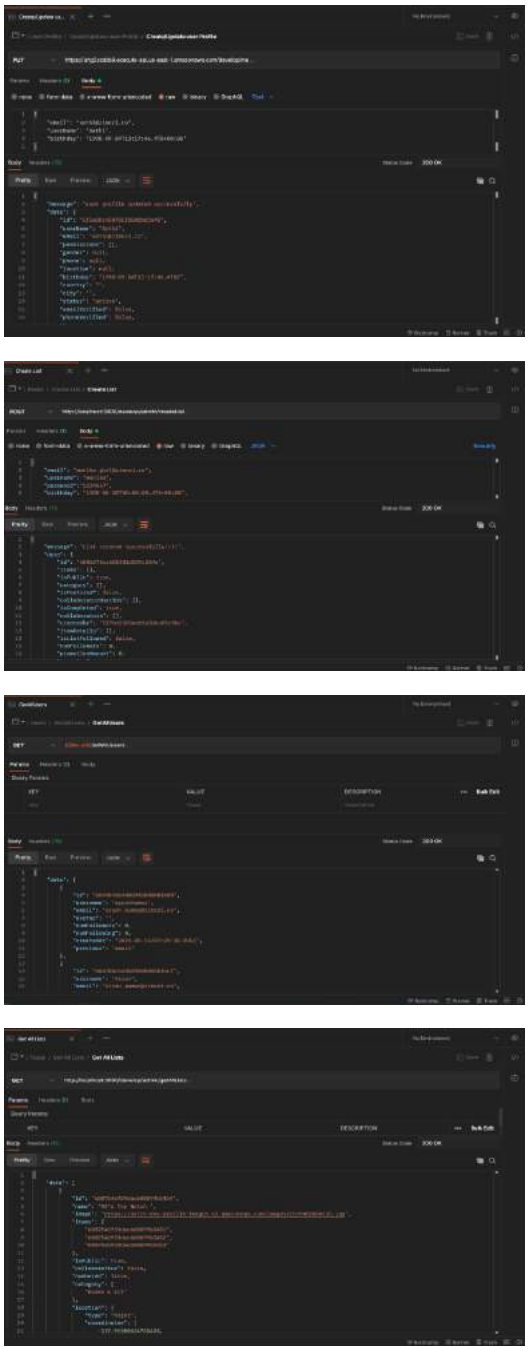


Figure 2.3 some screens

2.4 RESULT

We have come on result that our system working fine in each case. It is user friendly as well as efficient to use. we have done different things to verify the performance.

TEST CASE RESULT

TestCase#	Description	Result
TC#1	Loading the	Passed
TC#2	Login	Passed
TC#3	Validating	Passed
TC#4	Content	Passed
TC#5	Course page	Passed
TC#6	Reports page	Passed
TC#7	Logout	Passed

2.5 DISCUSSION

System was completely done after was duly coded. each modules of the project were checked to ensure they are fully functional units.[4] This was done by checking each unit to give assurance that it functions as required and that it performed exactly as defined. The success of each individual gave us go ahead to carryout testing properly[9].

The defined system was validated by the using a series of short questionnaire that was completely filled by representatives users who have used the system and give suggestion according to the need .This was done to the assess if the system met their respective needs and requirements. It was also find that it is easy to access the data as well as available when needed. With the flexibility of the cloud it is quite useful and better version of listing and posting system.

2.6 CONCLUSION & FUTURE SCOPE

2.6.1 CONCLUSION

A system means a lot of experience. I learned a lot of thing this project development. This project has also sharpened my concept NodeJS.

2.6.2 FUTURE SCOPE

To make our system more user friendly. I will add Helping BOT in the system. Adding of Video on the module.

2.7 REFRENCES

[1] "The benefits of web-based applications," [Online]. Available: http://www.magicwebsolutions.co.uk/blog /the-benefits-of-web-basedapplications.htm. [Accessed 25 November 2016].

[2] F. Bridge, "What Types of Developers Are There?," tree house, 24 June 2016. [Online]. Available: http://blog.teamtreehouse.com/what-types-of-developer-are-there. [Accessed 25 November 2016].

[3] M. Wales, "Front-End vs Back-End vs Full Stack Developers," Udacity, 08 December 2014. [Online]. Available:http://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html. [Accessed 25 November 2016].

[4] J. Long, "I Don’t Speak Your Language: Frontend vs. Backend," tree house, 25 September 2012. [Online]. Available: http:// blog.teamtreehouse.com/ i-dont-speak-your-language-frontend-vs-

backend. [Accessed 25 November 2016].
[5] A. Mardan, "PHP vs. Node.js," Programming Weblog, [Online]. Available: http://webapplog.com/ php-vs-node-js/. [Accessed 28 January 2016].
[6] J. Kaplan-Moss, "Quora," [Online]. Available: https://www.quora.com/What-are-the-benefits-of-developing-in-Node-js-versus-Python. [Accessed 29 June 2016].
[7] "Node.js Tutorial," tutorials point, [Online]. Available: https://www.tutorialspoint.com/node js/index.htm. [Accessed 25 November 2016].
[8] R. R. McCune, "Node.js Paradigms and Bench-marks," 2011
[9] N. Chhetri, "A Comparative Analysis of Node.js (Server-Side JavaScript)," Culminating Projects in Computer Science and Information Technology., p. 5, 2016.
[10] The Apache Software Foundation, "Apache MINA," [Online]. Available: http://mina.apache.org/. [Accessed 30 June 2016].
[11] The Apache Software Foundation, "Apache MINA," [Online]. Available: http://mina.apache.org/ async- web-project/index.html. [Accessed 30 June 2016].
[12] Ryan Dahl: Original Node.js presentation. [Film]. Youtube, 2012.
[13] "How Loading Time Effects Your Bottom Line," Kissmetrics Blog, [Online]. Available: https://blog.kissmetrics.com/loading -time/. [Accessed 25 November 2016].
[14] C. Buckler, "Site Point Smack Down: PHP vs Node.js," Site Point, [Online]. Available: http://www. sitepoint.com/sitepoint-smackdown-php-vs-node-js/. [Accessed 28 January 2016].
[15] Firehose, "Firehose," Firehose Project, [Online]. Available:http://blog.thefirehoseproj ect.com/posts/nodejs-vs-rails/. [Accessed 30 June 2016].
[16] "Quora," [Online]. Available: https://www.quora. com/What-are-the-disadvantages-of-using-Node-js. [Accessed 30 June 2016].
[17] A Harris, "The Birth of Node: Where Did it Come From? Creator Ryan Dahl Shares the History," silicon ANGLE, 01 April 2016. [Online]. Available: http://siliconangle.com/blog/2013/0 4/01/the-birth-of-node-where-did-it-come-from-creator-ryan-dahl-shar es-the-history/. [Accessed 18 November 2016].
[18] L. ORSINI, "What You Need To Know About Node.js," read write, 07 November 2013. [Online]. Available: http://readwrite.com/2013/11/07/wh at-you-need-to-know-about-nodejs/. [Accessed 2016 November 2016].
[19] G. Nemeth, "Rising Stack Engineering," [Online]. Available: https://blog.risingstack.com/history-of-node-js/. [Accessed 27 October 2016].

CHAPTER 3

FEASBILITY STUDY

3.1INTRODUCTION

Feasibility of the system in an important aspect, which is to be considered. The system needs to satisfy the law of economic, which states that the maximum output should be yielded in minimum available resources.

A feasibility analysis evaluates the project's potential for success; therefore, perceived objectivity is an essential factor in the credibility of the study for potential investors and lending institutions. There are five types of feasibility study—separate areas that a feasibility study examines, described below.

1. Technical Feasibility

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of the proposed system. As an exaggerated example, an organization wouldn't want to try to put Star Trek's transporters in their building—currently, this project is not technically feasible.

2. Economic Feasibility

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility—helping decision-makers determine the positive economic benefits to the organization that the proposed project will provide.

3. Legal Feasibility

This assessment investigates whether any aspect of the proposed project conflicts with legal requirements like zoning laws, data protection acts or social media laws. Let's say an organization wants to construct a new office building in a specific location. A feasibility study might reveal the organization's ideal location isn't zoned for that type of business. That organization has just saved considerable time and effort by learning that their project was not feasible right from the beginning.

4. Operational Feasibility

This assessment involves undertaking a study to analyze and determine whether—and how well—the organization's needs can be met by completing the project. Operational feasibility

studies also examine how a project plan satisfies the requirements identified in the requirements analysis phase of system development.

5. Scheduling Feasibility

This assessment is the most important for project success; after all, a project will fail if not completed on time. In scheduling feasibility, an organization estimates how much time the project will take to complete.

When these areas have all been examined, the feasibility analysis helps identify any constraints the proposed project may face, including:

- Internal Project Constraints: Technical, Technology, Budget, Resource, etc.
- Internal Corporate Constraints: Financial, Marketing, Export, etc.
- External Constraints: Logistics, Environment, Laws, and Regulations, et

3.2 MAIN ASPECTS

There are three aspects of feasibility to be considered namely.

1. Technical
2. Operational
3. Economical

TECHNICAL:

In the technical aspects one may consider the hardware equipment for the installation of the software. The system being centralized will required very little hardware appliances. Hence this helps the system to work smoothly with limited amount of working capitals.

OPERATIONAL:

In the operational aspects may think of the benefits of the workload that many a personal may have to share. This is eased out and the required output may be retrieved in a very short time. Thus there is accuracy in the work on time is also saved there will be very little work that needs to be performed.

ECONOMICAL:

Economical system is definitely feasible because the hardware requirement is less and the operational working for the system requires less number of recruits. This help introduction over-staffing and wastage funds.

We studied on the position to evaluate solution. Most important factors in this study were tending to overlook the confusion inherent in system Development the constraints and the assumed studies. It can be started that it the feasibility study is to serve as a decision document it must answer three key questions.

1. Is there a new and better way to do the job that will benefit the user?
2. What are the costs and savings of the alternatives?
3. What is recommended?

On these questions it can be explained that feasibility study of the system includes following different angles.

3.2.1 Technical feasibility:

This centers on the existing computer system (hardware, software etc.) and to what extent it can support the proposed additional equipment .in this stage of
study, we have collected information about technical tools available by which I could decide my system design as the technical requirements.

3.2.2 Operational Feasibility:

In this stage of study we have checked the staff availability. I concentrate on knowledge of end users that are going to use the system. This is also called as behavioral feasibility in which I have studied on following aspects; people are inherently resistant to change, and computers have been known to facilitate change .An estimate has been made to how strong a reaction the user staff is having toward the development of a computerized system. It is common knowledge that computer installations have something to do with turnover. I had explained that there is need to educate and train the staff on new ways of conducting business.

3.2.3Economical feasibility:

Economical analysis is the most frequently used method for evaluating the effectiveness of candidate system. More commonly known as cost\benefit analysis, the procedure is to determine the benefits and savings that benefits outweigh costs. The decision was to design and implement system because it is for having chanced to be approved. This is an on going effort that improves the accuracy at each phase of the system life cycle.
In developing cost estimates for a system I need to consider several cost elements. Among these is hardware personal facility. Operating and supply costs.

3.3BENEFITS

Benefits of conducting a feasibility study:

- Improves project teams' focus
- Identifies new opportunities
- Provides valuable information for a "go/no-go" decision
- Narrows the business alternatives
- Identifies a valid reason to undertake the project
- Enhances the success rate by evaluating multiple parameters
- Aids decision-making on the project
- Identifies reasons not to proceed

3.4SYSTEM REQUIREMENT SPECIFICATION

Any system can be designed after specifies the requirement of the user about that system. For this first of all gathered information from user by the preliminary investigation which is starting investigation about user requirement..

The data that the analysts collect during preliminary investigation are gathered through the various preliminary methods.

Documents Reviewing Organization

The analysts conducting the investigation first learn the organization involved in, or affected by the project. Analysts can get some details by examining organization charts and studying written operating procedures.

Collected data is usually of the current operating procedure:

- The information relating to clients, projects and students and the relationship between them was held manually.
- Managing of follow-ups was through manual forms.
- Complaints require another tedious work to maintain and solve.
- Payments details had to be maintained differently.

Gathering Information By Asking Questions

Interviewing is the most commonly used techniques in analysis. It is always necessary first to approach someone and ask them what their problems are, and later to discuss with them the result of your analysis.

Questionnaires

Questionnaires provide an alternative to interviews for finding out information about a system. Questionnaires are made up of questions about information sought by analyst. The questionnaire is then sent to the user, and the analyst analyzes the replies.

Electronic Data Gathering

Electronic communication systems are increasingly being used to gather information. Thus it is possible to use electronic mail to broadcast a question to a number of users in an organization to obtain their viewpoint on a particular issue.

In my project, with the help of Marg software solutions, I have send questionnaire through electronic mail to twenty employees of the company and retrieved the information regarding the problem faced by existing system.

Interviews

Interview allows the analysts to learn more about the nature of the project request and reason of submitting it. Interviews should provide details that further explain the project and show whether assistance is merited economically, operationally or technically.

One of the most important points about interviewing is that what question you need to ask.

It is often convenient to make a distinction between three kinds of question that is

- Open questions
- Closed question
- Probes

Open questions are general question that establish a persons view point on a particular subject.

Closed questions are specific and usually require a specific answer.

Probes are question that follow up an earlier answer.

CHAPTER 4

DESIGN

4.1 INTRODUCTION

System is created to solve problems. One can think of the systems approach as an organized way of dealing with a problem. In this dynamic world, the subject system analysis and design, mainly deals with the software development activities.

Since a new system is to be developed, the one most important phases of software development life cycle is system requirement gathering and analysis. Analysis is a detailed study of various operations performed by a system and their relationship within and outside the system. Using the following steps it becomes easy to draw the exact boundary of the new system under consideration.

All procedures, requirements must be analysed and documented in the form of detailed DFDs, logical data structure and miniature specifications.

System analyses also include sub-dividing of complex process involving the entire system, identification of data store and manual processes

4.2 SYSTEM DESIGN

System design is the process of planning a new system or to replace the existing system. Simply, system design is like the blueprint for building, it specifies all the features that are to be in the finished product.
System design phase follows system analysis phase. Design is concerned with identifying functions, data streams among those functions, maintaining a record of the design decisions and providing a blueprint the implementation phase.
Design is the bridge between system analysis and system implementation. Some of the essential fundamental concepts involved in the design of application software are:

- Abstraction
- Modularity
- Verification

Abstraction is used to construct solutions to problem without having to take account of the intricate details of the various component sub problems. Abstraction allows system designer to make step-wise refinement, which at each stage of the design may hide, unnecessary details associated with representation or implementation from the surrounding environment.

Modularity is concerned with decomposing of main module into well-defined manageable units with well-defined interfaces among the units. This enhances design clarity, which in turn eases implementation, Debugging, Testing, Documenting and Maintenance of the software product. Modularity viewed in this sense is a vital tool in the construction of large software projects.

Verification is fundamental concept in software design. A design is verifiable if it can be demonstrated that the design will result in implementation that satisfies the customer's requirements. Verification is of two types namely.

- Verification that the software requirements analysis satisfies the customer's needs.
- Verification that the design satisfies the requirement analysis.

Some of the important factors of quality that are to be considered in the design of application software are:

Reliability:
The software should behave strictly according to the original specification and should function smoothly under normal conditions.

Extensibility:
The software should be capable of adapting easily to changes in the specification.

Reusability:

The software should be developed using a modular approach, which permits modules to be reused by other application, if possible.
The System Design briefly describes the concept of system design and it contains four sections. The first section briefly describes the features that the system is going to provide to the user and the outputs that the proposed system is going to offer.
The second section namely Logical Design describes the Data Flow Diagrams, which show clearly the data movements, the processes and the data sources, and sinks, E-R diagrams which represent the overall logical design of the database, and high-level process structure of the system.

Preliminary Design:
Preliminary design is basically concerned with deriving an overall picture of the system. Deriving entire system into modules and sub-modules while keeping Cohesion and Coupling factors in mind. Tools, which assist in preliminary design process, are Data Flow Diagrams.

Code design:
The purpose of code is to facilitate the identification and retrieval for items of information. A code is an ordered collection of symbols designed to provide unique identification of an entity or attribute. To achieve unique identification there must be only one place where the identified entity or the attribute can be entered in the code; conversely there must be a place in the code for every thing that is to be identified. This mutually exclusive feature must be built into any coding system.
The codes for this system are designed with two features in mind. Optimum human oriented use and machine efficiencyThey are also operable i.e., they are adequate for present and anticipate data processing both for machine and human use.

Input /Output design:
is a part of overall system design, which requires very careful attention. The main objectives of input design are:

- To produce a cost-effective method of input.
- To achieve the highest possible level of accuracy.
- To ensure that the input is acceptable to and understood by the user staff.

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also to provide a permanent hard copy of these results for later consultation.
The various types of outputs are required by this system are given below:

- External outputs, whose destination is outside the concern and which require special attention because they, project the image of the concern.
- Internal outputs, whose destination is within the concern and which require careful design because they are the user's main interface within the computer.
- Operation outputs, whose use is purely within the computer department, E.g., program listings, usage statistics etc,

4.3 SDLC

Software Development Life Cycle (SDLC) is a framework that defines the steps involved in the development of software at each phase. It covers the detailed plan for building, deploying and maintaining the software.
SDLC defines the complete cycle of development i.e. all the tasks involved in planning, creating, testing, and deploying a Software Product.

Figure 4.1:Above image depicting the planning step

SDLC Phases

Given below are the various phases:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

Requirement Gathering and Analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

Design

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

Deployment

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

4.4 SOFTWARE ENGG. PARADIGM APPLIED

Software engineering is a layered technology. The foundation for software engineering is the process layer. Software engineering processes the glue that holds the technology layers together and enables ratios and timely development of computer software. Process defines a framework for a set of key process areas that must be established for effective delivery of software engineering technology.

Software engineering methods provide the technical how-to's for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing and support. Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another tool, a system for the support of software development, called computer-aided software engineering is established.

The following paradigms are available:

1. The Waterfall Model

2. The Prototyping Model

3. The Spiral model

Etc.

4.4.1 The Prototype model

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

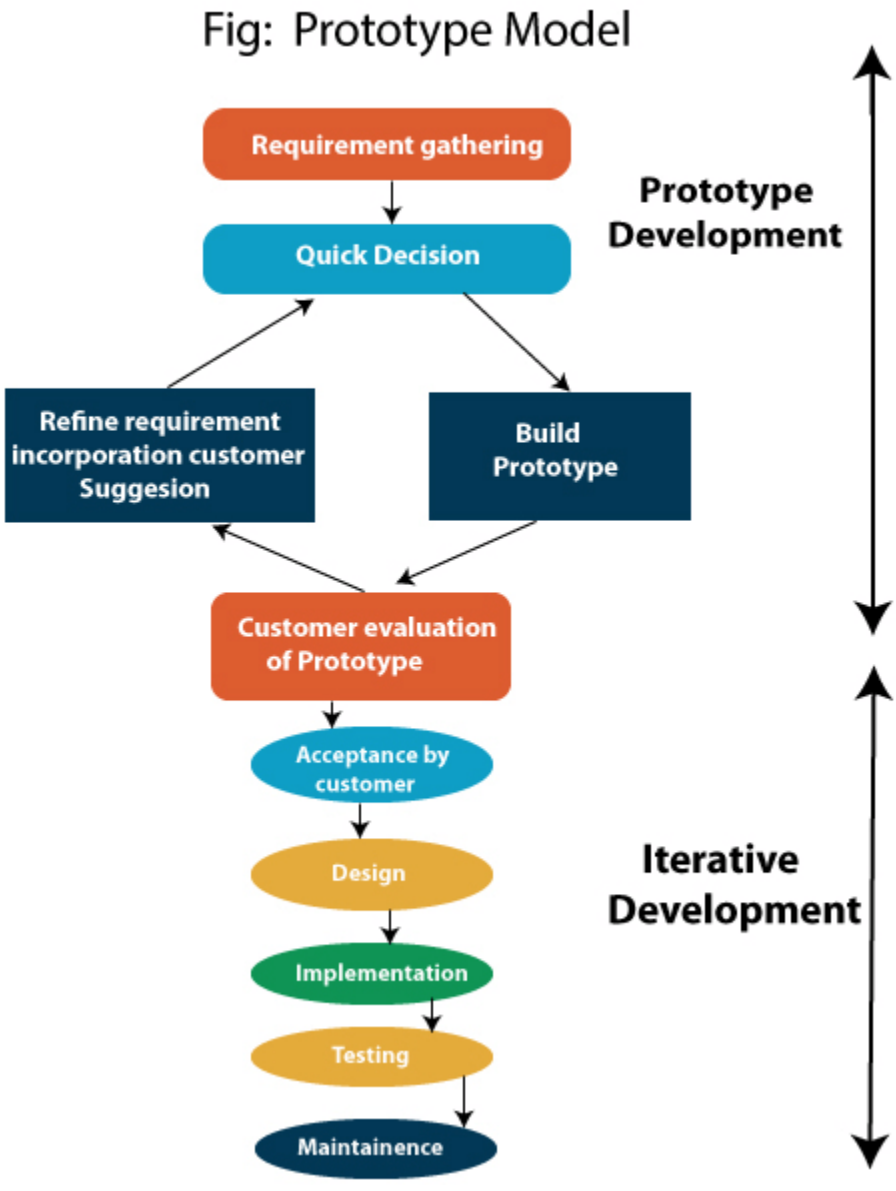


Figure 4.2:Prototype model

4.4.1.1Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids Presidency
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

4.4.1.2 Disadvantage of Prototype Model

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
    - Costs customer money
    - Needs committed customer
    - Difficult to finish if customer withdraw
    - May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required to build a prototype.
7. It is a time-consuming process.

4.5 DFD

DFD is the abbreviation for **Data Flow Diagram**. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. Data Flow Diagram can be represented in several ways.

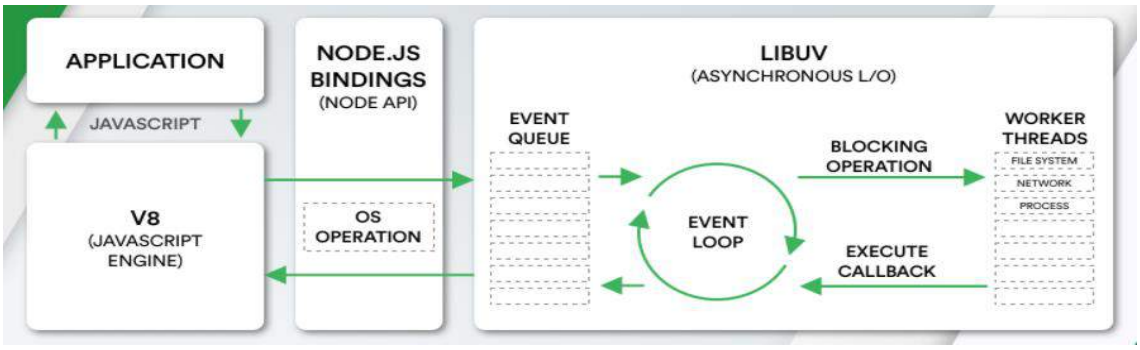


Figure 4.3 Nodejs working

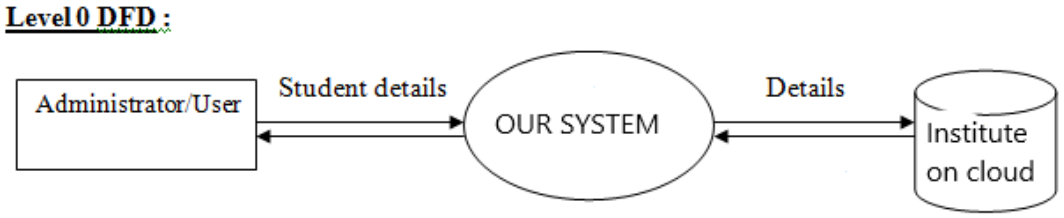


Figure 4.4 Context level

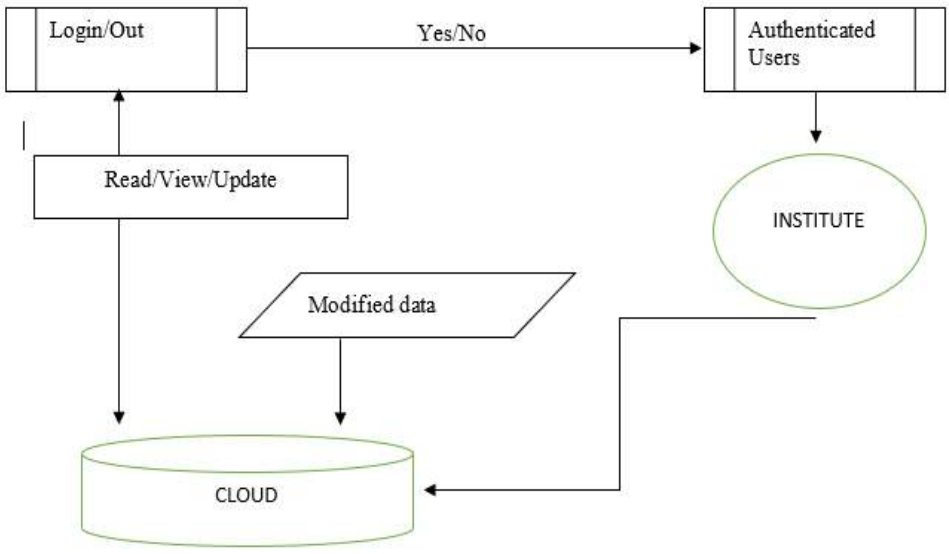


Figure 4.5Dfd 1 level

4.6 UML use case diagram

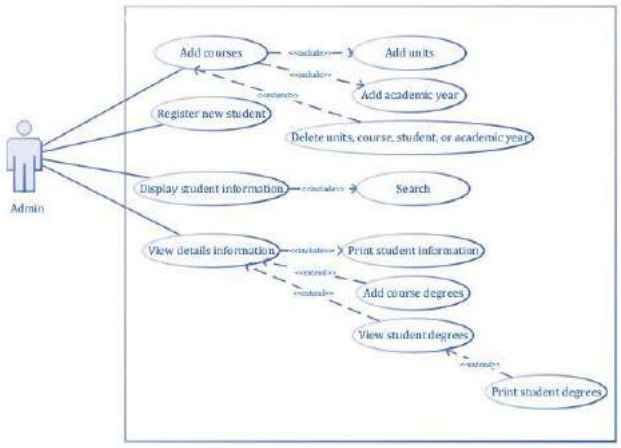


Figure 4.6 Uml use case

4.7 ER DIAGRAM

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

An ER diagram has three main components:
1. Entity
2. Attribute
3. Relationship

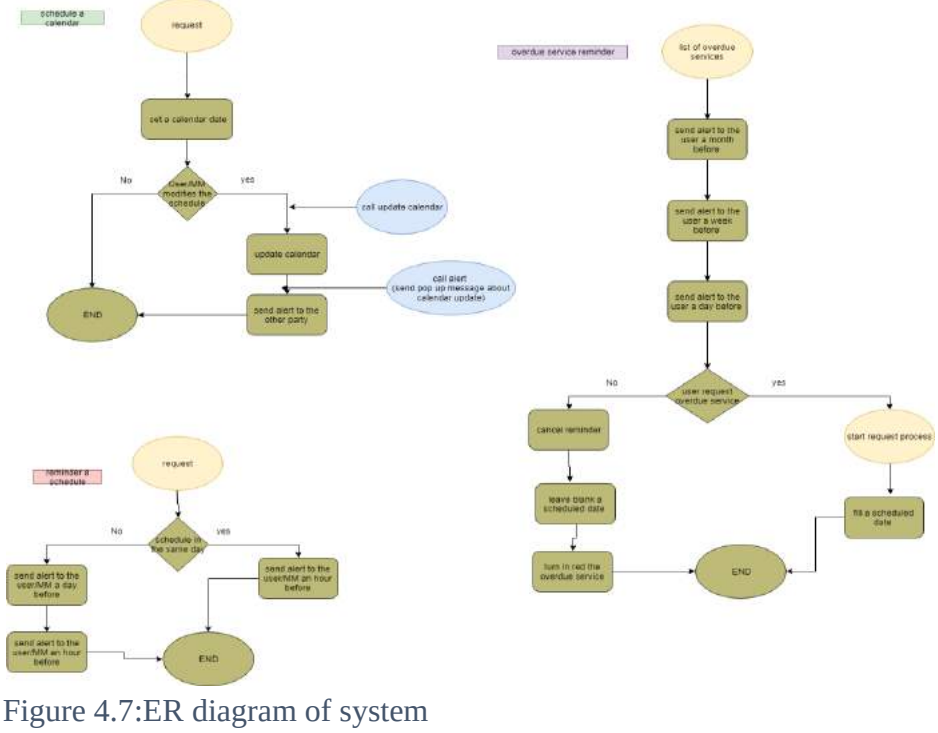


Figure 4.7:ER diagram of system

4.7.1 ER- Diagram Notations

ER- Diagram is a visual representation of data that describe how data is related to each other.

- **Rectangles:** This symbol represent entity types
- **Ellipses :**Symbolrepresent attributes
- **Diamonds:** This symbolrepresents relationship types
- **Lines:** It links attributes to entity types and entity types with other relationship types
- **Primary key:** attributes are underlined
- **Double Ellipses:** Represent multi-valued attributes

CHAPTER 5

REPORT

5.1 GIST

The diagram **figure 5.1,**depicting our system.
We have designed and developed an easy, Useful, reliable system.

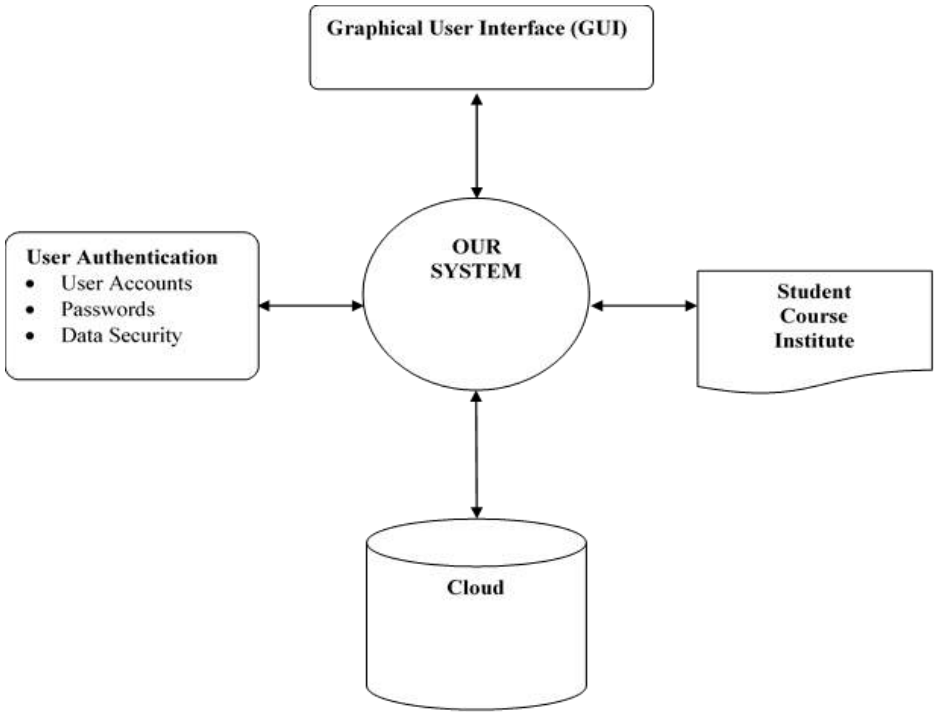
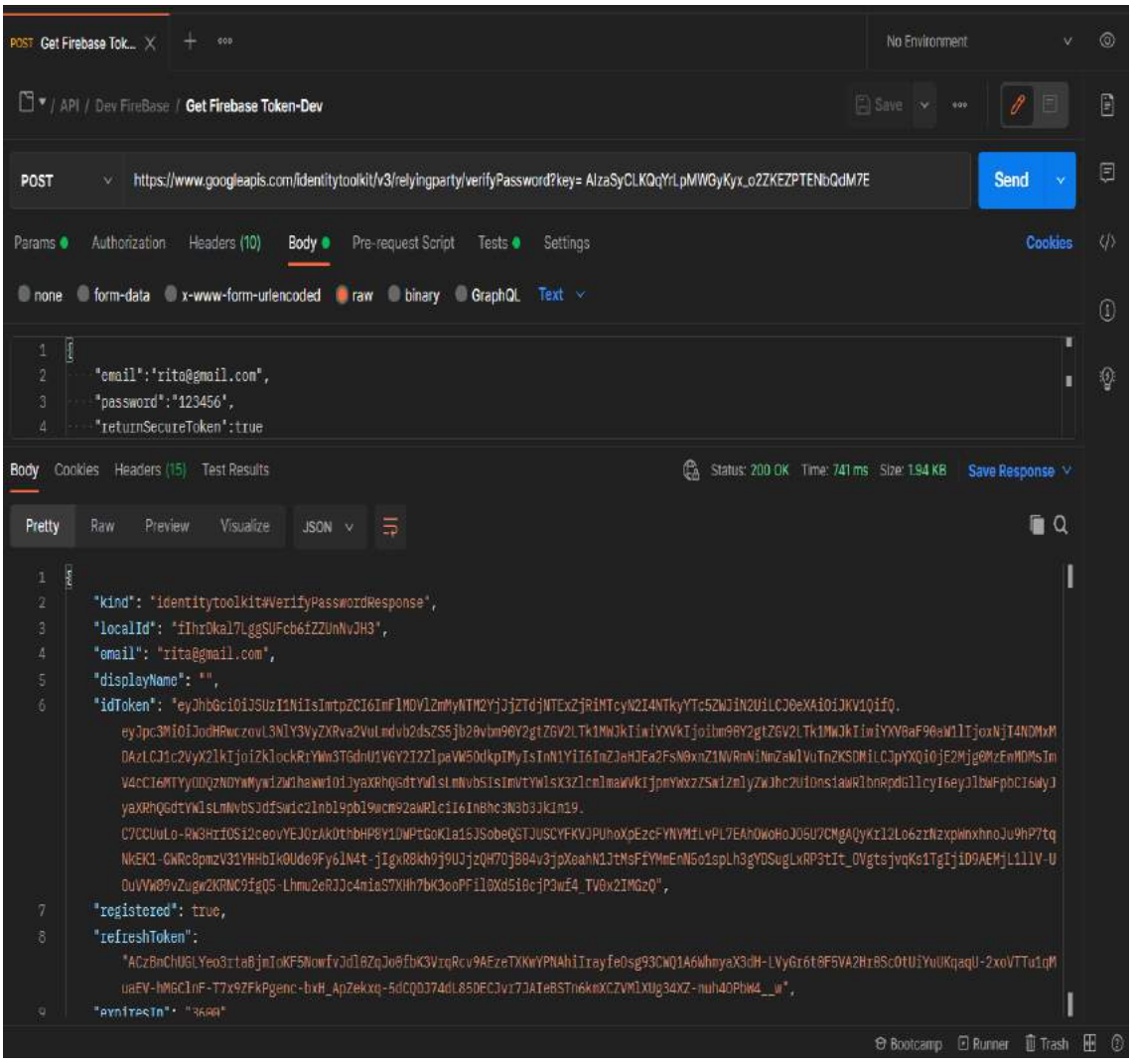


Figure 5.1 System

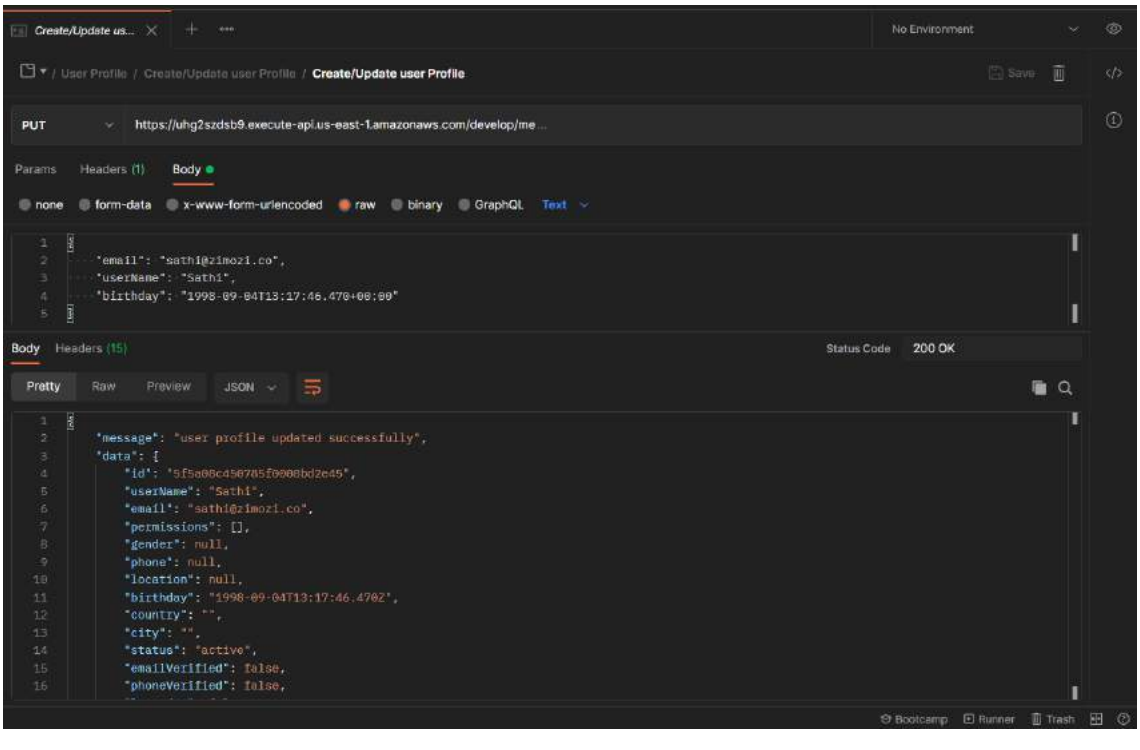
This gives a high level view of the system with the main components and the services they provide and how they communicate. It consists of the general graphical user interface facilities.

5.2 SOME SCREENSHOTS

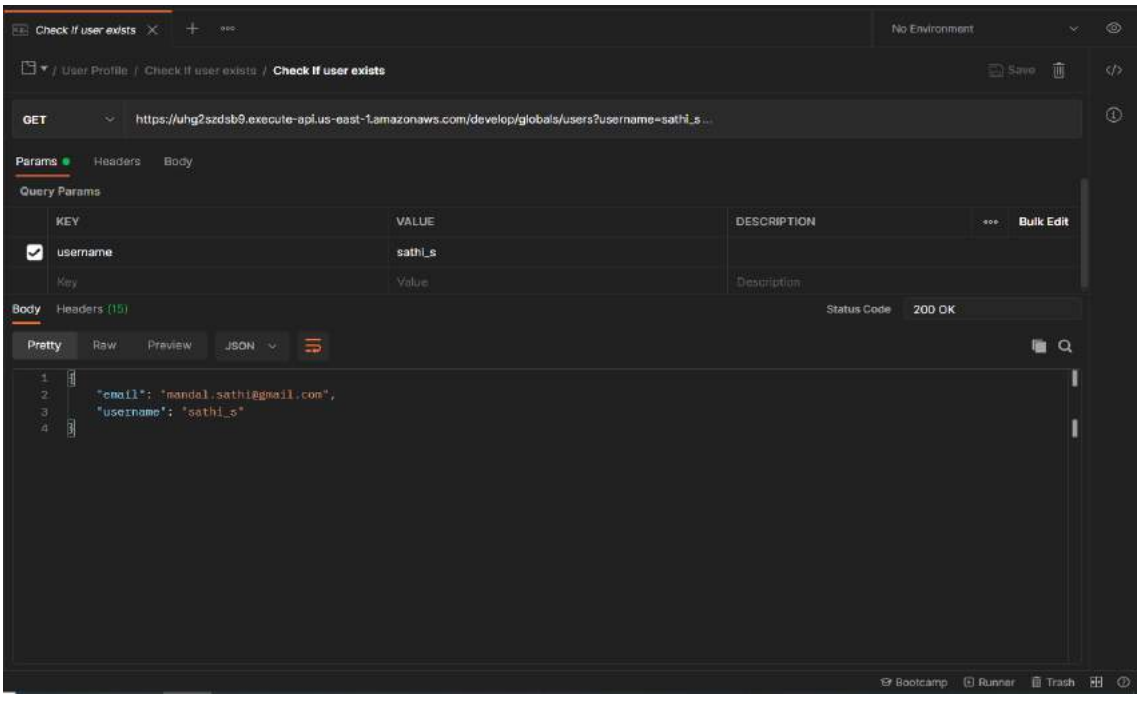
HOME PAGE



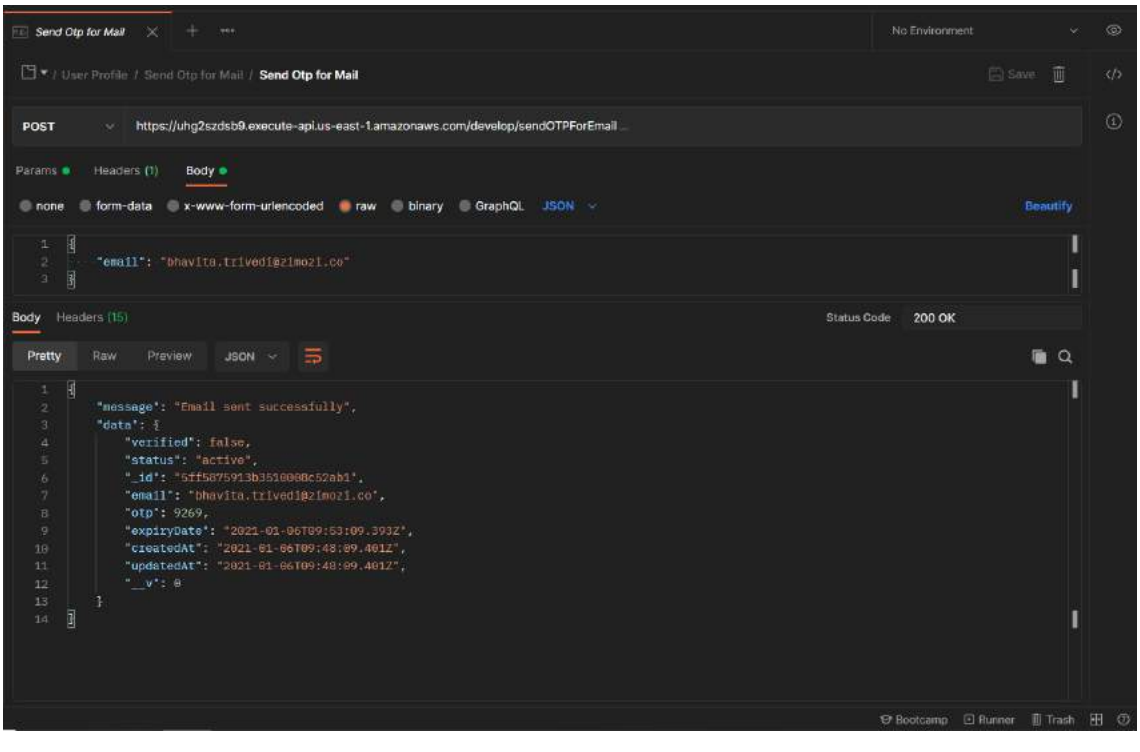
CREATING/UPDATING USER



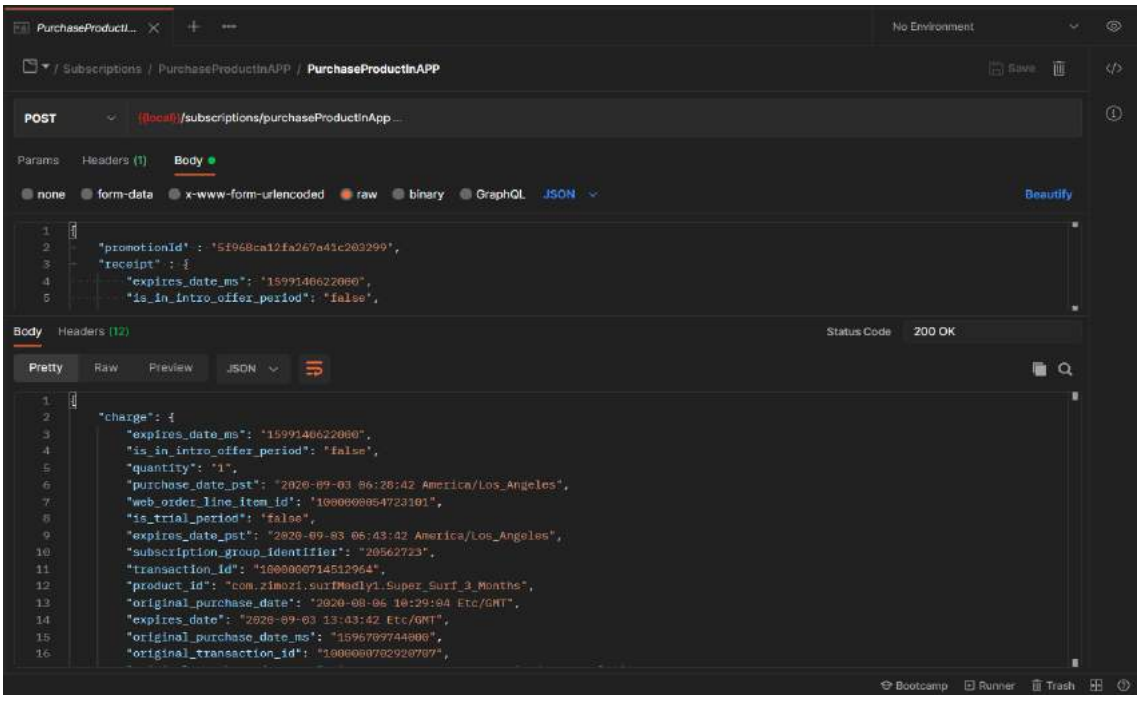
CHECK IF USER EXISTS



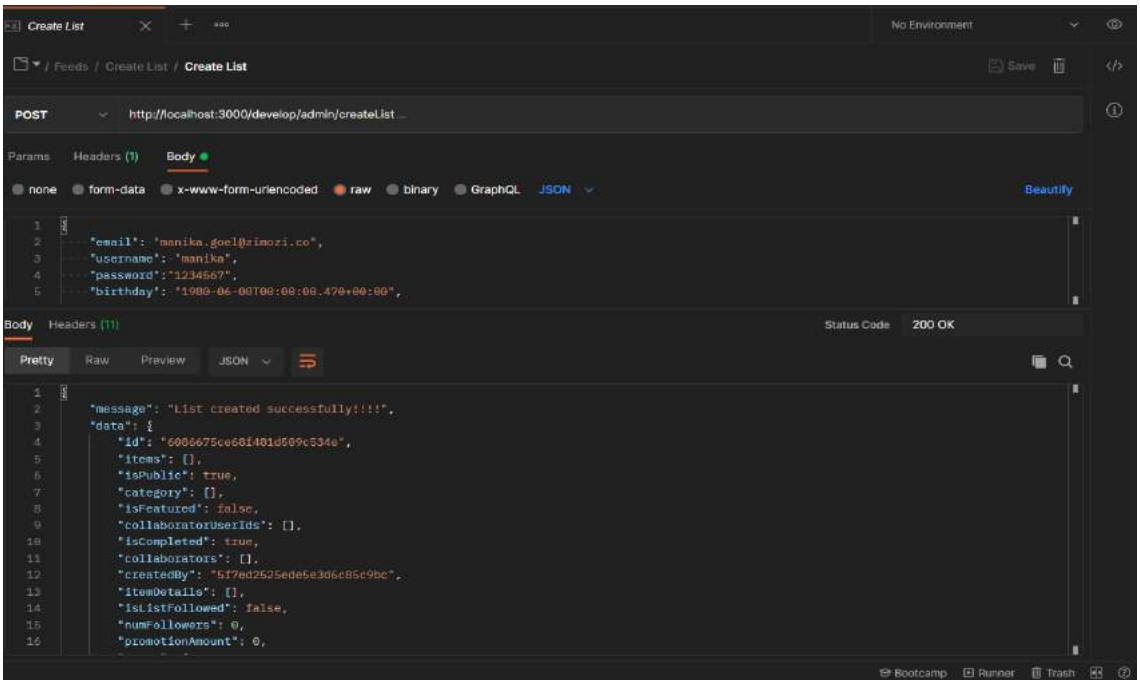
SEND OTP FOR MAIL



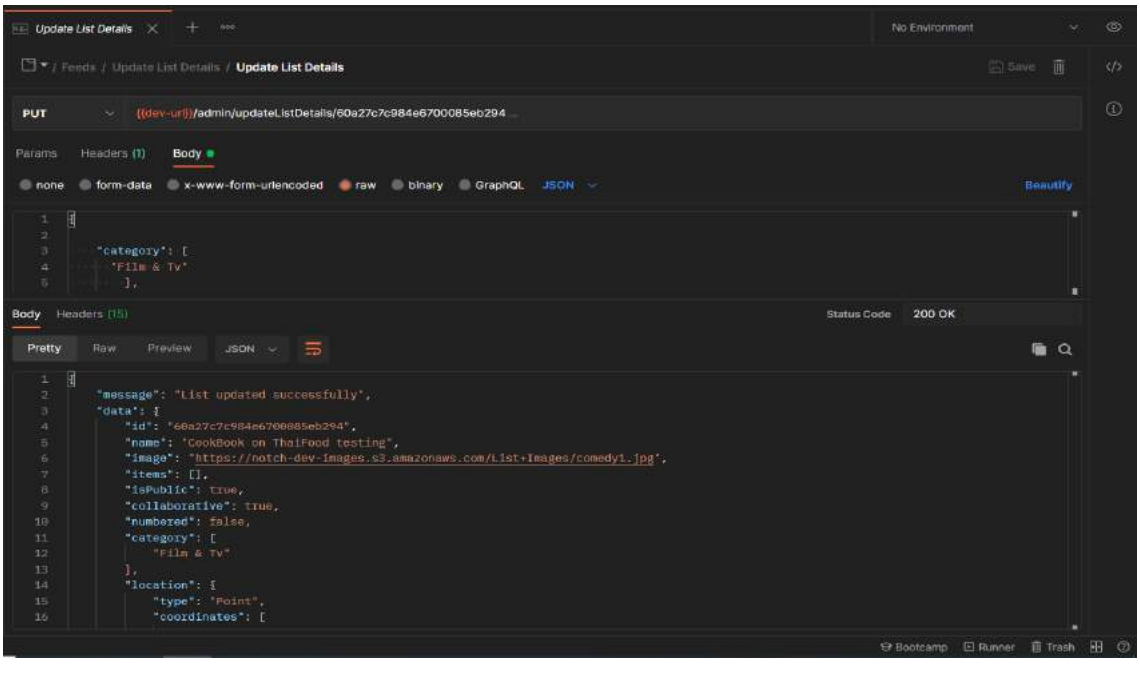
VALIDATION ON COURSE_FEE



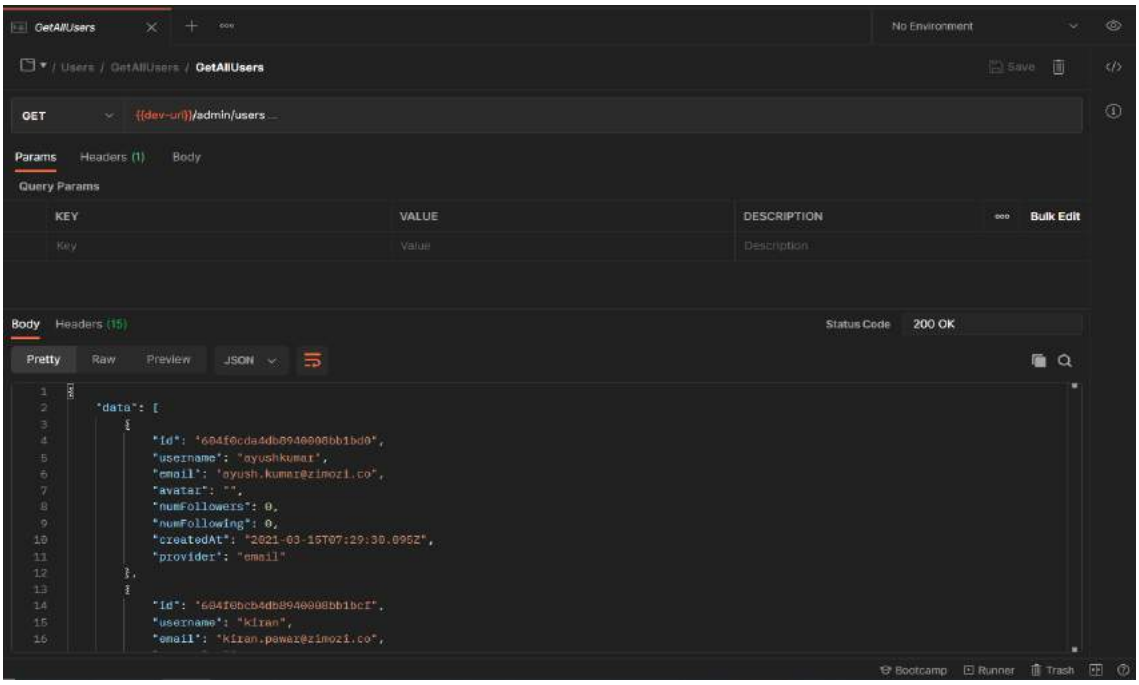
CREATE LIST



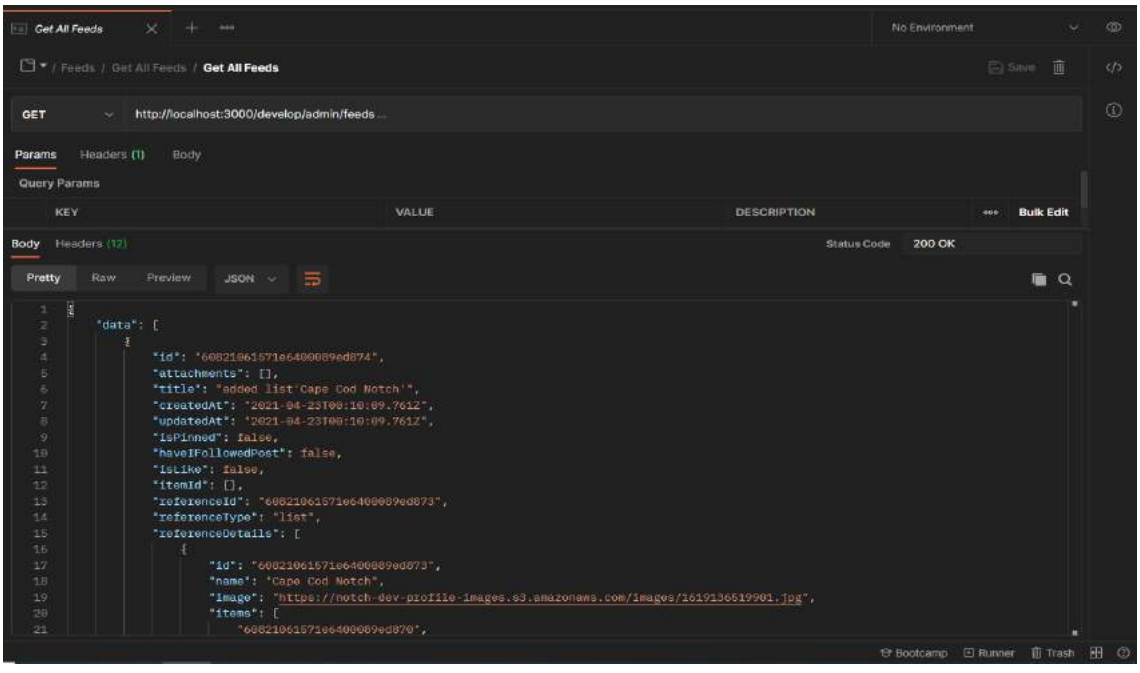
UPDATE LIST DETAILS



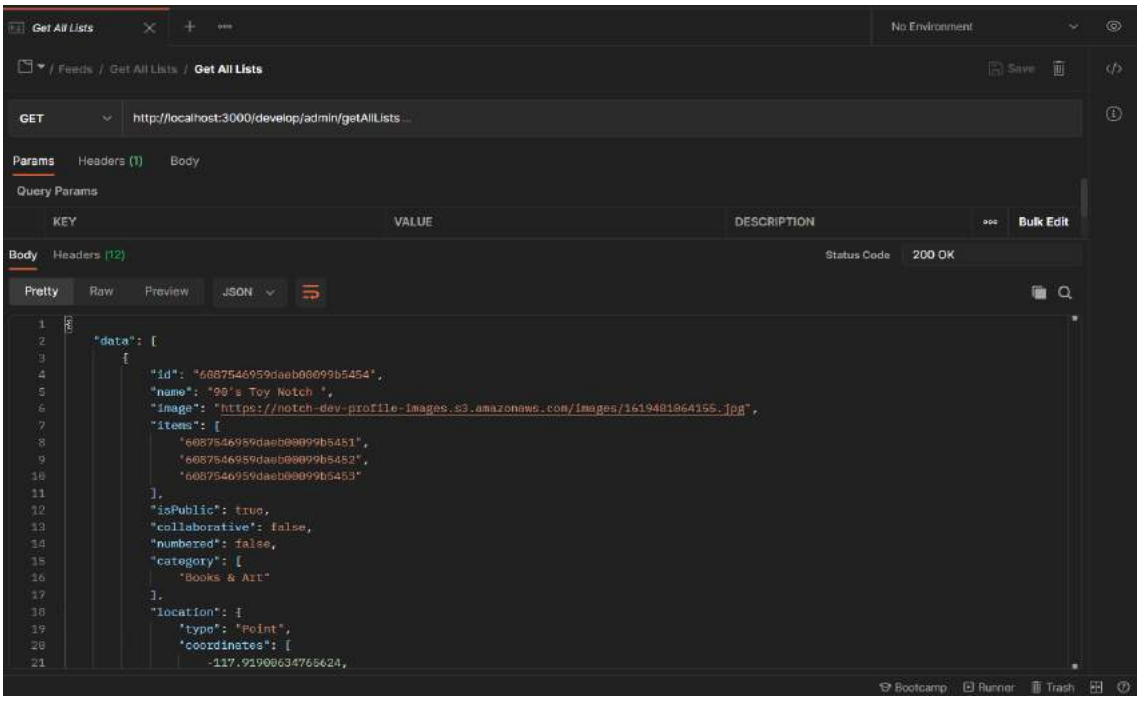
GET ALL USERS



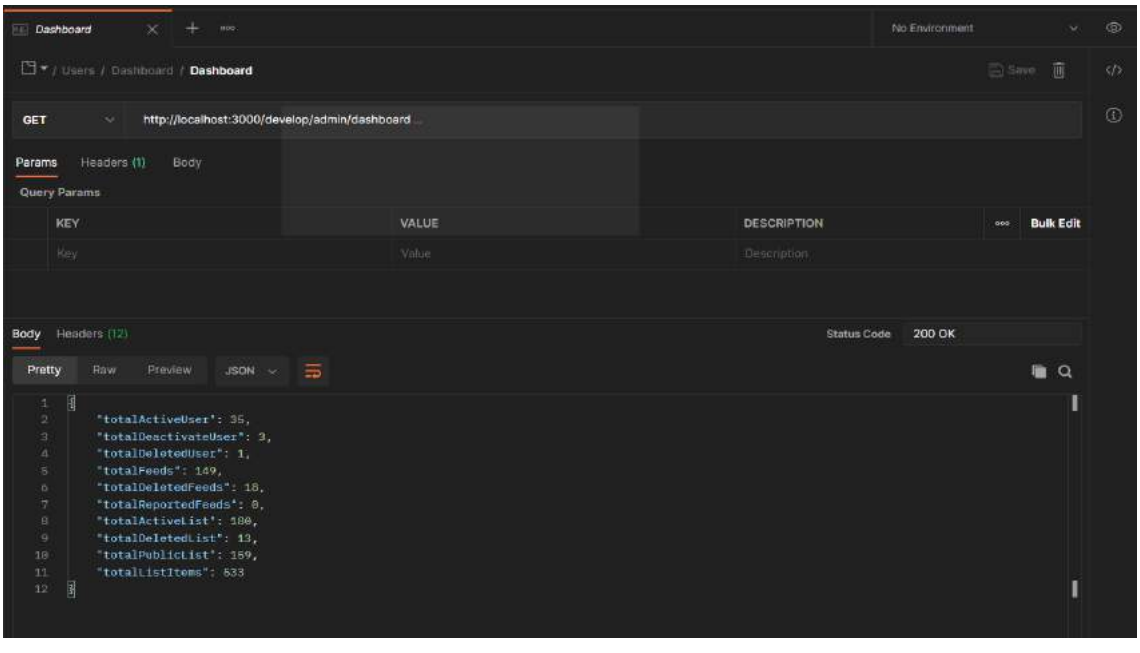
VALIDATIONS ON STUDENT



GET ALL LIST



DASHBOARD



CHAPTER 6

CODING

This chapter contains some codes of the project. The goal of the coding is to translate the design of the system into code in a given programming language. For a given design, the aim of this phase is to implement the design in the best possible manner. The coding phase affects both testing and maintenance profoundly.

Some Codes are as Written below:

```
// tslint:disable max-file-line-count
import { Response, Router } from "express";
import * as lodash from "lodash";
import * as AWS from "aws-sdk";
import * as moment from "moment-timezone";
import * as mongoose from "mongoose";
import * as referralCodeGenerator from "referral-code-generator";
import {
  createAndPushNotificationWithFormat,
  filterPagination,
  filterUserBlock,
  getContentNotification,
  parseBlock,
  parseFollowers,
  responseError,
  serializeSimpleUser,
  validateHeader,
  validateQuery,
  verifyFirebaseToken,
  DEFAULT_DATE_FORMAT,
  ErrorKey,
  HeaderAuthorizationSchema,
  IRequest,
  ISimpleUserResponse,
  IUser,
  NotificationKeyType,
  NotificationSchemaName,
```

```
  NotificationUser,
  Provider,
  ReferenceType,
  StatusCode,
  User,
  UserBlock,
  UserSchemaName,
  UserFollowing,
  FollowRequestStatus,
  LocationType,
  AccountActivity,
  ActivityType,
  updateUserFirebaseByAuthId,
  sendEmail,
  EmailTemplateType,
  // StatusSchema,
} from "../../../common";
import { Permissions } from "../../../common/models/permissions";
import {
  emptyJson,
  // parseParamFullTextSearch,
  parseProfiles,
  // getListExcludingMe,
  getMyFollowersUserIdList,
  getFollowersData,
  generatePin,
} from "../business";
import { parseMyNotification } from "../business/notification";
import { IUserFormCreate, UserFriend, UserOTP } from "../models";
import { AllowForPasswordSchema } from "../schemas";
import { PointCode, UserPoint } from "../../../points/src/models";
import {
  serializerNotificationItem,
  serializerUser,
  serializeSimpleUserForFollower,
} from "../serializers";
import console = require("console");
import { getPointsByCode } from "../../../points/src/business/points";
import { closeConnection } from "../../../common/services";
// import { appendFileSync } from "fs";
import { UserRatings } from "../models/user-rating";

let fileType = require("file-type");
```

```
export function userRouter(route: Router) {
  route
    .route("/me")
    .get(
      validateHeader(HeaderAuthorizationSchema, {
        allowUnknown: true,
      }),
      getProfile
    )
    .put(
      validateHeader(HeaderAuthorizationSchema, {
        allowUnknown: true,
      }),
      //  validateBody(UpdateUserSchema, {
      //   allowUnknown: true
      // }),
      updateProfile
    );
  route.post("/me/following/:userId", sendFollowRequest);
  route.post("/me/respondToFollowRequest/:userId", respondToFollowRequest);
  route.get("/me/following", getFollowing);
  route.get("/me/getMyfollowersRequests", getMyfollowersRequests);
  route.get("/me/getMyFollowingRequests", getMyFollowingRequests);
  route.post("/me/block/:userId", setBlock);
  route.get("/me/block", getBlock);
  route.get("/me/followers", getMyFollowers);
  route.get("/me/friends", getFriends);
  route.get("/me/notifications", getMyNotification);
  route.post("/me/markAllNotificationsRead", markAllNotificationsRead);
  route.post("/me/markThisNotificationRead/:id", markThisNotificationRead);
  route.post("/me/clearAllNotifications", clearAllNotifications);
  route.get("/me/auth-id/:authId", checkAuthId);
  route.post("/me/login", login);
  route.post("/me/logout", logout);
  route.post("/me/openApp", openApp);
  route.route("/me/settings").post(settings).get(getSettings);
  route.post("/me/deactivateAccount", deactivateAccount);

  route.post(
    "/me/reactivateAccount",
    validateHeader(HeaderAuthorizationSchema, {
      allowUnknown: true,
    }),
    reactivateAccount
```

```
  );
  route.get("/users", getUserList);
  route.get(
    "/users/allow-forgot-password",
    validateQuery(AllowForPasswordSchema),
    allowForgotPassword
  );
  route.get("/users/:id", getOtherProfile);
  route.get("/users/:username/username", getOtherProfileByUsername);
  route.post("/me/uploadProfileImage", uploadProfileImage);

  route.get("/public/users/:id", getOtherProfile);
  route.get("/public/users/:username/username", getOtherProfileByUsername);

  route.get("/users/getFollowingUsers/:userId", getOtherUsersFollowingRequests);
  route.get(
    "/users/getOtherUsersFollowersRequests/:userId",
    getOtherUsersFollowersRequests
  );
  route.post("/updatePassword", updatePassword);
  route.post("/sendOTPForEmail", sendOTPForEmail);
  route.post("/verifyOTP", verifyOTP);
  route.post(
    "/me/forgot-password",
    forgotPassword
  );
  route.post("/users/ratings/:userId",userRatings);
  route.get("/users/ratings/:userId",getUserRatings);
}

async function sendOTPForEmail(req: IRequest, res: Response) {
  const { email } = req.body;

  const otp = await generatePin();
  let now = new Date();
  const expiryMinutes = Number(process.env.OTP_EXPIRY_MINUTES);
  now.setMinutes(now.getMinutes() + expiryMinutes);

  const form = {
    email,
    otp,
    expiryDate: now,
  };
```

```
  const userOTP = await UserOTP.create(form);
  await sendEmail(
    { emailsReceive: [email] },
    {
      type: EmailTemplateType.SendOTP,
      params: {
        content: {
          otp: otp.toString(),
          expiryDate: now.toString(),
        },
      },
    }
  );
  await closeConnection();
  return res.json({
    message: "Email sent successfully",
    data: userOTP,
  });
}

async function verifyOTP(req: IRequest, res: Response) {
  const { email, otp } = req.body;
  let userOTP = await UserOTP.findOne({
    email,
    otp,
  }).sort({ createdAt: -1 });
  let verified = false;
  if (userOTP && userOTP.expiryDate > new Date()) {
    verified = true;
    userOTP.verified = true;
    userOTP.markModified("verified");
    await userOTP.save();
  }
  await closeConnection();
  return res.json({
    verified,
    data: userOTP,
  });
}

async function userRatings(req: IRequest, res: Response) {
  try {
    const userId = req.params.userId;
    const currentUserId = req.context.currentUser._id;
```

```
      const form: any = {
      userId: userId,
      ratings: req.body.ratings,
      review: req.body.review,
      createdBy: currentUserId,
    };
    let user = await User.findById(userId);
    if (!user) {
  await closeConnection();

      return responseError(req, res, ErrorKey.RecordNotFound);
    }
    let ratings = await UserRatings.findOne({userId: userId, createdBy: currentUserId,})
    if (ratings) {
      ratings = await UserRatings.findByIdAndUpdate(ratings._id, { ratings: form.ratings,
review: form.review, updatedAt: new Date() }, { new: true })
    } else {
      ratings = await UserRatings.create(form);
    }
    console.log("ratings: ", ratings)
    const ratingData = await UserRatings.find({ userId: userId });
    console.log("ratingData: ", ratingData)
    let getRating = ratingData.map(e => e.ratings);
    console.log("getRating: ", getRating)
    let sumRatings = getRating.reduce((a, b) => a + b, 0);
    console.log("sumRatings: ", sumRatings)
    let userratings = sumRatings / ratingData.length
    console.log("userratings: ", userratings)
    user = await User.findByIdAndUpdate(user._id, { ratings: userratings },{new:true})
    const result = serializerUser(user);
    await closeConnection();
    return res.json({
      data: result
    });
  } catch (error) {
    return responseError(req, res, error)
  }
}
async function getUserRatings(req: IRequest, res: Response) {
      try {
        const userId = req.params.userId;
        let user = await User.findById(userId);
        if (!user) {
```

```
        await closeConnection();
        return responseError(req, res, ErrorKey.RecordNotFound);
      }
      let modelFilter: any = {
        status: StatusCode.Active,
        userId: mongoose.Types.ObjectId(userId)
      }
      const models = await filterPagination(UserRatings, modelFilter, {
        ...req.query,
        sort: { createdAt: -1 },
        // buildQuery: (query, limit, skip) => {
        //   let queryBase = query.aggregate([
        //     { $match: modelFilter },
        //     { $sort: { createdAt: -1 } },
        //   ]);
        //   return queryBase.skip(skip).limit(limit);
        // }
      });
      console.log("model is :  ", models);
      console.log("models data: ", models.data);
      await closeConnection();
      return res.json({
        data: models.data,
        pagination: models.pagination,});
    } catch (error) {
      return responseError(req, res, error)
    }
  }

async function forgotPassword(req: IRequest, res: Response) {
  try {
    const { email, password, confirm } = req.body;

    if (!password || !confirm) {
      console.log("Does not have correct body");
  await closeConnection();

      return responseError(req, res, ErrorKey.FailedPasswordValidation);
    } else if (password.length < 6 || password != confirm) {
  await closeConnection();

      return responseError(req, res, ErrorKey.FailedPasswordValidation);
    }
```

```
    const user = await User.findOne({
      email,
      status: StatusCode.Active,
    });
    await closeConnection();
    if (!user) {
  await closeConnection();

      return responseError(req, res, ErrorKey.EmailNotFound);
    }

    try {
      var firebaseUser = await updateUserFirebaseByAuthId(user.authId, {
        password: password,
      });
      console.log(firebaseUser);
    } catch (e) {
      console.log("Firebase error");
      console.log(e);
      await closeConnection();
      return responseError(req, res, ErrorKey.ErrorUnknown);
    }
    await closeConnection();

    return res.json({
      message: "Password changed successfully",
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function updatePassword(req: IRequest, res: Response) {
  if (!req.query.hasOwnProperty("token")) {
  await closeConnection();

    return responseError(req, res, ErrorKey.Unauthorized);
  }
  if (
    !req.body.hasOwnProperty("password") ||
    !req.body.hasOwnProperty("confirm")
  ) {
    console.log("Does not have correct body");
    await closeConnection();
```

```
    return responseError(req, res, "Does not have correct body");
  } else if (
    req.body.password.length < 6 ||
    req.body.password != req.body.confirm
  ) {
    await closeConnection();
    return responseError(req, res, "Incorrect length");
  }
  console.log(req.query);
  // var user = await User.findOne({resetPasswordToken: req.query.token,
resetPasswordExpires: {$gt: Date.now()}})
  var user = await User.findById(req.query.token);
  if (user == null) {
    console.log("Token not valid");
    await closeConnection();
    return responseError(req, res, ErrorKey.Unauthorized);
  }

  // Update password
  var password = req.body.password;

  try {
    var firebaseUser = await updateUserFirebaseByAuthId(user.authId, {
      password: password,
    });

    console.log(firebaseUser);
  } catch (e) {
    console.log("Firebase error");
    console.log(e);
    await closeConnection();
    return responseError(req, res, ErrorKey.ErrorUnknown);
  }
  console.log(req.body);
  await closeConnection();
  return res.json({
    message: "Your password has been updated, you can now login.",
  });
}

async function login(req: IRequest, res: Response) {
  const currentUserId = req.context.currentUser._id;
  console.log("Current user is: ", currentUserId);
  var user = await User.findById(currentUserId);
```

```
 if (user.status = StatusCode.Deactivated) {
  user = await User.findByIdAndUpdate(currentUserId, {status: StatusCode.Active},
{new:true})
 }
 const adminUser = await User.findOne({
  status: StatusCode.Active,
  permissions: Permissions.Admin,
 });

 const lastLoggedIn = await AccountActivity.findOne({
  activityType: ActivityType.LOGIN,
  affectedUser: currentUserId,
  createdBy: currentUserId,
 }).sort({ createdAt: -1 });

 if (lastLoggedIn) {
  const lastLoggedInDate = moment(lastLoggedIn.createdAt);
  var today = moment();
  var diff = today.diff(lastLoggedInDate, "days");
  if (diff <= 1) {
   const numPoints = await getPointsByCode(PointCode.OPEN_NOTCH);
   if (user.settings && user.settings.points === true) {
    const userPoints = {
     description: "User logged in consecutively",
     code: PointCode.LOGIN_CONSECUTIVELY,
     numPoints,
     userId: currentUserId,
    };
    await UserPoint.create(userPoints);
    const notify = await createAndPushNotificationWithFormat(
                                        null,
                                        {
                                                ...getContentNotification(

        NotificationKeyType.EarnedPointsForConsecutiveLogin
                                                ),
                                                isFormat: false,
                                                senderId: adminUser._id,
                                                receiverIds: [currentUserId],
                                                type: ReferenceType.User,
                                                referenceId: currentUserId,
                                        },
                                        {
                                                actorId: adminUser._id,
```

```
                                        }
                                    );
        console.log("Sent Notification: ", notify);
        let rewardPoints = 0;
        if (user.rewardPoints) {
          rewardPoints = user.rewardPoints;
        }
        user.rewardPoints = rewardPoints + numPoints;
        await user.save();
      }
    }
  }
  const activityForm = {
    activityType: ActivityType.LOGIN,
    affectedUser: currentUserId,
    createdBy: currentUserId,
  };
  await AccountActivity.create(activityForm);

  const [result]:any = await parseProfiles([user]);
  const serializedUser = serializerUser(result);
  await closeConnection();
  return res.json({
    message: "User logged in succesfully",
    data: serializedUser,
  });
}

async function openApp(req: IRequest, res: Response) {
  try {
    const currentUserId = req.context.currentUser._id;
    console.log("Current user is: ", currentUserId);
    var user = await User.findById(currentUserId);
    const adminUser = await User.findOne({
      status: StatusCode.Active,
      permissions: Permissions.Admin,
    });
    const activityForm = {
      activityType: ActivityType.App_Open,
      affectedUser: currentUserId,
      createdBy: currentUserId,
    };
    await AccountActivity.create(activityForm);
    const numPoints = await getPointsByCode(PointCode.OPEN_NOTCH);
```

```
if (user.settings && user.settings.points === true) {
  const userPoints = {
    description: "User opened app",
    code: PointCode.OPEN_NOTCH,
    numPoints,
    userId: currentUserId,
  };
  await UserPoint.create(userPoints);
  const notify = await createAndPushNotificationWithFormat(
                                    null,
                                    {
                                            ...getContentNotification(

        NotificationKeyType.EarnedPointsForOpeningApp
                                            ),
                                            isFormat: false,
                                            senderId: adminUser._id,
                                            receiverIds: [currentUserId],
                                            type: ReferenceType.User,
                                            referenceId: currentUserId,
                                    },
                                    {
                                            actorId: adminUser._id,
                                    }
                            );
  console.log("Sent Notification: ", notify);
  let rewardPoints = 0;
  if (user.rewardPoints) {
    rewardPoints = user.rewardPoints;
  }
  user.rewardPoints = rewardPoints + numPoints;
  await user.save();
}
let rewardPoints = 0;
if (user.rewardPoints) {
  rewardPoints = user.rewardPoints;
}
user.rewardPoints = rewardPoints + numPoints;
await user.save();

const [result]:any = await parseProfiles([user]);
const serializedUser = serializerUser(result);
await closeConnection();
return res.json({
```

```
      message: "User opened app succesfully",
      data: serializedUser,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function logout(req: IRequest, res: Response) {
  const currentUserId = req.context.currentUser._id;
  console.log("Current user is: ", currentUserId);
  var user = await User.findById(currentUserId);
  const activityForm = {
    activityType: ActivityType.LOGOUT,
    affectedUser: currentUserId,
    createdBy: currentUserId,
  };
  await AccountActivity.create(activityForm);
  const [result]:any = await parseProfiles([user]);
  const serializedUser = serializerUser(result);
  await closeConnection();
  return res.json({
    message: "User logged out succesfully",
    data: serializedUser,
  });
}

export async function deactivateAccount(req: IRequest, res: Response) {
  try {
    const currentUserId = req.context.currentUser._id;
    let user = await User.findByIdAndUpdate(
      currentUserId,
      {
        status: StatusCode.Deactivated,
      },
      { new: true }
    );
    const activityForm = {
      activityType: ActivityType.DEACTIVATE,
      affectedUser: currentUserId,
      createdBy: currentUserId,
    };
    await AccountActivity.create(activityForm);
    await closeConnection();
```

```
    return res.json({
      message: "Deactivated user successfully",
      data: user,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

export async function reactivateAccount(req: IRequest, res: Response) {
  try {
    // const currentUserId = req.context.currentUser._id;
    const token = req.headers.authorization;
    console.log("Token passed is: ", token);
    const authId = await verifyFirebaseToken(token);
    console.log("Auth Id is: ", authId);
    let user = await User.findOne({
      authId,
    });
    if (!user) {
      await closeConnection();
      return responseError(req, res, ErrorKey.ProfileNotFound, {
        statusCode: 404,
      });
    }
    user = await User.findByIdAndUpdate(
      user._id,
      {
        status: StatusCode.Active,
      },
      { new: true }
    );
    const activityForm = {
      activityType: ActivityType.REACTIVATE,
      affectedUser: user._id,
      createdBy: user._id,
    };
    await AccountActivity.create(activityForm);
    await closeConnection();
    return res.json({
      message: "Reactivated user successfully",
      data: user,
    });
  } catch (error) {
```

```
    return responseError(req, res, error);
  }
}

export async function settings(req: IRequest, res: Response) {
  try {
    let currentUserId = req.context.currentUser._id;
    let settings = req.body;
    let currentUser = await User.findById(currentUserId);
    let userSettings = currentUser.settings;

    console.log("Setting: " + JSON.stringify(settings));
    console.log("User settings: " + JSON.stringify(userSettings));
    let gg = lodash.merge(userSettings, settings);
    console.log("gg: " + gg);
    console.log("user settings now: " + JSON.stringify(userSettings));

    currentUser.settings = userSettings;
    currentUser = await User.findByIdAndUpdate(
      currentUser._id,
      {
        settings,
      },
      { new: true }
    );
    // let setting = await User.findByIdAndUpdate(
    //   currentUserId,
    //   { settings: userSettings },
    //   { new: true }
    // );
    const [result]:any = await parseProfiles([currentUser]);
    const serializedUser = serializerUser(result);
    await closeConnection();
    return res.json(serializedUser);
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getSettings(req: IRequest, res: Response) {
  try {
    let currentUserId = req.context.currentUser._id;
    let user = await User.findById(currentUserId);
    let settings = user.settings;
```

```
    if (settings == null) {
      settings = {
        globalSocialFeed: false,
        friendSocialFeed: false,
        points: false,
        locationPublic: false,
      };
    }
    await closeConnection();
    return res.json({
      data: settings,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function uploadProfileImage(req, res: Response) {
  try {
    AWS.config.setPromisesDependency(require("bluebird"));
    AWS.config.update({
      accessKeyId: "AKIA4DD2NA3KCXSWRRR7",
      secretAccessKey: "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",
      region: "us-east-1",
    });
    const s3 = new AWS.S3();

    let base64String = req.body.base64String;
    console.log("base64String: ", base64String);

    // pass the base64 string into a buffer
    let buffer = new Buffer(base64String, "base64");

    let fileMime = await fileType.fromBuffer(buffer);
    console.log("fileMime: " + JSON.stringify(fileMime));

    // check if the base64 encoded string is a file
    if (fileMime === null) {
      console.log("The string supplied is not a file type");
    }

    let file = getFile(fileMime, buffer);
    let params = file.params;
    console.log("params: ", params);
```

```
    let location = "";
    let key = "";
    const { Location, Key } = await s3.upload(params).promise();
    location = Location;
    key = Key;
    console.log(location, key);

    return res.json({
      filePath: location,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

function getFile(fileMime, buffer) {
  // get the file extension
  let fileExt = fileMime.ext;
  //  let now = moment().format('YYYY-MM-DD HH:mm:ss');

  let fileName = Date.now() + "." + fileExt;
  let bucketName = "notch-dev-profile-images";
  // if(process.env.ENV_NAME === 'production') {
  //    bucketName = "surfmadly-prod-images";
  // }
  let params = {
    Bucket: bucketName,
    Key: "images/" + fileName,
    // 'this is simply the filename and the extension, e.g fileFullName + fileExt',
    Body: buffer,
    ContentType: fileMime.mime,
    ContentEncoding: "base64",
  };
  let uploadFile = {
    size: buffer.toString("ascii").length,
    type: fileMime.mime,
    name: fileName,
    //  full_path: fileFullPath
  };

  return {
    params: params,
    uploadFile: uploadFile,
```

```
  };
}

export async function getProfile(req: IRequest, res: Response) {
  try {
    const token = req.headers.authorization;
    console.log("Token passed is: ", token);
    const authId = await verifyFirebaseToken(token);
    console.log("Auth Id is: ", authId);
    const user = await User.findOne({
      authId,
      status: StatusCode.Active,
    });
    const deactivatedUser = await User.findOne({
      authId,
      status: StatusCode.Deactivated,
    });
    if (deactivatedUser) {
      return responseError(req, res, ErrorKey.UserDeactivated)

    }
    if (!user) {
      await closeConnection();
      return responseError(req, res, ErrorKey.ProfileNotFound, {
        statusCode: 404,
      });
    }
    const [result]:any = await parseProfiles([user]);
    const serializedUser = serializerUser(result);
    await closeConnection();
    return res.json(serializedUser);
  } catch (error) {
    console.log("Error Get Profile", error);
    await closeConnection();
    return responseError(req, res, ErrorKey.Unauthorized, {
      statusCode: 401,
    });
  }
}

async function updateProfile(req: IRequest, res: Response) {
  try {
    const token = req.headers.authorization;
    const form: IUserFormCreate = req.body;
```

```
    const adminUser = await User.findOne({
                            status: StatusCode.Active,
                            permissions: Permissions.Admin,
                    });
    delete form.permissions;
    if (
      req.body.hasOwnProperty("latitude") &&
      req.body.hasOwnProperty("longitude")
    ) {
      console.log("Setting location field");
      const locationForm = {
        type: LocationType.POINT,
        coordinates: [req.body.longitude, req.body.latitude],
      };
      form.location = locationForm;
      console.log("Form: ", form);
    }
    const authId = await verifyFirebaseToken(token);
    const user = await User.findOne({ authId });
    let modelUpdated;
    if (user) {
      if (form.username && form.username !== user.username) {

        form.username = form.username.toLocaleLowerCase();
        const [checkUsername] =
                                    await Promise.all([
                                        User.findOne({
                                            username: { $regex: `^${form.username}$`, $options: "i" },
                                            status: StatusCode.Active,
                                        })
                                    ]);
                                if (checkUsername) {
                                    await closeConnection();
                                    return responseError(req, res, ErrorKey.UsernameExisted);
                                }
      }
      if (!form.username) {
        var str = form.email;
                                var nameReplace = str.replace(/@.*$/, "");
                                form.username = nameReplace !== str ? nameReplace : null;
      }
```

```
    modelUpdated = await User.findByIdAndUpdate(
                                    user._id,
                                    lodash.omit(form, [
                                            "email",
                                            "myReferralCode",
                                            "friendReferralCode",
                                    ]),
                                    {
                                            new: true,
                                    }
                            );
   } else {
    form.authId = authId;
    form.provider = form.provider || Provider.Email;
    if ((form.provider === Provider.Facebook || form.provider === Provider.Google) ||
!form.username) {

      var str = form.email;
      var nameReplace = str.replace(/@.*$/, "");
     form.username = nameReplace !== str ? nameReplace : null;
     const userexisted = await User.findOne({
                                    username: form.username ,
     });
     if (userexisted) { form.username = form.email; }
    }
    if (form.username) {
     form.username = form.username.toLocaleLowerCase();
    }
    const [checkEmail, checkUsername, checkAuthIdExisted] = await Promise.all(
     [
      <any>User.findOne({
       email: form.email,
       status: StatusCode.Active,
      }),
      User.findOne({
       username: { $regex: `^${form.username}$`, $options: "i" },
       status: StatusCode.Active,
      }),
      User.findOne({
       authId: form.authId,
       status: StatusCode.Active,
      }),
     ]
```

```
    );
    if (checkEmail) {
      await closeConnection();
      return responseError(req, res, ErrorKey.EmailExisted);
    }
    if (checkUsername) {
      await closeConnection();
      return responseError(req, res, ErrorKey.UsernameExisted);
    }
    if (checkAuthIdExisted) {
      await closeConnection();
      return responseError(req, res, ErrorKey.AuthExisted);
    }

    try {
        let referralCode = referralCodeGenerator.alpha("lowercase", 12);
      form.myReferralCode = referralCode;

      console.log("Creating user");
      modelUpdated = await User.create(form);
      console.log("user created");
      let currentUserId = modelUpdated._id;
      let referredUser = await User.findOne({
                                    myReferralCode: form.friendReferralCode,
      });
      console.log("user is", JSON.stringify(referredUser));
                                if (referredUser) {
                                    const numPoints = await
getPointsByCode(PointCode.INVITE_FRIENDS);
                                    if (referredUser.settings &&
referredUser.settings.points === true) {
                                        const userPoints = {
                                            description: "User invites friend",
                                            code:
PointCode.INVITE_FRIENDS,
                                            numPoints,
                                            userId: referredUser._id,
                                        };
                                        await UserPoint.create(userPoints);
                                        const notify = await
createAndPushNotificationWithFormat(
                                            null,
                                            {
                                                ...getContentNotification(
```

```
            NotificationKeyType.EarnedPointsForInvitingAFriend
                                                        ),
                                                        isFormat: false,
                                                        senderId: adminUser._id,
                                                        receiverIds:
[referredUser._id],
                                                        type: ReferenceType.User,
                                                        referenceId: currentUserId,
                                                    },
                                                    {
                                                        actorId: adminUser._id,
                                                    }
                                                );
                                                console.log("Sent Notification: ", notify);
                                                let rewardPoints = 0;
                                                if (referredUser.rewardPoints) {
                                                    rewardPoints                        =
referredUser.rewardPoints;
                                                }
                                                referredUser.rewardPoints  =  rewardPoints
+ numPoints;
                                                await referredUser.save();
                                            }
                                        }
        // await followDefaultAccounts(modelUpdated);
        // console.log("Followed default accounts");
        // await addDefaultPluggers(modelUpdated);
        // console.log("Added default pluggers");
        // await addDefaultFriends(modelUpdated);
        // console.log("Added default friends");

        // const userAdmins = (await User.find({ permissions: Permissions.Admin, status:
StatusCode.Active }))
        //   .map((e) => e.email);
        // const userAdmins = process.env.EMAIL;
        // if (userAdmins) {
        //   await sendEmail({ emailsReceive: [userAdmins] }, {
        //     type: EmailTemplateType.NewUser,
        //     params: {
        //       content: {
        //         username: modelUpdated.username,
        //         createdAt: moment(modelUpdated.createdAt).tz(DEFAULT_TIMEZONE)
        //           .format(DEFAULT_DATE_TIME_FORMAT)
```

```
      //       }
      //     }
      //   });
      // }
     } catch (error) {
      if (error && error.code === 11000 && /username/g.test(error.errmsg)) {
       await closeConnection();
       return responseError(req, res, ErrorKey.UsernameExisted, {
        statusCode: 404,
       });
      }

      if (error && error.code === 11000 && /email/g.test(error.errmsg)) {
       await closeConnection();
       return responseError(req, res, ErrorKey.EmailExisted, {
        statusCode: 404,
       });
      }
      await closeConnection();
      return responseError(req, res, error);
     }
    }
    const [result]:any = await parseProfiles([modelUpdated]);
    const serializedUser = serializerUser(result);
    await closeConnection();

    return res.json(serializedUser);
   } catch (error) {
    return responseError(req, res, error);
   }
}

async function sendFollowRequest(req: IRequest, res: Response) {
  try {
    const friendId = req.params.userId;

    // let authId = 'JcDFe8ou6qY9JzGGToHaudVrD973';
    // let currentUser = await User.findOne({authId});
    // const currentUserId = currentUser._id;

    const friendUser = await User.findById(friendId);
    const currentUserId = req.context.currentUser._id;
    console.log("Current user: " + currentUserId);
    // const currentUser = await User.findById(currentUserId);
```

```
if (friendId.toString() === currentUserId.toString()) {
  await closeConnection();
  return responseError(req, res, ErrorKey.NotFollowYourSelf);
}
if ((await filterUserBlock(currentUserId, [friendId])).length === 0) {
  await closeConnection();
  return responseError(req, res, ErrorKey.FollowBockUser);
}

let message;
let followRequestSent;

const followForm: any = {
  userId: currentUserId,
  followingId: friendUser._id,
  requestStatus: FollowRequestStatus.Accepted,
  createdBy: currentUserId,
};
let userFollowing = await UserFollowing.findOne(followForm);
if (userFollowing) {
  userFollowing.requestStatus = FollowRequestStatus.Unfollow;
  userFollowing.updatedBy = currentUserId;
  await userFollowing.save();
  const formFriend = {
    friendIds: [friendUser._id, currentUserId],
  };
  let userFriend = await UserFriend.findOne(formFriend);
  if (userFriend) {
    await UserFriend.findByIdAndDelete(userFriend._id);
  }

  followRequestSent = false;
  message = "Unfollowed user successfully!!!";
} else {
  followForm.requestSendDate = new Date();
  await UserFollowing.create(followForm);
  message = "Follow request sent successfully!!!";
  const notify = await createAndPushNotificationWithFormat(
    null,
    {
      ...getContentNotification(NotificationKeyType.StartedFollowing),
      senderId: currentUserId,
      receiverIds: [friendId],
```

```
          type: ReferenceType.User,
          referenceId: currentUserId,
        },
        {
          actorId: currentUserId,
        }
      );
      console.log("Sent Notification: ", notify);
      followRequestSent = true;
    }

    let parsedFriend : any = await parseProfiles([friendUser]);
    console.log("parsedFriend: ", parsedFriend)
    parsedFriend[0].followRequestSent = followRequestSent;
    parsedFriend[0].isFollowing = true;
    // parsedFriend.isMyFriend = friendIdList.length > 0 &&
friendIdList.includes(data._id.toString());
    console.log("Data: " + JSON.stringify(parsedFriend));
    const serializedFriend = parsedFriend.map(serializerUser)
    await closeConnection();
    return res.json({
                        message,
                        data: serializedFriend,
                    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function respondToFollowRequest(req: IRequest, res: Response) {
  try {
    const friendId = req.params.userId;
    const friendResponse = req.query.friendResponse;
    const currentUserId = req.context.currentUser._id;

    var friendRequestStatus = "";
    // const currentUser = await User.findById(currentUserId);
    const form = {
      userId: friendId,
      followingId: currentUserId,
      requestStatus: FollowRequestStatus.Sent,
    };
    let parsedFriend;
    const friendRequest = await UserFollowing.findOne(form);
```

```
    let friend = await User.findById(friendId);

    parsedFriend = await parseProfiles([friend]);
    console.log("Parsed Friend: ", parsedFriend);
    if (friendRequest != null) {
      if (friendResponse === FollowRequestStatus.Accepted) {
        friendRequest.requestStatus = FollowRequestStatus.Accepted;
        friendRequest.requestApprovedDate = new Date();
        friendRequest.updatedBy = currentUserId;
        let updatedRequest = await friendRequest.save();
        //          const          updatedRequest          =          await
UserFollowing.findByIdAndUpdate(friendRequest._id, {
        //   requestStatus: FollowRequestStatus.Accepted,
        //   requestApprovedDate: new Date(),
        //   updatedBy: currentUserId
        // }, {new: true});
        console.log("Updated Request: ", updatedRequest);
        friendRequestStatus = FollowRequestStatus.Accepted;
        const formFriend = {
          friendIds: [friendId, currentUserId],
        };

        if (friend.status === StatusCode.Active) {
          if (friendResponse === FollowRequestStatus.Accepted) {
            await UserFriend.create(formFriend);
          }

          // const currentUser = await User.findById(currentUserId);
          parsedFriend.isMyFriend = true;
          const notify = await createAndPushNotificationWithFormat(
            null,
            {
              ...getContentNotification(
                NotificationKeyType.AcceptFollowingRequest
              ),
              senderId: currentUserId,
              receiverIds: [friend._id],
              type: ReferenceType.User,
              referenceId: currentUserId,
            },
            {
              actorId: currentUserId,
            }
          );
```

```
        console.log("Notification sent is: ", notify);
      }
    } else if (friendResponse === FollowRequestStatus.Declined) {
      // const updatedRequest = await ProfileFriend.findByIdAndUpdate(friendRequest._id, {
      //   requestStatus: FriendRequestStatus.Declined,
      //   requestApprovedDate: new Date(),
      //   updatedBy: currentUserId
      // }, {new: true});
      friendRequest.requestStatus = FollowRequestStatus.Declined;
      friendRequest.requestApprovedDate = new Date();
      friendRequest.updatedBy = currentUserId;
      let updatedRequest = await friendRequest.save();
      console.log("Updated Request: ", updatedRequest);
      // friendRequestStatus = FriendRequestStatus.Declined;
      parsedFriend.isMyFriend = false;
      const notify = await createAndPushNotificationWithFormat(
        null,
        {
          ...getContentNotification(
            NotificationKeyType.RejectFollowingRequest
          ),
          senderId: currentUserId,
          receiverIds: [friend._id],
          type: ReferenceType.User,
          referenceId: currentUserId,
        },
        {
          actorId: currentUserId,
        }
      );
      console.log("Notification sent is: ", notify);
    }
    if (req.body.hasOwnProperty("notificationId")) {
      const notificationId = req.body.notificationId;
      let notification = await NotificationUser.findOne({
        notificationId: notificationId,
        userId: currentUserId,
      });
      notification = await NotificationUser.findByIdAndUpdate(
        notification._id,
        { isRead: true },
        { new: true }
      );
```

```
        console.log("Notification updated: ", notification);
      }
    } else {
      await closeConnection();
      return responseError(req, res, ErrorKey.InvalidFollowingRequest);
    }
    await closeConnection();
    return res.json({
      message: "Following Request " + friendRequestStatus + " Successfully!!!!",
      data: parsedFriend,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getMyFollowers(req: IRequest, res: Response) {
  try {
    const currentUserId = req.context.currentUser._id;
    // const userId = req.params.userId;
    let user = await User.findById(currentUserId);
    if (user == null) {
      await closeConnection();
      return responseError(req, res, ErrorKey.ProfileNotFound);
    }

    const models:any = await getFollowersData(currentUserId, req);
    console.log("Users: ", models);

    if (models.data.length > 0) {
      // const friendIdList = await getFriendIdStringList(currentUserId);
      // const mySentRequests = await mySentFriendRequests(currentUserId);
      models.data = await parseProfiles(models.data);
      models.data = await parseFollowers(currentUserId, models.data)
      models.data = await parseBlock(currentUserId, models.data);
      // for(var k = 0; k< models.data.length; k++) {
      //   var data = models.data[k];
      //       // data.isMyFriend = friendIdList.length > 0 &&
friendIdList.includes(data._id.toString());
      //       // data.friendRequestSent = mySentRequests.length > 0 &&
mySentRequests.includes(data._id.toString());
      // }
      // models.data = await parseUserProfiles(models.data);
    }
```

```
    const serializedUser = models.data.map(serializerUser);
    await closeConnection();
    return res.json({
      data: serializedUser,
      pagination: models.pagination,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getOtherUsersFollowersRequests(req: IRequest, res: Response) {
  try {
    let currentUserId = req.context.currentUser._id;
    let userId = req.params.userId;
    let modelFilter: any = {
      followingId: mongoose.Types.ObjectId(userId),
      requestStatus: {
        $in: [FollowRequestStatus.Accepted],
      },
    };
    let reqQuery = req.query;
    const models = await filterPagination(UserFollowing, modelFilter, {
      ...req.query,
      sort: { createdAt: -1 },
      buildQuery: (query, limit, skip) => {
        let queryBase;
        const lookupUser = {
          $lookup: {
            from: UserSchemaName,
            localField: "createdBy",
            foreignField: "_id",
            as: "following",
          },
        };
        const matchUser = {
          $match: {
            "following.status": StatusCode.Active,
          },
        };

        if (reqQuery != null && reqQuery.name != null) {
          let searchString:any = reqQuery.name;
          let matchQuery = {
```

```
        $match: {
          $or: [
            {
              "following.displayName": new RegExp(searchString, "i"),
            },
            {
              "following.username": new RegExp(searchString, "i"),
            },
            {
              "following.lastName": new RegExp(searchString, "i"),
            },
            {
              "following.firstName": new RegExp(searchString, "i"),
            },
          ],
        },
      };
      queryBase = query.aggregate([
        { $match: modelFilter },
        lookupUser,
        matchUser,
        matchQuery,
        { $sort: { createdAt: -1 } },
      ]);
    } else {
      queryBase = query.aggregate([
        { $match: modelFilter },
        lookupUser,
        matchUser,
        { $sort: { createdAt: -1 } },
      ]);
    }

    return queryBase.skip(skip).limit(limit);
  },
});

console.log("models data: " + JSON.stringify(models.data));
let users = [];

if (models.data != null) {
  let followUsersIds = models.data.map((e) => e.userId);
  console.log("followUsersIds: " + JSON.stringify(followUsersIds));
  let searchFilter: any = {
```

```
        userId: mongoose.Types.ObjectId(currentUserId),
        createdBy: mongoose.Types.ObjectId(currentUserId),
        followingId: { $in: followUsersIds },
        status: StatusCode.Active,
      };
      let myFollowings = await UserFollowing.find(searchFilter);
      console.log("my followings: " + JSON.stringify(myFollowings));
      for (let i = 0; i < models.data.length; i++) {
        let user = models.data[i];
        console.log("user: " + JSON.stringify(user));
        let following = user.following ? user.following[0] : null;
        if (following != null) {
          let userData = following.toJSON ? following.toJSON() : following;
          let myFollowingData = myFollowings.filter(
            (f) => f.followingId.toString() === user.following[0]._id.toString()
          );
          console.log("Following data: " + JSON.stringify(myFollowingData));
          userData.isFollowing =
            myFollowingData && myFollowingData.length > 0
              ? myFollowingData[0].requestStatus ===
                FollowRequestStatus.Accepted
                ? true
                : false
              : false;
          userData.followRequestStatus =
            myFollowingData && myFollowingData.length > 0
              ? myFollowingData[0].requestStatus
              : null;
          users.push(serializeSimpleUser(userData));
        }
      }
    }
    await closeConnection();
    return res.json({
      data: users,
      pagination: models.pagination,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getOtherUsersFollowingRequests(req: IRequest, res: Response) {
  try {
```

```
let currentUserId = req.context.currentUser._id;
let modelFilter: any = {
  userId: mongoose.Types.ObjectId(req.params.userId),
  createdBy: mongoose.Types.ObjectId(req.params.userId),
  requestStatus: {
    $in: [FollowRequestStatus.Accepted],
  },
};

let reqQuery = req.query;
const models = await filterPagination(UserFollowing, modelFilter, {
  ...req.query,
  sort: { createdAt: -1 },
  buildQuery: (query, limit, skip) => {
    let queryBase;
    const lookupUser = {
      $lookup: {
        from: UserSchemaName,
        localField: "followingId",
        foreignField: "_id",
        as: "following",
      },
    };
    const matchUser = {
      $match: {
        "following.status": StatusCode.Active,
      },
    };

    if (reqQuery != null && reqQuery.name != null) {
      let searchString:any = reqQuery.name;
      let matchQuery = {
        $match: {
          $or: [
            {
              "following.displayName": new RegExp(searchString, "i"),
            },
            {
              "following.username": new RegExp(searchString, "i"),
            },
            {
              "following.lastName": new RegExp(searchString, "i"),
            },
            {
```

```
            "following.firstName": new RegExp(searchString, "i"),
          },
        ],
      },
    };
    queryBase = query.aggregate([
      { $match: modelFilter },
      lookupUser,
      matchUser,
      matchQuery,
      { $sort: { createdAt: -1 } },
    ]);
  } else {
    queryBase = query.aggregate([
      { $match: modelFilter },
      lookupUser,
      matchUser,
      { $sort: { createdAt: -1 } },
    ]);
  }

  return queryBase.skip(skip).limit(limit);
 },
});

let users = [];
if (models.data != null) {
 let followUsersIds = models.data.map((e) => e.followingId);
 console.log("followUsersIds: " + JSON.stringify(followUsersIds));
 let searchFilter: any = {
  userId: mongoose.Types.ObjectId(currentUserId),
  createdBy: mongoose.Types.ObjectId(currentUserId),
  followingId: { $in: followUsersIds },
  status: StatusCode.Active,
 };
 let myFollowings = await UserFollowing.find(searchFilter);
 console.log("my followings: " + JSON.stringify(myFollowings));
 for (let i = 0; i < models.data.length; i++) {
  let user = models.data[i];
  console.log("user: " + JSON.stringify(user));
  let following = user.following ? user.following[0] : null;
  if (following != null) {
   let userData = following.toJSON ? following.toJSON() : following;
   let myFollowingData = myFollowings.filter(
```

```
          (f) => f.followingId.toString() === user.following[0]._id.toString()
        );
        console.log("Following data: " + JSON.stringify(myFollowingData));
        userData.isFollowing =
          myFollowingData && myFollowingData.length > 0
            ? myFollowingData[0].requestStatus ===
              FollowRequestStatus.Accepted
              ? true
              : false
            : false;
        userData.followRequestStatus =
          myFollowingData && myFollowingData.length > 0
            ? myFollowingData[0].requestStatus
            : null;
        users.push(serializeSimpleUser(userData));
      }
    }
  }
  await closeConnection();
  return res.json({
    data: users,
    pagination: models.pagination,
  });
 } catch (error) {
  return responseError(req, res, error);
 }
}

export async function getFollowersIdStringList(currentUserId: String) {
 try {
  const myFriends = await getMyFollowersUserIdList(currentUserId);
  var friendIdList = [];
  if (myFriends != null && myFriends.length > 0) {
    for (var i = 0; i < myFriends.length; i++) {
      var myFriendId = myFriends[i];
      friendIdList.push(myFriendId.toString());
    }
  }
  await closeConnection();
  return friendIdList;
 } catch (error) {
  return Promise.reject(error);
 }
}
```

```
export async function mySentFriendRequests(currentUserId: String) {
  try {
    let myForm: any = {
      status: StatusCode.Active,
      userId: currentUserId,
      createdBy: currentUserId,
    };
    const mySentRequests = await UserFollowing.find(myForm);
    var mySentRequestsList = [];
    if (mySentRequests != null && mySentRequests.length > 0) {
      for (var i = 0; i < mySentRequests.length; i++) {
        var sentRequest = mySentRequests[i];
        mySentRequestsList.push(sentRequest.followingId.toString());
      }
    }
    console.log("mySentRequestsList", mySentRequestsList);
    await closeConnection();
    return mySentRequestsList;
  } catch (error) {
    return Promise.reject(error);
  }
}

export async function setFollowing(req: IRequest, res: Response) {
  try {
    let userId = req.params.userId;
    let currentUserId = req.context.currentUser._id;
    if (req.query.reverse) {
      userId = req.context.currentUser._id;
      currentUserId = req.params.userId;
    }
    if (userId.toString() === currentUserId.toString()) {
      await closeConnection();
      return responseError(req, res, ErrorKey.NotFollowYourSelf);
    }
    if ((await filterUserBlock(currentUserId, [userId])).length === 0) {
      await closeConnection();
      return responseError(req, res, ErrorKey.FollowBockUser);
    }

    // const userId = req.context.currentUser._id;
    // const currentUserId = req.params.userId;
```

```
const form = {
  createdBy: currentUserId,
  followingId: userId,
};
const formFriend = {
  friendIds: [userId, currentUserId],
};
let message = "Following";

await UserFollowing.findOne(form).then((r) => {
  if (r) {
    message = "Unfollow";

    return UserFriend.deleteOne({
      friendIds: {
        $all: formFriend.friendIds,
      },
    }).then(() => UserFollowing.deleteOne(form));
  }

  return UserFollowing.create(form).then(async () => {
    await createAndPushNotificationWithFormat(
      null,
      {
        ...getContentNotification(NotificationKeyType.StartFollowing),
        senderId: currentUserId,
        receiverIds: [userId],
        referenceId: currentUserId,
        type: ReferenceType.User,
      },
      { actorId: currentUserId }
    );

    return UserFollowing.findOne({
      createdBy: form.followingId,
      followingId: form.createdBy,
    }).then(async (u) => {
      return u
        ? UserFriend.create(formFriend)
        : UserFollowing.deleteOne(formFriend);
    });
  });
});
await closeConnection();
```

```
    return res.json({
      message: message + " successfully",
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getFriends(req: IRequest, res: Response) {
  try {
    const currentUserId = req.context.currentUser._id;
    const friends = await UserFriend.find({ friendIds: currentUserId });
    const ids = friends
      .reduce((pre, cur) => {
        return pre.concat(cur.friendIds);
      }, [])
      .filter((e) => e.toString() !== currentUserId.toString());

    const users = (
      await User.find({ _id: { $in: ids }, status: StatusCode.Active })
    ).map(serializeSimpleUser);
    await closeConnection();
    return res.json(users);
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getMyfollowersRequests(req: IRequest, res: Response) {
  try {
    let currentUserId = req.context.currentUser._id;
    let modelFilter: any = {
      followingId: mongoose.Types.ObjectId(currentUserId),
      requestStatus: {
        $in: [FollowRequestStatus.Accepted, FollowRequestStatus.Sent],
      },
    };

    let reqQuery = req.query;
    const models = await filterPagination(UserFollowing, modelFilter, {
      ...req.query,
      sort: { createdAt: -1 },
      buildQuery: (query, limit, skip) => {
        let queryBase;
```

```
const lookupUser = {
  $lookup: {
    from: UserSchemaName,
    localField: "createdBy",
    foreignField: "_id",
    as: "following",
  },
};
const matchUser = {
  $match: {
    "following.status": StatusCode.Active,
  },
};

if (reqQuery != null && reqQuery.name != null) {
  let searchString:any = reqQuery.name;
  let matchQuery = {
    $match: {
      $or: [
        {
          "following.displayName": new RegExp(searchString, "i"),
        },
        {
          "following.username": new RegExp(searchString, "i"),
        },
        {
          "following.lastName": new RegExp(searchString, "i"),
        },
        {
          "following.firstName": new RegExp(searchString, "i"),
        },
      ],
    },
  };
  queryBase = query.aggregate([
    { $match: modelFilter },
    lookupUser,
    matchUser,
    matchQuery,
    { $sort: { createdAt: -1 } },
  ]);
} else {
  queryBase = query.aggregate([
    { $match: modelFilter },
```

```
          lookupUser,
          matchUser,
          { $sort: { createdAt: -1 } },
        ]);
      }

      return queryBase.skip(skip).limit(limit);
    },
  });

  console.log("models data: " + JSON.stringify(models.data));
  let users = [];
  if (models.data != null) {
    for (let i = 0; i < models.data.length; i++) {
      let user = models.data[i];
      let following = user.following ? user.following[0] : null;
      if (following != null) {
        let userData = following.toJSON ? following.toJSON() : following;
        userData.isFollowing =
          user.requestStatus === FollowRequestStatus.Accepted ? true : false;
        userData.followRequestStatus = user.requestStatus;
        users.push(serializeSimpleUser(userData));
      }
    }
  }
  await closeConnection();
  return res.json({
    data: users,
    pagination: models.pagination,
  });
 } catch (error) {
  return responseError(req, res, error);
 }
}

async function getMyFollowingRequests(req: IRequest, res: Response) {
 try {
  let modelFilter: any = {
    userId: mongoose.Types.ObjectId(req.context.currentUser._id),
    createdBy: mongoose.Types.ObjectId(req.context.currentUser._id),
    requestStatus: {
      $in: [FollowRequestStatus.Accepted, FollowRequestStatus.Sent],
    },
  };
```

```
let reqQuery = req.query;
const models = await filterPagination(UserFollowing, modelFilter, {
  ...req.query,
  sort: { createdAt: -1 },
  buildQuery: (query, limit, skip) => {
    let queryBase;
    const lookupUser = {
      $lookup: {
        from: UserSchemaName,
        localField: "followingId",
        foreignField: "_id",
        as: "following",
      },
    };
    const matchUser = {
      $match: {
        "following.status": StatusCode.Active,
      },
    };

    if (reqQuery != null && reqQuery.name != null) {
      let searchString:any = reqQuery.name;
      let matchQuery = {
        $match: {
          $or: [
            {
              "following.displayName": new RegExp(searchString, "i"),
            },
            {
              "following.username": new RegExp(searchString, "i"),
            },
            {
              "following.lastName": new RegExp(searchString, "i"),
            },
            {
              "following.firstName": new RegExp(searchString, "i"),
            },
          ],
        },
      };
      queryBase = query.aggregate([
        { $match: modelFilter },
        lookupUser,
```

```
          matchUser,
          matchQuery,
          { $sort: { createdAt: -1 } },
        ]);
      } else {
        queryBase = query.aggregate([
          { $match: modelFilter },
          lookupUser,
          matchUser,
          { $sort: { createdAt: -1 } },
        ]);
      }

      return queryBase.skip(skip).limit(limit);
    },
  });

  let users = [];
  if (models.data != null) {
    for (let i = 0; i < models.data.length; i++) {
      let user = models.data[i];
      let following = user.following ? user.following[0] : null;
      if (following != null) {
        let userData = following.toJSON ? following.toJSON() : following;
        userData.isFollowing =
          user.requestStatus === FollowRequestStatus.Accepted ? true : false;
        userData.followRequestStatus = user.requestStatus;
        users.push(serializeSimpleUser(userData));
      }
    }
  }

  // const users = models.data != null ? models.data
  //   .map((e) => <any>e.following ? e.following[0] : null)
  //   .map((e) => ({ ...e.toJSON ? e.toJSON() : e, isFollowing: true }))
  //   .map(serializeSimpleUser) : [];
  await closeConnection();
  return res.json({
    data: users,
    pagination: models.pagination,
  });
} catch (error) {
  return responseError(req, res, error);
}
```

```
}

async function getFollowing(req: IRequest, res: Response) {
  try {
    // if (!("nextPageToken" in req.query)) {
    //   const _users = (await UserFollowing.find({
    //     createdBy: req.context.currentUser._id
    //   }).populate("followingId"))
    //     .map((e) => <any>e.followingId)
    //     .filter((e) => e)
    //     .filter((e) => e.status !== StatusCode.Deleted)
    //     .map((e) => ({ ...e.toJSON ? e.toJSON() : e, isFollowing: true }))
    //     .map(serializeSimpleUser);

    //   return res.json(_users);
    // }
    let currentUserId = req.context.currentUser._id;
    console.log(currentUserId);

    let modelFilter: any = {
      userId: mongoose.Types.ObjectId(req.context.currentUser._id),
      createdBy: mongoose.Types.ObjectId(req.context.currentUser._id),
      requestStatus: FollowRequestStatus.Accepted,
    };

    let reqQuery = req.query;
    const models = await filterPagination(UserFollowing, modelFilter, {
      ...req.query,
      sort: { createdAt: -1 },
      buildQuery: (query, limit, skip) => {
        let queryBase;
        const lookupUser = {
          $lookup: {
            from: UserSchemaName,
            localField: "followingId",
            foreignField: "_id",
            as: "following",
          },
        };
        const matchUser = {
          $match: {
            "following.status": StatusCode.Active,
          },
        };
```

```
    if (reqQuery != null && reqQuery.name != null) {
      let searchString:any = reqQuery.name;
      let matchQuery = {
        $match: {
          $or: [
            {
              "following.displayName": new RegExp(searchString, "i"),
            },
            {
              "following.username": new RegExp(searchString, "i"),
            },
            {
              "following.lastName": new RegExp(searchString, "i"),
            },
            {
              "following.firstName": new RegExp(searchString, "i"),
            },
          ],
        },
      };
      queryBase = query.aggregate([
        { $match: modelFilter },
        lookupUser,
        matchUser,
        matchQuery,
        { $sort: { createdAt: -1 } },
      ]);
    } else {
      queryBase = query.aggregate([
        { $match: modelFilter },
        lookupUser,
        matchUser,
        { $sort: { createdAt: -1 } },
      ]);
    }

    return queryBase.skip(skip).limit(limit);
  },
});

const users = models.data
  .map((e) => (<any>e.following ? e.following[0] : null))
  .map((e) => ({ ...(e.toJSON ? e.toJSON() : e), isFollowing: true }))
```

```
      .map(serializeSimpleUser);
    await closeConnection();
    return res.json({
      data: users,
      pagination: models.pagination,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

export async function getFollowers(req: IRequest, res: Response) {
  try {
    // if (!("nextPageToken" in req.query)) {
    //   const _users: IUser[] = (await UserFollowing.find({
    //     followingId: req.context.currentUser._id
    //   }).populate("createdBy"))
    //     .map((e) => <any>e.createdBy)
    //     .filter((e) => e)
    //     .filter((e) => e.status !== StatusCode.Deleted);

    //   const _modelParsed = (await parseFollowers(
    //     req.context.currentUser._id,
    //     _users
    //   )).map(serializeSimpleUserForFollower);

    //   return res.json(_modelParsed);
    // }

    const reqQuery = req.query;
    const modelFilter = {
      followingId: mongoose.Types.ObjectId(req.context.currentUser._id),
    };
    const models = await filterPagination(UserFollowing, modelFilter, {
      ...req.query,
      sort: { createdAt: -1 },
      buildQuery: (query, limit, skip) => {
        let queryBase;
        const lookupUser = {
          $lookup: {
            from: UserSchemaName,
            localField: "createdBy",
            foreignField: "_id",
            as: "owner",
```

```
  },
};
const matchUser = {
  $match: {
    "owner.status": StatusCode.Active,
  },
};
if (reqQuery != null && reqQuery.name != null) {
  let searchString:any = reqQuery.name;
  let matchQuery = {
    $match: {
      $or: [
        {
          "owner.displayName": new RegExp(searchString, "i"),
        },
        {
          "owner.username": new RegExp(searchString, "i"),
        },
        {
          "owner.lastName": new RegExp(searchString, "i"),
        },
        {
          "owner.firstName": new RegExp(searchString, "i"),
        },
      ],
    },
  };
  queryBase = query.aggregate([
    { $match: modelFilter },
    lookupUser,
    matchUser,
    matchQuery,
    { $sort: { createdAt: -1 } },
  ]);
} else {
  queryBase = query.aggregate([
    { $match: modelFilter },
    lookupUser,
    matchUser,
    { $sort: { createdAt: -1 } },
  ]);
}

return queryBase.skip(skip).limit(limit);
```

```
      },
    });

    const users: IUser[] = models.data.map((e) =>
      <any>e.owner ? e.owner[0] : null
    );

    const modelParsed = (
      await parseFollowers(req.context.currentUser._id, users)
    ).map(serializeSimpleUserForFollower);

    await closeConnection();
    return res.json({
      data: modelParsed,
      pagination: models.pagination,
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getUserList(req: IRequest, res: Response) {
  try {
    const currentUserId = req.context.currentUser._id;

    let modelFilter: any = {
      status: StatusCode.Active,
      _id: { $nin: [mongoose.Types.ObjectId(currentUserId)] },
    };
    if (req.query != null && req.query.name != null) {
      const nameForFullTextSearch: any = req.query.name;
      modelFilter.$or = [
                                    {
                                            username: new RegExp(nameForFullTextSearch,
"i"),
                                    },
                                    {
                                            displayName:                                    new
RegExp(nameForFullTextSearch, "i"),
                                    },
                                    {
                                            firstName: new RegExp(nameForFullTextSearch,
"i"),
                                    },
```

```
                                {
                                    lastName: new RegExp(nameForFullTextSearch,
"i"),
                                },
                            ];
    }

    const models = await filterPagination(User, modelFilter, {
      ...req.query,
      sort: { createdAt: -1 },
      buildQuery: (query, limit, skip) => {
        let queryBase = query.aggregate([
          { $match: modelFilter },
          { $sort: { createdAt: -1 } },
        ]);

        return queryBase.skip(skip).limit(limit);
      },
    });

    const users:any = await parseFollowers(currentUserId, models.data);
    const parsedBlock = await parseBlock(currentUserId, users);
    models.data = <any>parsedBlock.map(serializeSimpleUser);

    return res.json(models);
  } catch (error) {
    return responseError(req, res, error);
  }
}

export async function getOtherProfile(req: IRequest, res: Response) {
  try {
    const userId = req.params.id;
    let currentUserId;
    if (req.context.currentUser && req.context.currentUser._id) {
      currentUserId = req.context.currentUser._id;
    }
    const user = await User.findOne({
      _id: userId,
      status: StatusCode.Active,
    });

    if (!user) {
      await closeConnection();
```

```
      return responseError(req, res, ErrorKey.ProfileNotFound, {
        statusCode: 404,
      });
    }

    const [resultParseProfile]:any = await parseProfiles([user]);

    let result = resultParseProfile;

    if (currentUserId) {
      const [resultParseFollowers]:any = await parseFollowers(currentUserId, [
        resultParseProfile,
      ]);
      const [resultParseIsBlock]:any = await parseBlock(currentUserId, [
        resultParseFollowers,
      ]);
      result = resultParseIsBlock;
    }
    const serializedUser = serializerUser(result);
    await closeConnection();
    return res.json(serializedUser);
  } catch (error) {
    return responseError(req, res, error);
  }
}

export async function getOtherProfileByUsername(req: IRequest, res: Response) {
  try {
    const username = req.params.username;
    let currentUserId;
    if (req.context.currentUser && req.context.currentUser._id) {
      currentUserId = req.context.currentUser._id;
    }
    const user = await User.findOne({
      username,
      status: StatusCode.Active,
    });

    if (!user) {
      await closeConnection();
      return responseError(req, res, ErrorKey.ProfileNotFound, {
        statusCode: 404,
      });
    }
```

```
    const [resultParseProfile]:any = await parseProfiles([user]);
    let result:any = resultParseProfile;
    if (currentUserId) {
      const [resultParseFollowers]:any = await parseFollowers(currentUserId, [
        resultParseProfile,
      ]);
      const [resultParseIsBlock] = await parseBlock(currentUserId, [
        resultParseFollowers,
      ]);
      result = resultParseIsBlock;
    }
    const serializedUser = serializerUser(result);
    await closeConnection();
    return res.json(serializedUser);
  } catch (error) {
    return responseError(req, res, error);
  }
}

export async function allowForgotPassword(req: IRequest, res: Response) {
  try {
    const email:any = req.query.email;
    const check = await User.findOne({
      email,
      status: StatusCode.Active,
    });

    if (!check) {
      await closeConnection();
      return responseError(req, res, ErrorKey.EmailNotFound);
    }
    if (check && check.provider === Provider.Email) {
      await closeConnection();
      return res.json({
        message: "This email allow forgot password",
      });
    }
    await closeConnection();
    return responseError(req, res, ErrorKey.AllowForgotPassword);
  } catch (error) {
    return responseError(req, res, error);
  }
}
```

```
export async function checkAuthId(req: IRequest, res: Response) {
  try {
    const authId = req.params.authId;
    const user = await User.findOne({
      authId,
      status: StatusCode.Active,
    });
    if (!user) {
      await closeConnection();
      return responseError(req, res, ErrorKey.RecordNotFound, {
        statusCode: 404,
      });
    }
    await closeConnection();
    return res.json({
      message: "Check authorization firebase successfully",
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function markAllNotificationsRead(req: IRequest, res: Response) {
  try {
    const currentUserId = req.context.currentUser._id;
    let notifications = await NotificationUser.find({
      userId: currentUserId,
    });
    if (notifications != null) {
      const notificationIds = notifications.map((e) => e._id);
      console.log("Notification Ids: ", notificationIds);
      let readNotifications = await NotificationUser.updateMany(
        { _id: { $in: notificationIds } },
        { $set: { isViewed: true } }
      );
      console.log("notifications: ", readNotifications);
      return res.json({
        message: "All notifications read successfully!!",
        // data: (
        //   await parseMyNotification(readNotifications, {
        //     actorId: req.context.currentUser._id,
        //   })
        // ).map(serializerNotificationItem),
```

```
      // data: notifications
     });
    }
    await closeConnection();
    return res.json(emptyJson());
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function markThisNotificationRead(req: IRequest, res: Response) {
  try {
    const currentNotification = req.params.id;
    const currentUserId = req.context.currentUser._id;
    console.log("currentUserId: ", currentUserId)
    let notification = await NotificationUser.findOne({ notificationId: currentNotification
});
    if (!notification) {
      return responseError(req, res, ErrorKey.RecordNotFound);
    }
    if (notification) {
      console.log("Notification Id: " + notification._id);
      let readNotification = await NotificationUser.findByIdAndUpdate(
        notification._id,
        { isViewed: true },
        { new: true }
      );
      console.log("Read Notification Id: " + JSON.stringify(readNotification));
      return res.json({
        message: "Read Notification successfully",
        data: readNotification,
      });
    }
    await closeConnection();

    return res.json(emptyJson());
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function clearAllNotifications(req: IRequest, res: Response) {
  try {
    const currentUserId = req.context.currentUser._id;
```

```
    let notifications = await NotificationUser.find({
      userId: currentUserId,
    });
    if (notifications != null) {
      const notificationIds = notifications.map((e) => e._id);
      console.log("Notification Ids: ", notificationIds);
      let readNotifications = await NotificationUser.updateMany(
        { _id: { $in: notificationIds } },
        { $set: { isRead: true } }
      );
      console.log("notifications: ", readNotifications);
      await closeConnection();
      return res.json({
        message: "All notifications cleared successfully!!",
        // data: (
        //   await parseMyNotification(readNotifications, {
        //     actorId: req.context.currentUser._id,
        //   })
        // ).map(serializerNotificationItem),
      });
    }

    await closeConnection();
    return res.json(emptyJson());
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getMyNotification(req: IRequest, res: Response) {
  try {
    const currentUserId = req.context.currentUser._id;
    const createdAt = (await User.findById(currentUserId)).createdAt;
    // const notificationUser = await filterPagination(NotificationUser, {
    //   $or: [
    //     { userId: currentUserId },
    //     { typeRole: { $all: req.context.currentUser.role } }
    //   ]
    // }, { ...req.query, sort: "-createdAt" });
    const modelFilter = {
      userId: mongoose.Types.ObjectId(currentUserId),
      createdAt: { $gte: createdAt },
      isRead: false,
    };
```

```
const userBlock: any =
  (await UserBlock.findOne({ createdBy: currentUserId })) || {};
const userBlocked = await UserBlock.find({
  userIds: { $all: [currentUserId] },
});
const models = await filterPagination(NotificationUser, modelFilter, {
  ...req.query,
  sort: { createdAt: -1 },
  buildQuery: (query, limit, skip) => {
    const lookupNotification = {
      $lookup: {
        from: NotificationSchemaName,
        localField: "notificationId",
        foreignField: "_id",
        as: "notification",
      },
    };

    if (userBlocked && userBlocked.length > 0) {
      userBlock.userIds = userBlock.userIds || [];
      userBlock.userIds = userBlock.userIds.concat(
        userBlocked.map((e) => e.createdBy)
      );
    }
    let matchNotification;
    if (userBlock && userBlock.userIds && userBlock.userIds.length > 0) {
      matchNotification = {
        $match: {
          "notification.senderId": {
            $nin: userBlock.userIds,
          },
          $or: [
            {
              "notification.expiryDate": {
                $gte: new Date(
                  moment(new Date()).format(DEFAULT_DATE_FORMAT)
                ),
              },
            },
            {
              "notification.expiryDate": {
                $exists: false,
              },
            },
```

```
            ],
          },
        };
      }

      return query
        .aggregate(
          [
            { $match: modelFilter },
            lookupNotification,
            matchNotification,
            { $sort: { createdAt: -1 } },
          ].filter((e) => e)
        )
        .skip(skip)
        .limit(limit);
    },
  });
  console.log("mode is: ", models.data);

  const parsedNotifications = await parseMyNotification(models.data, {
    actorId: req.context.currentUser._id,
  });
  const serializedNotifications = parsedNotifications.map(
    serializerNotificationItem
  );
  await closeConnection();
  return res.json({
    data: serializedNotifications,
    pagination: models.pagination,
  });
 } catch (error) {
  return responseError(req, res, error);
 }
}

async function setBlock(req: IRequest, res: Response) {
 try {
  const userId = req.params.userId;
  const currentUserId = req.context.currentUser._id;
  const usersBlock = await UserBlock.findOne({
    createdBy: currentUserId,
  });
  let message = "Block";
```

```
const removeRelation = async () => {
  await UserFollowing.deleteOne({
    createdBy: userId,
    followingId: currentUserId,
  });
  await UserFollowing.deleteOne({
    createdBy: currentUserId,
    followingId: userId,
  });
  await UserFriend.deleteOne({
    friendIds: {
      $all: [currentUserId, userId],
    },
  });
};
if (usersBlock) {
  if (
    userId &&
    !usersBlock.userIds.map((e) => e.toString()).includes(userId.toString())
  ) {
    usersBlock.userIds.push(userId);
    await removeRelation();
    await UserBlock.findByIdAndUpdate(usersBlock._id, {
      userIds: usersBlock.userIds,
    });
  } else {
    usersBlock.userIds = usersBlock.userIds.filter(
      (e) => e.toString() !== userId.toString()
    );
    await UserBlock.findByIdAndUpdate(usersBlock._id, {
      userIds: usersBlock.userIds,
    });
    message = "Unblock";
  }
} else {
  await UserBlock.create({
    userIds: [userId],
    createdBy: currentUserId,
  });
  await removeRelation();
}
await closeConnection();
return res.json({
  message: message + " user successfully",
```

```
    });
  } catch (error) {
    return responseError(req, res, error);
  }
}

async function getBlock(req: IRequest, res: Response) {
  try {
    const userBlock = await UserBlock.findOne({
      createdBy: req.context.currentUser._id,
    });
    const userIds = userBlock ? userBlock.userIds : [];
    const query = req.query;
    let result: ISimpleUserResponse[] = [];
    if (userIds && userIds.length > 0) {
      let users;
      if (query != null && query.name != null) {
        const searchString:any = query.name;
        users = await User.find({
          $and: [
            { _id: { $in: userIds } },
            {
              $or: [
                { displayName: new RegExp(searchString, "i") },
                { username: new RegExp(searchString, "i") },
                { firstName: new RegExp(searchString, "i") },
                { lastName: new RegExp(searchString, "i") },
              ],
            },
          ],
        });
      } else {
        users = await User.find({ _id: { $in: userIds } });
      }

      result = users.map(serializeSimpleUser);
    }
    await closeConnection();
    return res.json(result);
  } catch (error) {
    return responseError(req, res, error);
  }
}
```

CHAPTER 7

TESTING

7.1 INTRODUCTION

Software Testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation. The increasing visibility of software as a system element and the attendant “costs” associated with a software failure are motivating forces for well planned, thorough testing.

7.1.1 Testing Objectives

The following are the testing objectives:

-Testing is a process of executing a program with the intent of finding an error.

-A good test case is one that has a high probability of finding an as-yet-undiscovered error

-successful test is one that uncovers an as yet undiscovered error.

7.1.2 Testing Principles

The basic principles that guide software testing are as follows:

-All tests should be traceable to customer requirements.

-Tests should be planned long before testing begins.

-The parate principle applies to software testing.

Pareto principle states that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components.

Testing should begin “in the small “and progress toward testing “in the large.”

Exhaustive testing is not possible.

7.2 LEVEL OF TESTING

There are different levels of testing

->Unit Testing

->Integration Testing

->System Testing

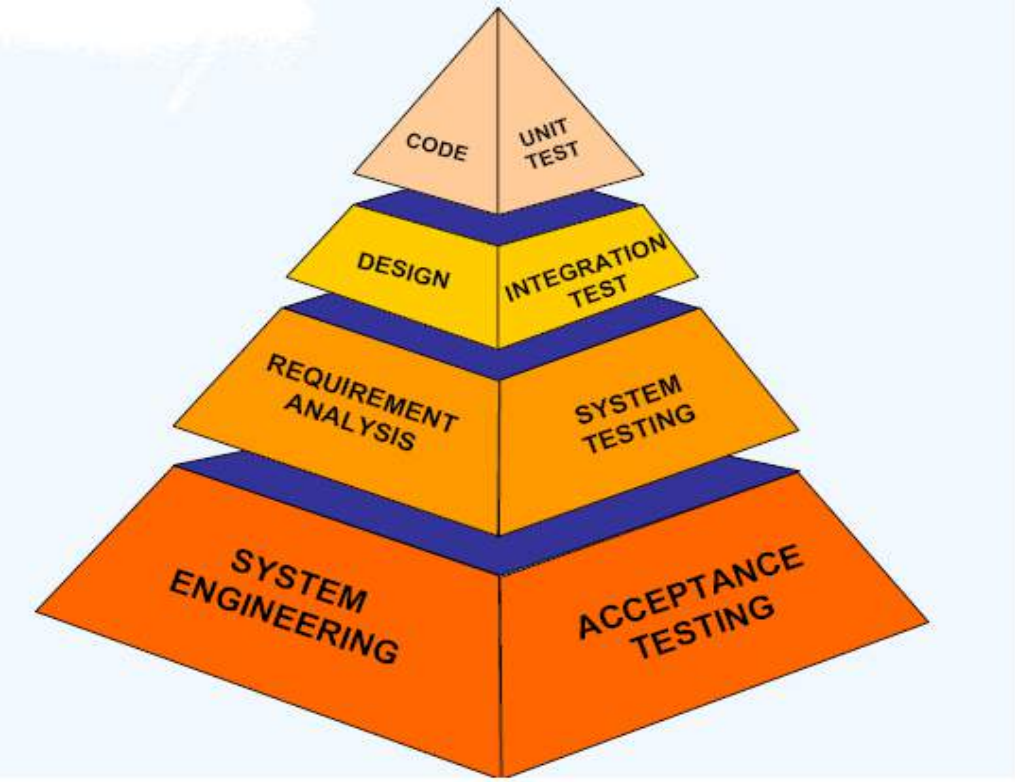


Figure 7.1:Testing pyramid

7.2.1 Unit testing

Unit testing focuses verification effort on the smallest unit of software design, the module. The important control parts are tested to uncover with in the boundary of the module. The module interface is tested to ensure that the information properly flows into and out of the program unit and boundary conditions are tested to ensure that the modules operate properly at boundaries established to limit or restrict processing. Test date is provided through testing screens.

7.2.2 Integration testing

Integrating testing is a systematic technique for constructing Program structure while conducting tests to uncover error associates with interfacing .The objective is to take unit modules and built a program structure that has been directed by design.

• Integration Testing will test whether the modules work well together.

• This will check whether the design is correct.

• Integration can be done in 4 different ways:

7.2.3 System testing

System testing is the process of testing the completed software as a part of the environment it was created for. It is done to ensure that all the requirements specified by the customer are met. System testing involves functional testing and performance testing.

• System Testing will contain the following testing :

- Functional Testing.
- Performance Testing.

• Function Testing will test the implementation of the business needs.

• Performance Testing will test the non-functional requirements of the system like the speed, load etc

7.3 SOME IMPORTANT OBSERVTIONS

7.3.1 System Testing and Validation Results.

System testing was done after the system was duly coded. Individual modules of the system were checked to ensure they are fully functional units before the integrating them. This was done by examining each unit; each script was checked to ensure that it functions as required and that it performed exactly as intended. The success of each individual unit gave us the go ahead to carryout integration testing.

The system was validated using a short questionnaire that was filled by representatives of the users who were let to interact with the system using test data and provided feedback about the system features. This was done to assess if the system met their needs and requirements as regards. It was found out that the system performed in conformance to the then defined user needs and requirements. Results of the validation are shown as percentages of respondents against each requirement.

7.3.2Testing Test Scenarios

1.Checkif the page load time is within the acceptable range.
2.Checkthe page load on slow connections.
3.Check the response time for any action under a light, normal, moderate, and heavy load conditions.
4.Check the performance of database stored procedures and triggers.
5.Check the database execution time.
6.Check for load testing of the application.
7.Check for the Stress testing of the application.
8. Check CPU and memory usage under peak load conditions.
We have checked for scenarios and find that our system performing well in the circumstances.

7.4TESTCASERESULTSUMMARY

TestCase#	Description	Result
TC#1	Loadingthehomepage	Passed
TC#2	Login	Passed
TC#3	Validating	Passed
TC#4	Content	Passed
TC#5	Course pageloading	Passed
TC#6	Reportspageloading	Passed
TC#7	Logout	Passed

`

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

8.1 CONCLUSION

A software project means a lot of experience. I learned a lot through this project. This project has sharpened our concept NodeJS.
It provides easy methods to manage the load of work easily for the users.
It is much fast and more efficient as the data once entered can be used and accessed easily.
This project has given me an ample opportunity to design, code, test and implements an application. This has helped in putting into practice of various Software Engineering principles concepts like maintaining integrity and consistency of data.

8.2 FUTURE SCOPE

- The Future scope is to make the system more user friendly and enhanced.
- And we will make Mobile app for our system.
- I will add Helping BOT in the system.
- Online examination module would be introduced to conduct online examination.
- Further, the user can upload the videos of their interest on to this app and other users who had to see and can get further details can view those videos.

BIBLIOGRAPHY

The books , which are referred and which really helped me in building this system in time, are as follows:-

Books:

BEGINNING NODE.JS Basarat Ali Syed

BEGINNING NODEJS, EXPRESS & MONGODB DEVELOPMENT Francis Buttle

GET PROGRAMMING WITH NODE.JS Jonathan Wexler

SOFTWARE ENGINEERING Roger S. Pressman

WEB DEVELOPMENT WITH NODE AND EXPRESS Ethan Brown

Web Sites:

https://www.w3schools.com/nodejs/
https://www.tutorialsteacher.com/nodejs/nodejs-tutorials
https://www.tutorialspoint.com/nodejs/index.htm
https://www.javatpoint.com/nodejs-tutorial
https://www.tutorialspoint.com/mongodb/index.htm
https://docs.mongodb.com/manual/tutorial/
https://docs.mongodb.com/manual/tutorial/getting-started/