DEVELOPMENT OF A PERSONALISED COURSE TIMETABLE SCHEDULING SYSTEM

AKHIGBE, BENJAMIN OSAZE

16010301027

BEING A PROJECT SUBMITTED IN THE DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS, COLLEGE OF BASIC AND APPLIED SCIENCES IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF BACHELOR OF SCIENCE MOUNTAIN TOP UNIVERSITY, IBAFO OGUN STATE, NIGERIA

2020

CERTIFICATION

This Project titled, **DEVELOPMENT OF A PERSONALISED COURSE TIMETABLE SCHEDULING SYSTEM**, prepared and submitted by **AKHIGBE BENJAMIN OSAZE** in partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE (Computer Science)**, is hereby accepted.

_____ (Signature and Date)

MR. J. A. Balogun

Supervisor

_____ (Signature and Date)

Dr. I.O. Akinyemi

Head of Department

Accepted as partial fulfillment of the requirements for the degree of BACHELOR of SCIENCE (Computer Science)

_____ (Signature and Date)

Professor A.P. Olalusi

Dean, College of Basic and Applied Sciences

DEDICATION

This project is dedicated to God Almighty and all the people that supported me throughout my academic session.

ACKNOWLEDGEMENTS

I owe my profound gratitude to God Almighty who gave the strength, wisdom and courage, divine help and provision to me from the beginning to the completion of this work. I express gratitude to my supervisor, Mr Jeremiah Balogun, for his guidance and support in ensuring the successful completion of this research. God bless you Sir.

I sincerely appreciate the past Dean, College of Basic and Applied Sciences, Dr. Akinwande A.I., for his fatherly advice, guidance and teachings. My heart-felt gratitude goes to the Head of Department, Computer Science and Mathematics – Dr. Akinyemi I.O., and all other members of staff of the department of Computer Science: Late Dr. Oyetunji M.O., Dr. (Mrs.) Kasali F.A., Dr (Mrs.) Oladejo Bola, Dr. Idowu P.A., Dr. Okunoye O.B., Dr. (Mrs.) Oladeji F.A., Mr. Ebo I.O and others to mention but a few.

I acknowledge the constant support of my mentors who had contributed to my academic achievement. They include: Mr. Tomison Oyegoke, Mr. Asuelimhen Christain, Miss Fowobi O. and Mr Jeremiah Balogun, I pray God would continue to increase their knowledge.

I will forever be grateful to my parents Mr. and Mrs. Akhigbe, who sacrificed wealth, time and other resources for the sake of my success; and my siblings for their prayers and support. I also want to appreciate Mr. Jean Cole, Mr. Akinlo Oluwasemiloore, Mr Onwunali Tochukwu, Mr, Solomon Esenyi and all Mountain Top University colleagues and friends for their prayers and support, and help in one way or the other. God bless them all greatly.

ABSTRACT

The aim of this study is to design and implement an automated personalized course scheduling system that will optimize the allocation of courses to their respective venues based on the size of registered students and venue capacity within the acceptable hours of meetings. This was achieved by eliciting knowledge on the various user and system requirements of the faculty staff, formulating the objective function and constraint of the scheduling algorithm, specifying the system design and implementing a prototype system.

Structured interview with department timetable representative were conducted in order to obtain user and system requirements. Hard and soft constraint of the genetic algorithm were formulated based on the limitations and feedbacks of timetable representative. System design was specified using unified modeling language (UML) diagrams such as use-case, sequence and class diagrams. The system was implemented using the combination of Hypertext Mark-Up language (HTML), Cascading Styling Sheets (CSS), Bootstrap, and React JavaScript (JS) framework for web user interface while for the Mobile interface, UIKit and Material design framework for IOS and Android platform respectively. For server side, Node JavaScript framework was used for connecting web and mobile interface to the database while MongoDB atlas for the database implementation. User interface (UI) and unit testing for the web, was carried out with Jester framework, mobile testing was done with XCTest and Junit for IOS and android respectively while the application user interface testing was carried out using Postman.

The results showed that the primary user was responsible for creating course scheduling and managing information regarding a department which in return lead slow response time, the results showed that the secondary users can only access the

system using their school email or Id and passwords provided by the system administrator of the system. The result also shows that the primary user is can view course schedules based on their registered courses, set reminder and make create a conversation.

In conclusion, this study has designed and implemented a system to solve the currently challenge faced in scheduling of courses in academic institutions. The study was able to identify the respective user and system requirements of the system and appropriate designs were used to specify these requirements provided by the users using use-case and class diagrams. The system database was implemented in order to suit the mechanisms and inner workings of the proposed system.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

Scheduling is one of the imperative assignments that are experienced in everyday lifestyle circumstances. In the real world, a series of choices to be made to solve the dilemma always arise. Many fields like school, travel, entertainment etc. are all about planning and scheduling (Rohin, 2016). According to Wren (1996), Scheduling is the allocation, subject to constraints, of resources to objects being place space-time, in such a way as to minimize the total cost of some set of the resources used. In the educational sector, timetable scheduling is an essential component in which activities are logically structured in a time-wise manner, presented on papers and, placed on notice boards in order to avoid conflicts of event.

However, the use of paper and notice board-based schedules presents the challenges of timetable forgetfulness, misinterpretation, wastage of paper, cluster of schedules to fit into a specific paper size, allocation of large number of student to more than the capacity of the room, cost of rescheduling and slow time to reach for the rescheduled timetables (Muhammad, Mustapha, & Yahaya, 2017). However, automated timetable system that is currently existing is faced with the challenges of lack of user-friendliness, cluster of schedules, and inability to personalize timetable based on registered courses.

hence, there is a great need for an automated personalized course scheduling system.

The automated timetable scheduling system comprises of a mobile and web-based interface were the web interface is implemented for scheduling timetables and the mobile interface for accessing, registrations of courses and enforcing schedules (setting reminders) based on user category and preference. In this system, genetic

algorithm was employed as the preferred choice of scheduling optimization algorithm since it offers a favorable compromised between the quality of solution and run time spent finding it (Rivera *et al,*. 2019).

1.2 Statement of the Problem

For academic institutes such as universities, creating an error-free timetable is a complicated task and a lot of constraints arise when doing so. these constraints are divided into two types, one is hard constraints such as time conflicts, room assignments, etc. and the other one is soft constraints such as departmental preference. Here, hard constraints cannot be avoided when creating timetable (Chaya *et al*., 2016).

Generally, scheduling of courses in many universities is prepared manually based on the level of the administrator experience. Facilities and resources such as courses, instructors, rooms and, laboratories etc. are consider by the administrator when creating a timetable. Therefore, based on all the mentioned constraint, the manual based approach for creating a timetable is a very exhaustive and time-consuming task which leads to issues of resources optimization due to inadequate lab and hall space (Mohammed *et al*., 2017).

However, automated timetable system that is currently existing is faced with the lack of user-friendliness, challenges of cluster, and inability to personalize timetable based on registered courses.

1.3 Aim and Objectives of the Study

The aim of this study is to design and implement an automated personalized course scheduling system that will optimize the allocation of courses to their respective

venues based on the size of registered students and venue capacity within the acceptable hours of meetings.

The specific objectives of the study are to

i. elicit knowledge on the various user and system requirements of the faculty staff;
ii. formulate the objective function and constraint of the scheduling algorithm based on (i);
iii. specify the system design; and
iv. implement of a prototype system.

1.4 Research Methodology

In order to meet up with aforementioned objective of this study, the following methods will be adopted.

a. Structured interview with department timetable representative were conducted in order to obtain user and system requirements.
b. Hard and soft constraint of the genetic algorithm were formulated based on the limitations and feedbacks of timetable representative.
c. System design was specified using unified modeling language (UML) diagrams such as use-case, sequence and class diagrams
d. The system was implemented using the combination of Hypertext Mark-Up language (HTML), Cascading Styling Sheets (CSS), Bootstrap, and React JavaScript (JS) framework for web user interface while for the Mobile interface, UIKit and Material design framework for IOS and Android platform respectively.
e. For server side, Node JavaScript framework was used for connecting web and mobile interface to the database while Mongodb atlas for the database implementation.

f. User interface (UI) and unit testing for the web, was carried out with Jester framework, mobile testing was done with XCTest and Junit for IOS and android respectively while the application user interface testing was carried out using Postman.

1.5 Scope and Limitation of the Study

This study is limited to the development of automated timetable scheduling for scheduling of courses for departments in a university. The system is limited to the departmental timetable representative and student. Faculty based courses was not taken in consideration in the development of the schedules.

1.6 Significant of the Study

a. For Mountain Top University

The system would drastically minimize the time and resources expended on schedules, thereby allowing the departmental timetable representative more time to focus on other pressing matters of the institute which in return will lead to greater productivity. It help improve flexibility in timetable construction.

b. For educational system and other educational institutes:

Educational institutions can implement the system and benefit from a computed scheduling approach. It would help to ease the administration of the school and, if possible, minimize costs by providing an improved timetable schedule.

c. For research

The development of the system would make a significant contribution to the improvement of computer science by integrating several algorithms. It would also allow the rising artificial intelligence community to take a step forward. The research will be

part of the portfolio that demonstrates the researchers' abilities to solve difficult issues. (Celiz, 2018).

1.7 Justification of the Study

In an education system, most especially a tertiary institution the problem of clashing course and well as cluster of courses may arise due to the use of traditional timetable as it is a generalized format for course scheduling. A personalized automated course scheduling system will improve the flexibility of a traditional timetable construction, productivity as well as vast academic practices, thus, the focal point of this study.

1.8 Arrangement of Thesis

This chapter presents the introductory aspect of the project. Chapter two contains a literature review on the said topic. Research on already existing works to get detailed information about features and functionality. Other observations were from journals and articles on the concept of Automated timetable scheduling system. Chapter three is the Methodology; this chapter provides product description, its function design techniques, and various UML diagrams. Chapter four is the implementation and testing. The chapter describes how the application was developed, the developmental techniques, and the reason for the choice of techniques. It also describes the testing techniques and provides results showing that the web application is in conformance with the requirement specification. Chapter five is the recommendation and conclusion. It summarizes the project experience and discusses how the project can be developed further.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

For a number of decades scheduling has been the subject of intense research. Generally scheduling and timetable are commonly considered to be two distinct tasks with the assumption that scheduling is used to as generic term to cover specific categories of problems and timetable for special case of generic scheduling activities (Sandhu, 2003). However, scheduling refers to the restricted allocation of resources to objects, to be put in space-time in order to reduce the overall expense of the resources needed and Timetable construction is the allocation, subject to constraint of given resources to objects being placed in space-time in such a way as to satisfy or nearly satisfy a desirable set of possible objective (Sandhu, 2003).

Thus, the term scheduling covers all aspect of the activity of allocating resource and at the same time, satisfying some predetermined objective. However, due to the enormity of the problem, it becomes necessary to classify the scheduling problem int specialized activities such as timetabling. Thus, in practical terms the timetabling problem can be described as scheduling a sequence of lectures between teacher and students in prefixed time period, satisfying a set of varying constraints (Sandhu, 2003).

2.1.1 Classification of Scheduling

Scheduling can be generally classified into one of the following types.

a. Semi-active Scheduling

These workable schedules are accomplished as early as possible by sequencing operations. No procedure may be begun beforehand without modifying the processing sequences in a semi-active schedule(Rohini, 2016).

b. Active Scheduling

This feasible schedules are schedules in which no process will start early without interruption or breaching a precedence cap. Semi-active timetables are still active. An optimal strategy is often active in order to restrict search space safely to the collection of active programs (Rohini, 2016).

c. Non-delay Scheduling

This viable schedules are schedules under which there is no uninterrupted machine until it will start to function. The schedules for non-delay are necessarily active and thus semi-active (Rohini, 2016).

Examples of scheduling system includes, Job scheduling system, Parallel Machine Scheduling, Group Job Scheduling , Resource Constraint Scheduling, Timetable Scheduling and Dynamic Task Scheduling etc (Rohini, 2016).

2.1.2 Timetable Scheduling

A timetable is an organized list, usually set out in tabular form, providing information about a series of arranged events in particular, the time at which it is planned these events will take place (Onuwa, 2015). When Constructing a timetable many approaches and models have been proposed for dealing with the variety of timetable problem. The problem ranges from the construction of semester or annual timetable in school, colleges and universities to exam timetabling at the end of these periods (Sandhu, 2003).

In academic intuitions for instance, there are two types of timetable that is recognized which are course and the examination schedules (Herath, 2017). When creating a timetable schedule, course should be assigned to a specific timeslot for five working days of the week taking into consideration the specific classroom suitable for the respective courses and the registered number of students. Thus, a feasible timetable

in academic institution is a description of the movement of students and staff from one classroom to the next one, the location of the classroom, and the timeslot (Herath, 2017).

Although the manual approach of solving time problem is time consuming and inaccurate, it was observed by Herath (2017), that as the complexity of university increase it become necessary to adopt automate scheduling method to ease the task of timetabling since as the number of student with diverse interests, and requirements increases and the teaching technique program get complicated with the growth of university, the number of growing constraint grows resulting to an exponent rise in the computation time, making it an NP-Complete operation.

2.1.3 Scheduling System Algorithms

The early technique used in solving timetabling problem were based on a simulation of the human approach in resolving the problem. These included techniques based on successive augmentation that were called direct heuristics. These techniques were based on the idea of creating a partial timetable by scheduling the most constrained lecturer first and then extending this partial solution lecture by lecture until all lectures were scheduled (Sandhu, 2003). Evolutionary technique has also been used to solve the timetable scheduling problem which includes algorithms such as genetic algorithm,Tuba Search. The use of artificial with the combination of operation technique has also been used in resolve the problem of timetable scheduling.

2.2 Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) is a framework that defines each phase of the steps involved in software development. It covers the comprehensive software design, delivery, and maintenance scheme. SDLC describes the entire

development process shown in fig 2.1 i.e. all the activities involved in the preparation, production, testing, and delivery of a Software Product. Methodologies in application development have improved progressively over time.

System development in application engineering can be referred to as a series of steps, activities, methods, tools, and techniques which are used in developing web application products to produce a project. Many believed that the major reason for application failure is a result of a lack of planning and system development methodology. To avoid failure of whatsoever, proper planning of a project is encouraged. However adopting a particular methodology would not always guarantee success in a project (Bagul, 2015).

The basic stages for developing an application are listed as follows: Requirement analysis, Design, Implementation, Testing and Maintenance as shown in Figure 2.1. The software development cycle can be divided into three different models; incremental phase, iterative phase, and iterative-incremental phase

2.2.1 Incremental phase

The incremental phase of a software development life cycle is a traditional and highly sequential model that relies heavily on upward planning through documentation. Requirements for a software project are thoroughly defined upfront in this model. Because of this reason, the model is suitable for developing complex systems. Examples of this model are the waterfall model, Nolan's stage model, B, and V-model. The Waterfall model encourages specification of requirements before design and also design before coding and reduces development and maintenance cost by generating documents that can be utilized in test and maintenance.

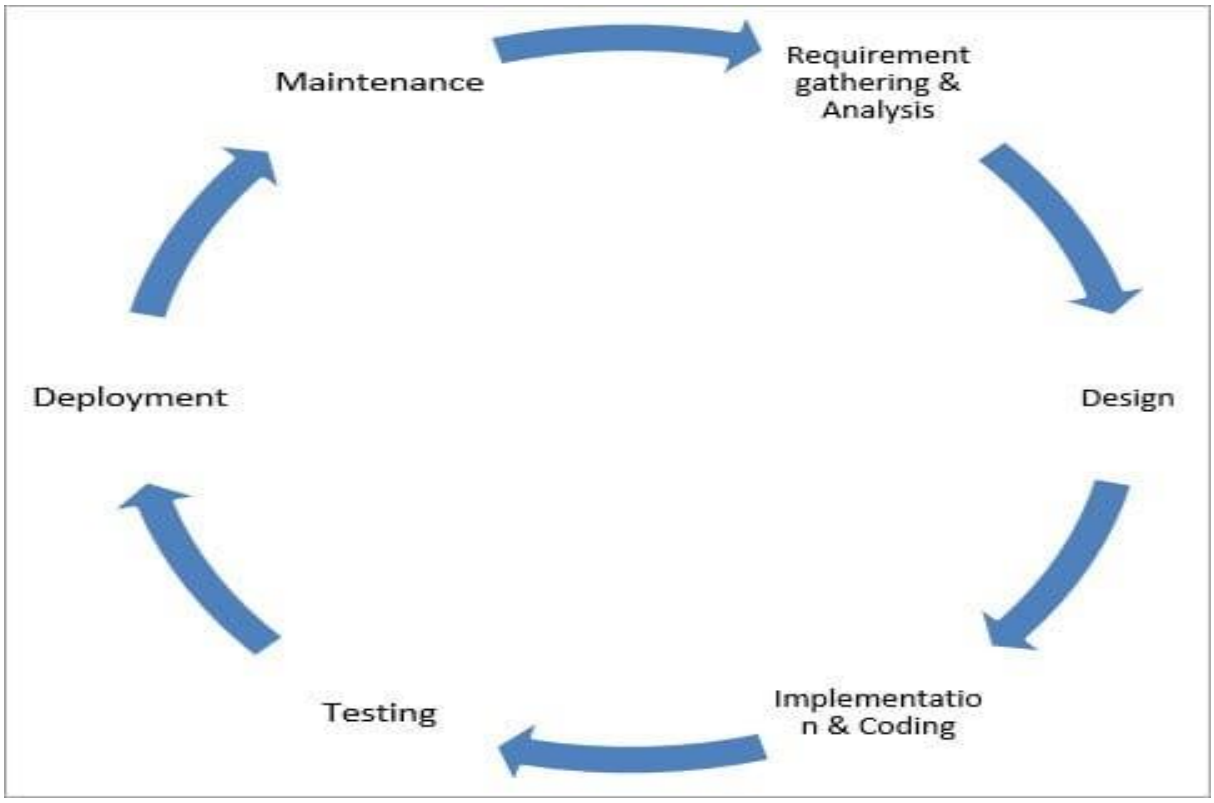

Figure 2.1 Software Development Life Cycle

(Source: Bagul, 2015)

The incremental phase life cycle such as the V-model, Waterfall model is a good development process but it is not a suitable choice of methodology for this project due to the following disadvantages

a. The next phase cannot be started without completing the previous stage and because of this, it is termed a rigid methodology.
b. It usually takes a longer time to develop an application product using this methodology as compared to others. Considering the time frame of this frame of this project which is a matter of a few month.
c. It lacks the customer's involvement after this requirement definition is completed and is not suitable for this project.

2.2.2 Iterative model

This process starts with a simple implementation of software requirements. An iterative model, the model stages of development can start from any phase which basically means that the next phase can start even if the previous one is not yet completed. It can start without fully specified requirements. The model involves the end-user in the development stages of the product. Examples of iterative models include joint application development (JAD) prototyping and rapid application development (RAD).

2.2.3 Iterative-incremental model

This is a combination of iterative and incremental models. It is believed to be the fastest and latest model for developing software applications. It is lightweight and iterative, able to adapt to changes, ensure that the production of quality of application meets the needs of the end-users. The model is divided into broad groups:

a. Agile method e.g. Extreme programming and scum; and
b. Spiral model e.g. Rational unified process (RUP)

The agile method is the most widely used in this group. According to (Pressman, 2010) extreme programming method function on iterative development, recurrent discussion with the consumer, needing minor and regular publication, and projects are delivered just in time.

2.3 Genetic Algorithm

Genetic Algorithm (GA) is an efficient heuristic search algorithm focused on evolution and natural selection. GA is working towards sustainable options in the design space utilizing the *survival of the fittest* concept. According to this theory, the best-suited human should overshadow the others. There is a demographic in every generation and every member of this demographic is a possible solution to the problem. Each individual in this population is going through an evolution, where only the strongest individual survives.

Individuals that are best suitable for each age should have an enhanced probability of replication contributing to better replication. Two good individuals can produce an offspring that is better than their parents because they can inherit a mixture of genes from both parents and end up surpassing them. Because of this, every generation should aspire to change and become better adapted to its environment, just as in nature. (Hakan, 2015).

2.3.1 Genetic Algorithm Operation

In its natural form, a genetic algorithm consists of the following genetic operations, selection, reproduction, crossover, and mutation as shown in Figure 2.2.

a. Selection

The selection process is used for choosing the most appropriate individual to be the progenitors of the next generation of the population. In nature there are various

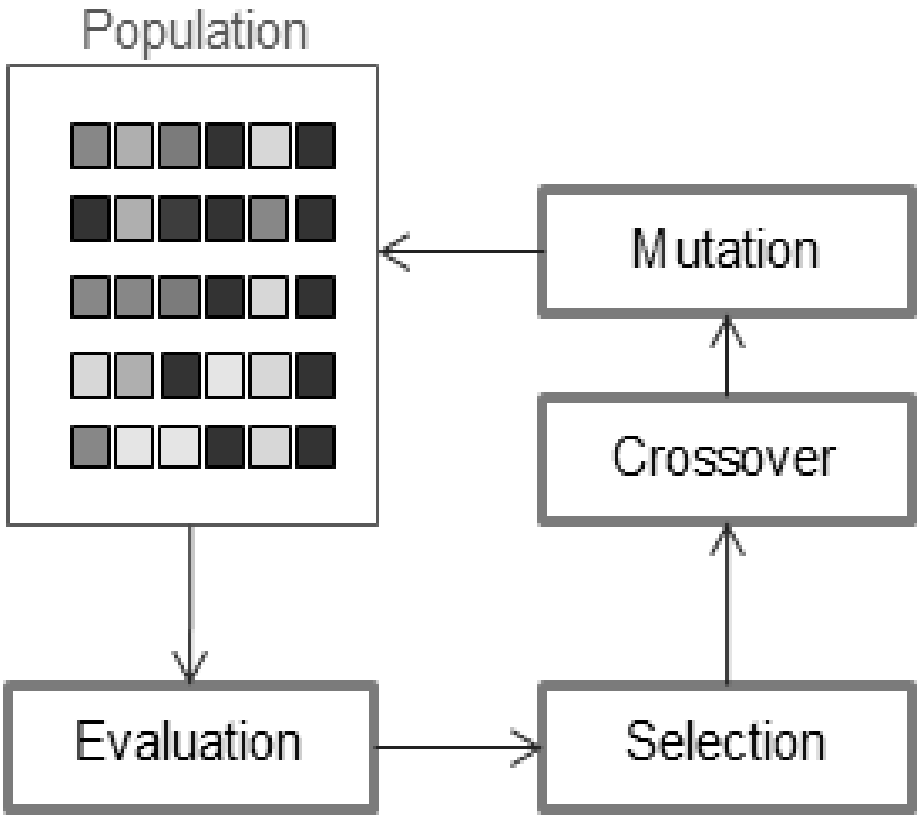

Figure 2.1: Genetic Algorithm Operations

(Source: Mijwel, 2016)

factors that make it possible for an individual to have offspring. Firstly, it lives either because the predators do not consume it or because it can procure food. Secondly, you're looking for a reproductive mate. The last consideration is that both individual to create a new one. However, in particular *the greatest* individual cannot replicate, but another individual of worst qualities is conceivable.

In genetic algorithm, selection is a series of rules for the choosing of the next generation parents. These parents reproduce and produce offspring (genetic crossing). Selection per tournament is a method commonly used in genetic algorithms. This scheme entails selecting a certain number of people randomly from the population. The best of all is selected by the father among these people. The procedure is replicated when choosing the mother: a few people from the community is selected randomly and the person with the highest quality is selected. This method maintains a minimum of variation as not all the greatest person in the community chooses to have descendants (Mijwel, 2016).

b. Reproduction

Reproduction in this context involves cloning an individual. In other words, an individual can move without change to the next generation. Reproduction is thus a genetic operator that rejects crossing and mutation as the latter modifies individuals that are passed to the next generation. The aim of breeding is to retain individuals with a high degree of health in the next generation. The idea of reproduction concerns the idea of *elitism* that preserves the best people from generation to generation, so that their knowledge is not wasted (Mijwel, 2016).

c. Crossing

The individuals chosen for the preceding phase are crossed or combined during this phase. In other words, the genes of both parents are combined together to cause the multiple offspring (Mijwel, 2016).

d. Mutation

The Mutation is seen as an integral operator, which gives the individuals of the population a minor aspect of randomness. Although the crossing operator is considered responsible for the search for potential solutions, it is for the mutation operator to increase or decrease the field of search of genetic algorithms and to encourage heterogeneity (Mijwel, 2016).

2.3.2 Reasons for using Genetic Algorithm

Reason for using genetic algorithm can also be referred to as the advantage of genetic algorithm and they are discussed as follows: effective with regards to large number of variables, no derivative information is needed, effective for machines in parallel, optimizing variable surface at very complex cost, provide a list of best variables, not just one solution and develops data or analytic functions produced by numerical data experiment.

2.3.3 Genetic Algorithm Process

A chromosome populace with a random gene array is initiated by the following steps in a genetic algorithm as shown in Figure 2.3. Produce an initial chromosome population, assess the adequacy of each (individual) population chromosome, based on the initial findings, pick chromosomes for mating, develop offspring by crossing the chromosomes selected, randomly mutate genes and repeat steps 3-5 to establish a new population. Stop the algorithm until, after a predetermined number of generations, the best answer has not improved.

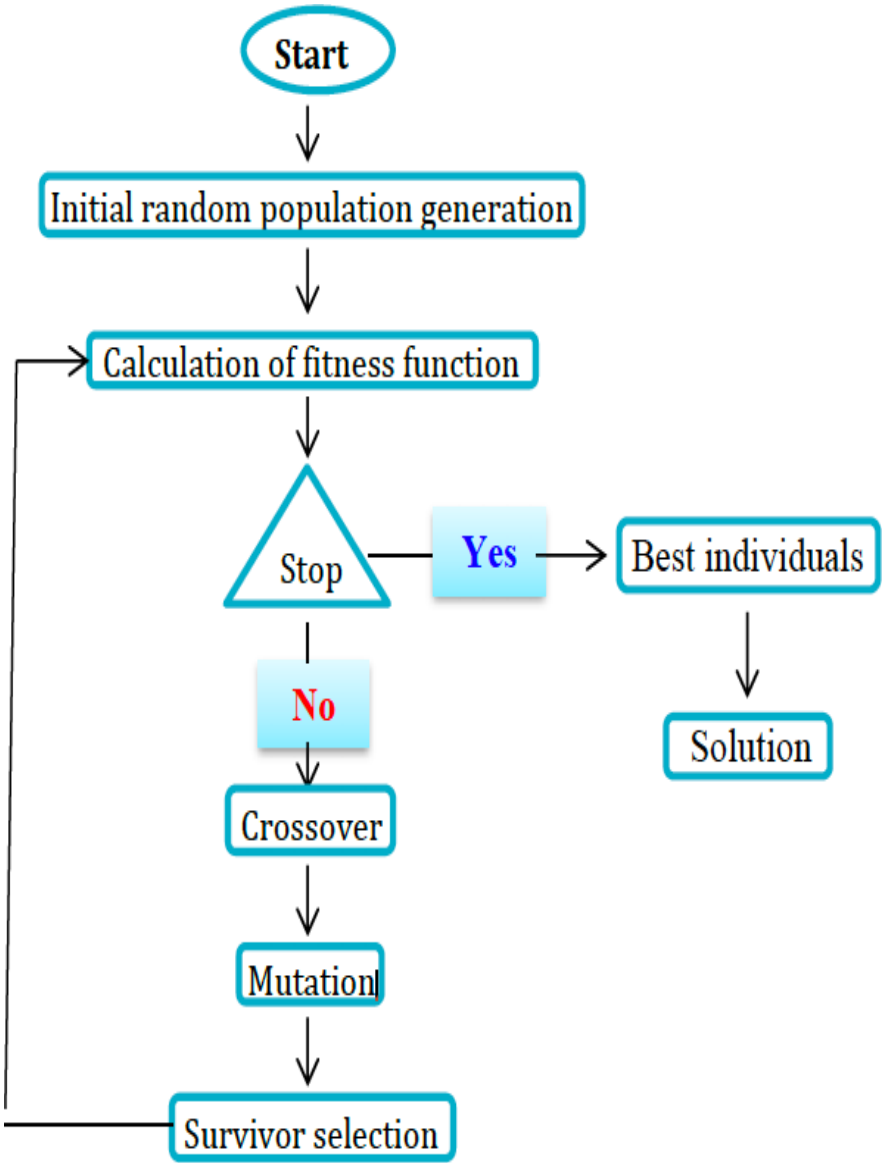

FIGURE 2.2 PROCESS OF GENETIC ALGORITHM

(Source: Herath, 2017)

2.3.4 Use and Limitation of Genetic Algorithm

The genetic algorithm (GA) produced better scheduling frameworks. It gives the user the flexibility of choice within a set of different schedules and can be applied to other highly constrained combination optimization problems. Further, some data mining issues without having any solution and lottery games with probabilistic theory also use this algorithm. The major disadvantage of the GA is when the population is large the algorithm execution time also increases. Chiu-Hung Chen and team workers supply evidence with the use of GA for solving multimodal manufacturing optimization problems. GA itself takes a long time to be executed and requires a certain machine configuration. This can be a problem for timely execution. The second limit of the algorithm is the importance of the random part. Due to a huge set of solutions, the algorithm cannot guarantee to get the best result or the achievement of a certain level of fitness.

2.4 Theory of Natural Selection

The history of the species is focused on *Preserving favorable variations and removing unfavorable variations*. The variability applies to the differences exhibited by the members of the species as well as the offspring of the same parents. There are a lot more people born than they will live, and there's a constant fight for survival. Individuals with an edge have a better chance of survival, i.e. the survival of the fittest. For example, Giraffes with long necks can have food from tall trees as well as from the ground, on the other hand, goat, deer with small necks can have food only from the ground. As a consequence, natural selection plays a vital function in this evolutionary cycle. The beneficial (fit) person lives on parallel lines in the genetic equation in each iteration and the unfavorable (unfit) person dies out. With each step, the cycle begins

and persists until the point at which secure or streamlined approaches are achieved, which can be referred to as adaptability (Mijwel, 2016). Table 2.1 shows the commonly used terms in natural evolution and genetic algorithm

2.5 Fitness Function

The fitness function is an indicator of the individual's efficiency. The fitness feature shall be structured to provide an evaluation of the individual's success in the present population. The fitness task based on objective merit is followed by the selection of process (Kumar, 2012). This fitness is used in the actual selection process shown in equation 2.1

$$\text{minf}(x) = 50(x(1)^{!} - x(2))^{!} + (1 - x(1))^{!} \quad (2.1)$$

2.6 Review of Related Works

Chaya Andradi and Saminda Premaratne, (2016), worked on Utilization of Timetable Management System to a Medium Scaled University in order to solve the problem of resource optimization. Genetic algorithm was employed in this study and the used of PHP server as scripting language for coding process. MVC architecture was used and MYSQL database management system for the relational database management System. The overall system developed was user friendly and time effective but when tested with large set of data the performance of the genetic algorithm reduces to do it fitness function. However, resource optimization process was achieved. The study was unable to personalize timetable and the performance declined as the dataset increased.

Alinaswe Siame and Douglas Kunda,(2016), developed university course timetabling using Bayesian based Optimization Algorithm. The algorithm was

Table 2. 1 Camparison of Natural Evolution and Gentic Algorithm Terminology

Natural evolution	Genetic algorithm
Chromosome	String
Gene	Feature or character
Allele	String position
Locus	Structure or coded string
Phenotype	Parameter set, a decoded structure

(Source: Mijwel, 2016)

employed in this research developing a model that help in the allocation of course based by predicting possible outcomes. The allocation course by predicting the preferred days and timings of lecturers was achieved. However, this approach is highly prone to technical challenges and mistakes due to misspecification of the model.

Muhammad, Mustapha, and Yahaya (2017) Designed and implemented an android web-based customization system which is limited to a web-based system for scheduling and android based system for students. The research focused object-oriented design approach which was used data manipulation and UML Model. The system was able to scheduling, customization and provide on-the go reminder facility for timetable schedules but the system was not user-friendly.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

Scheduling is continuously said to be a complex optimization issue that has appeared to be related to the clique of minimization issue which is called NP-Complete. In such a kind of issue where no proficient algorithm is known, it is perfect to apply genetic algorithm to such an issue which is utilized for searching a solution space. The general constraint for timetable scheduling can be classified in two types which are the hard constraint and soft constraints. A practical and feasible timetable scheduling must satisfy all the hard constraints with no consideration while the soft constraint are not absolutely essential but the amount of related violation should be minimized to the maximize the perfection of the schedule. To implement the Course Scheduling System, users and systems must meet a number of requirements that are typically known as functional and non-functional requirements.

3.1.1 Functional Requirement analysis

This is a summary of the system operation and components. it is liable for the inputs, actions, and outputs of the software system. The system is evaluated using the necessary unified modeling languages. Therefore the functional requirements for developing the system by users were as follows:

a. The system will only allow authorized students access to the system using usernames and passwords provided by system administrator;

b. New student and existing student must be able to change their default passwords to their preferred password;

c. The system must be able to create an initial population when at initial stage for creating a schedule

d. The system must be able to assign a fitness point for resolving clashes when two or more course are allocated to the same time

e. Student must be able to view course based on the registered courses

3.1.2 Non-Functional Requirement analysis

This determines the system's quality function. it is a set of standards that are used to test the specific functionality of a system. It enables the ability to impose restrictions or restrictions on the system design across the different agile backlogs. The external limitation of the system should include:

a. **Accessibility**: tackles unequal app-related issues Disability Training for men. As regards online usability, that means Disabled people should experience, grasp, process, and communicate similarly Or resources and websites.

b. **Usability**: it is designed to be efficient, reliable, and competitive. Usability includes the design of user experience. This could include general aspects that affect all and do not impact people with disabilities negatively.

c. **Inclusion:** addresses ethnicity and encourages the inclusion of all residents as far as practicable. This is also recognized as a uniform style in certain regions and For everything, architecture.

d. **Security**: the mechanism for enforcing activities and processes intended to safely monitor and preserve knowledge is defined in terms of information management security. The program was built to adjust device storage control permissions only from the system data administered; backup every 24 hours of the system data and backup copies in a secure place that is not in the same building as the network. This system will use special encryption techniques in securing users' data, communication between the system's data server and clients will be authenticated in all external correspondence.

e. **Authentication**: authentication is the method of verifying a person's or device's identification in computing. It's used to figure out what someone or something really is, whether or what that is.

f. **Authorization**: Authorization refers to the method of assigning rights to systems and, generally, to users. This varies from authentication which is the process used to identify a user. Once identified(reliable), the privileges right, property, and permissible actions of the user are determined by authorization.

g. **Integrity Control**: Integrity in terms of data and network protection is requirement that information can only be obtained by those who are allowed to do so.

h. **Reliability**: A device is capable of executing the necessary tasks over a given amount of time, under certain circumstances.

i. **Confidentiality**: is the degree to which the information program preserves confidential data and enables only permitted access to the data

j. **Dependability**: is the durability of a computer system to deliver a function that can be trusted by consumers.

3.1.3 Hardware requirements

For reliable and productive project efficiency, certain hardware requirements must be fulfilled which are as follows: a web server with a considerable amount of large RAM and hard disk, an Intel i5 macOS with a minimum version of Mojave (version 10.1) and 8 GB ram size, an IOS 12.4 (minimum, iPhone and iPad users), a wireless router or alternative internet service provider (ISP) and Uninterrupted power supply (UPS) or Inverter.

3.1.4 Software requirements

For efficiency of use and to have system performance in developing of software various software requirement must be met as showed in Table 3.1.

3.2 Constraints

Constraints are limits which are beyond the project team control and need to be addressed. They are not conflicts actually. The project manager should, however, be mindful of the limitations since they are limitations within the project. For instance, date constraints mean that some events (perhaps the project's end) need to take place on certain dates. Resources are almost always a constraint since in an unlimited supply they are not available. (Life cycle for time table).

a. 3.2.1 Hard constraints

A timetable that violates a hard constraint is not a viable solution, and the scheduling algorithm must be remedied or refused. Hard constraint contains *First Order Conflicts* (Life cycle for time table).

i. No student can be assigned to more than one course at the same time
ii. A lecturer cannot teach more than one class at the same time
iii. The room should satisfy the feature required by the course.
iv. The number of students attending the course should be less than or equal to the capacity of the room.
v. No more than one course is allowed at time slot in each room.
vi. To Generate the timetable based on the number of periods and time schedule.
vii. All the available courses must be entered in the timetable with flexibility for multi-period sessions.
viii. Lecturers teach at their available schedule.

Table 3.1 Software Requirements

Platform	Requirements
Web user interface	HTML, Bootstrap 4
Web Client-side scripting language	React JS
Mobile client-side language	Swift (IOS) and Kotin (Android)
Server-side language	Node JS
Database	Mongodb
Web server	Heroku (sever-side) and Netlify (client-side)

ix. Classes can only be scheduled in a free room.

3.2.2 Soft constraints

Soft constraints are less important than hard constraints, and it is usually impossible to avoid breaking at less some of them. Whichever timetabling method is applied, timetable are usually rated by a penalty function, which calculates the extent to which a timetable has violated its soft constraints. Some soft constraints are more important than others, and this is often specified with a primary value (Life cycle for timetable). Soft constraint are less important than hard constraint, and it is generally difficult to prevent violating at least any of them. Whatever method of scheduling is used, timetable are usually calculated using a penalty function to measure the extent to which a schedule has breached its soft constraint. Some soft constraint are more important than others and sometimes a priority value is defined.

i. A student has attend only one course in a day.

ii. A student has attend more than two courses consecutively.

iii. A lunch break must be scheduled.

iv. Lecturer's daily lecture hours should be restricted to be within the allowed maximum hours.

v. Lecturer should have normalized distributed load based on the instructor pool of subjects.

3.3.3 Method of software development

Extreme programming (XP) is probably the best known and commonly used agile approaches. It was thought that Beck coined this name in 2002, when the method was formulated by pushing recognized good practice, such as iterative development, to 'extreme' levels (Sommerville, 2011). it is methodology is very flexible and make the use of iterative development a central focus of it as shown in Figure 3.1.

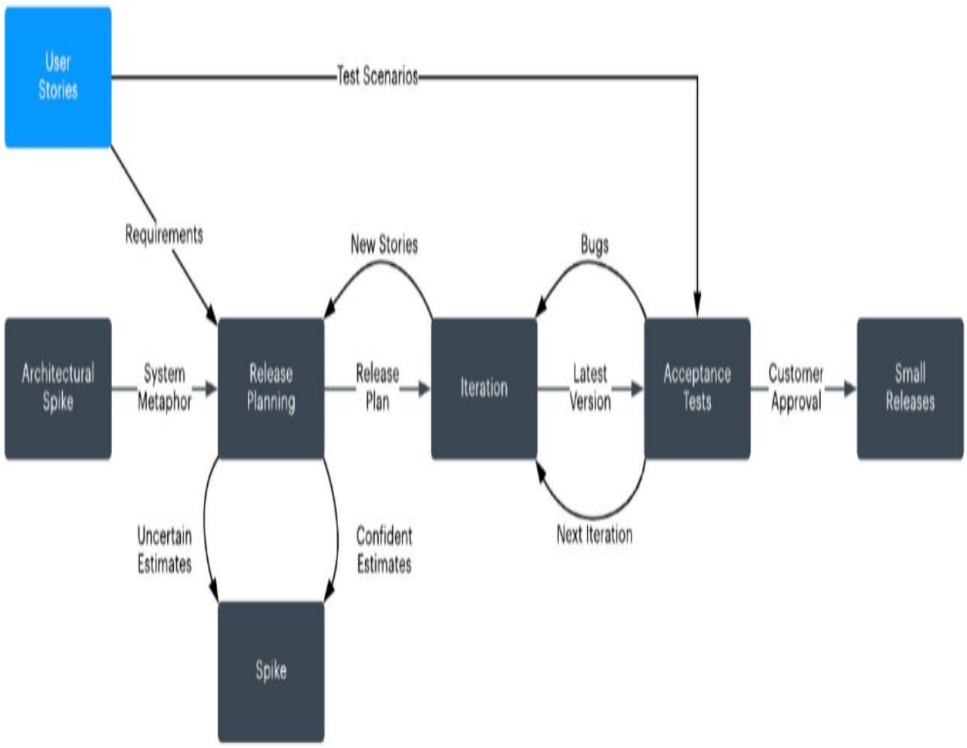

Figure 3.1: Extreme Programming (XP) Methodology

In extreme programming, every contributor to the project is an integral part of the *whole team*. The team form around a business representative called "The Customer", who sits with the team and works with them daily (Jeffries, 2011). Mostly extreme programming is idea for software environments that is dynamically changing based the customer requirement. extreme programming is set up for small group of programmers between 1 and 12 which enable easy testability of system requirements. According to (Well, 2013), the main goal of is to deliver software that is needed when it is needed. Due to this reason and the aforementioned above extreme programming was the preferred choice for developing the Automated timetable scheduling system since the system relies on tweaking and managing constraint.

3.2 System Analysis and Design

System analysis is the method of gathering and interpreting the data, diagnosing issues, and proposing changes to the program by utilizing the details. System design involves analyzing and configuring the necessary components of hardware and software to support the architecture of a solution. The design process transforms the basic concept specifications into a robust and accurate device specification. Many of the most relevant tasks in this process are:

a. Identification of all necessary data to handle a course scheduling system and carrying out all required operations for the system user;
b. Characterization and documentation of all related entities required in the implementation of a course scheduling system.
c. System components design: coherent diagram for unified modeling language (UML) shows the relationship between the course scheduling system, the

structures, inputs, outputs, central processing, handling, device Interfaces, technical and system-wide architecture;

d. Carry out design procedures to ensure sure everything is programmable and fully technically; and
e. Beginning development of approaches to user support and system maintenance afterwards.

3.2.1 Objectives of the Design

The objective of the design is to create an automated personalized course scheduling system that is ease the creation of course schedules and give student adequate access to the schedules based on registered courses.

3.2.2 Factors considered in the Design

The following factors were put into consideration during the design of the system. User friendliness, flexible timetable construction, effective execution of academic activities, programming paradigm and organization of system components

3.3 System Modeling

System modeling is a method abstracting and organizing essential aspects of how the system appears. It involves the desiging of the software application. Modeling is done prior to coding of the system. In simulation of the system, Unified Modeling Language (UML) tools were used.

3.3.1 UML Diagram

This is a system notation for object-oriented models which provides a collection of modeling conventions for defining or representing an object software system. In its field of system analyzes and development, UML has become an Object Modeling Standard, introducing a number of techniques (Onuwa, 2015). Hence it is the pereffered

choice for this project. To model a system, UML provides 10 different diagrams. The following diagram can be found as follows: Use case diagram, Class diagram, Object diagram, Sequence diagram, Collaboration diagram, State diagram, Activity diagram, Component diagram, Deployment diagram and Package Diagram. In this project, The Context flow diagram, The Use case diagram, Class diagram, Sequence diagram, Activity diagram and state diagram will be used for modeling the system.

3.3.2 Context flow diagram

The context diagram is used to evaluate the systems context and limits to be modeled: what objects are modeling inside and beyond the system and what are the system 's relation to these external entities as shown in Figure 3.2. In order to describe and explain the software device constraints, a background diagram, also called a data flow level 0 diagram is drawn. The knowledge flows between the system and external actors are defined. The whole system is displayed as a single operation.

3.3.2 Use Case diagram

The use case diagram is primarily designed to capture the system 's dynamic aspect. It is used to identify system requirements, including internal and external influences. Most of these requirements are design requirements. Thus, when the system is analyzed to gather its functionality, use cases are prepared and actors are identified. The use case diagram was used to describe the functions provided by the system that have a visible effect on the various actors involved in system use, such as the system administrator, lecturers and students who are key users of the system. The recognition of actors and use cases leads to the system boundary concept, which is to differentiate the tasks carried out by the system and the tasks carried out by its environment.

The case diagram shown in Figure 3.3 demonstrates the definition by using the proposed structure of actors along with their respective activities. The use-case diagram

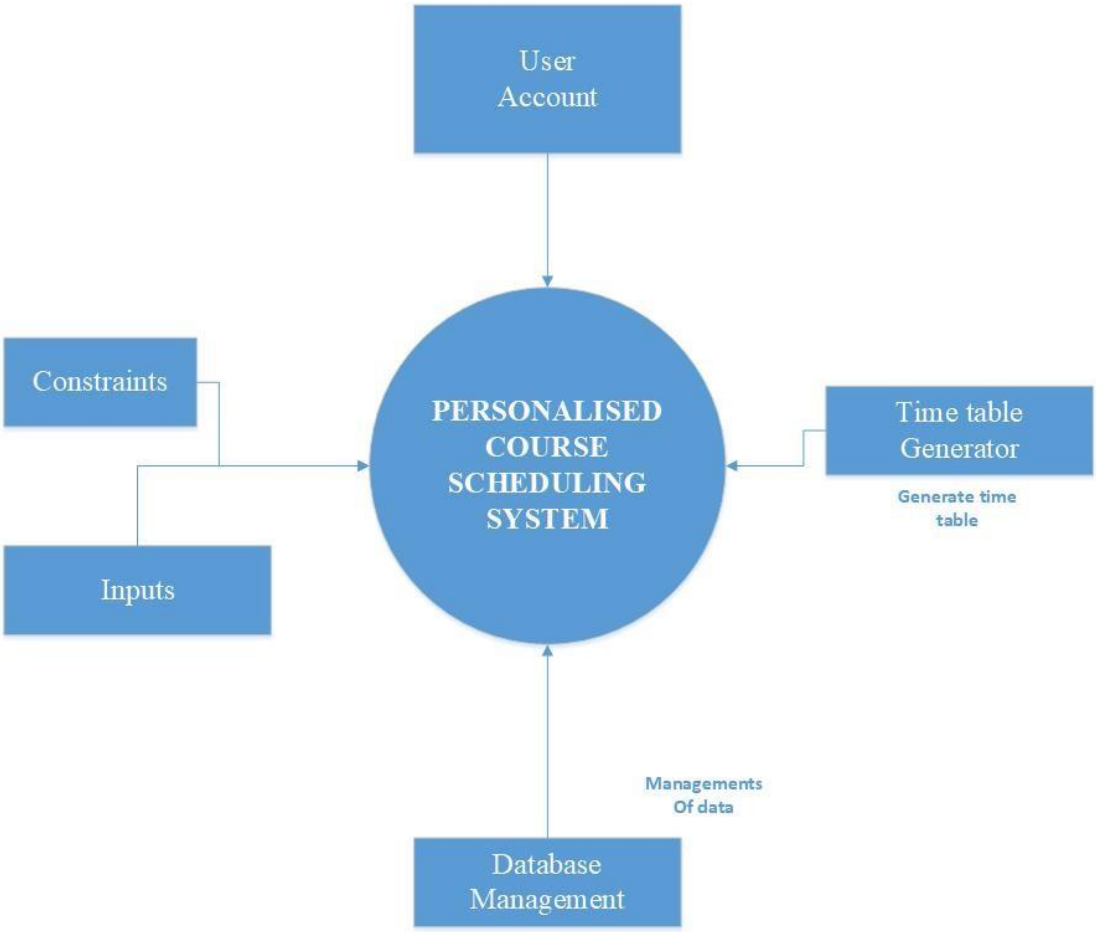

Figure 3. 2: Context Diagram

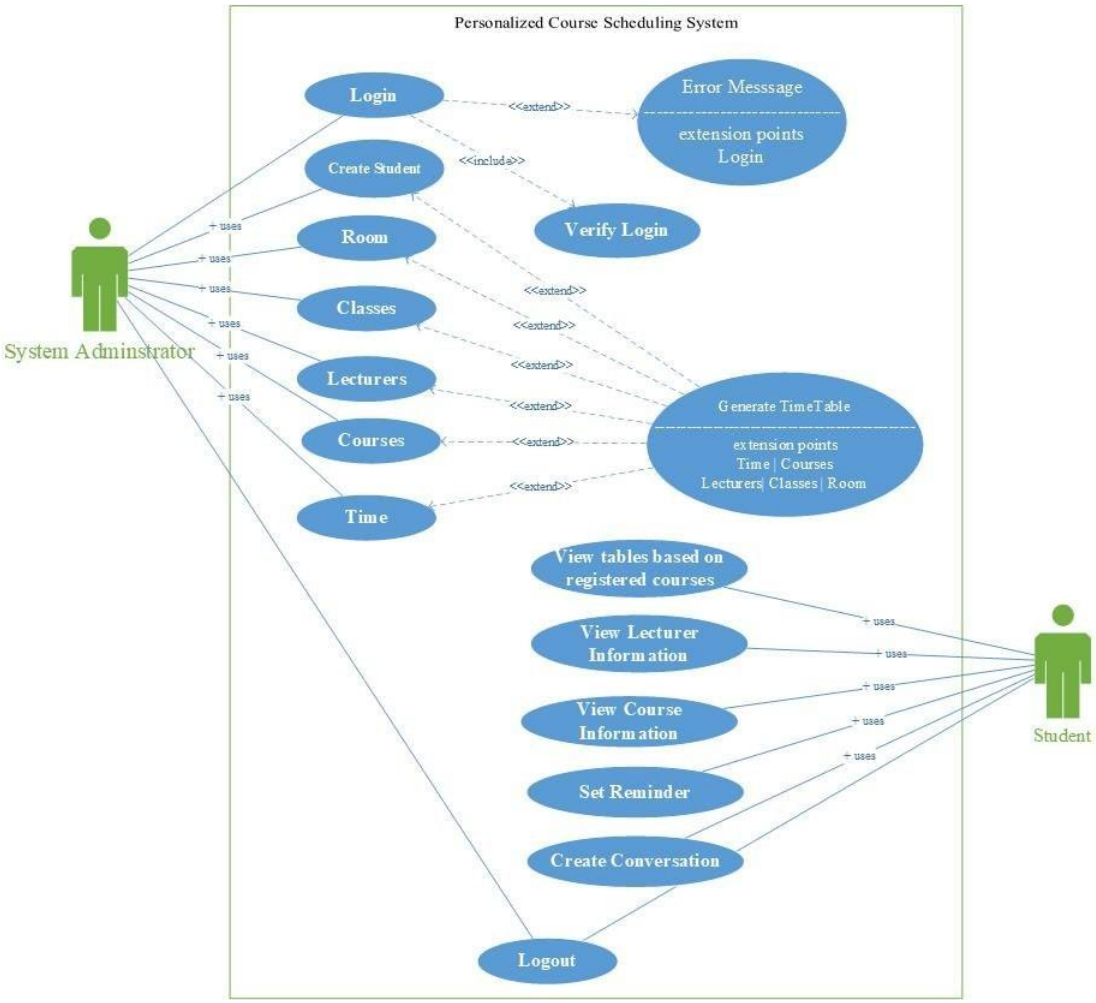

Figure 3. 3: Use Case Diagram

shown in Figure 3.3 shows a description of the actors alongside their respective activities using the proposed system.

i. **System Administrator**: is the super-user of the system. He is responsible for creating access to the system by an authorized user and create course schedules. He is also referred to as the timetable departmental representative. The primary responsibilities of the system administrator are to: add student in a department into the system; create lecturer room in an institute; add lecturer in a department into the system; add Course for a particular department; allocated class for a course in a department and create course schedules

i. **Student**: is the primary user of the system. The primary responsibilities of students include viewing course schedules, viewing information lecturer handling a course; viewing information regarding a course; setting reminder and create conversation.

3.2.3 Sequence diagram

This explains how objects communicate with each other through messages during the execution of a use case or any process. They demonstrate how messages are transmitted and received between objects and the sequence of message transmission as seen in Figure 3.4. It also explains how operations are performed on a time-by-time basis (Onuwa, 2015).

3.2.4 Activity diagram

Activity Diagram explains the sequential movement of business or use case operation. It can also be used for modelling actions that are carried out while an operation is carried out and the results of such actions as shown in Figure 3.5. It explains how actions are mutually related (Onuwa, 2015).

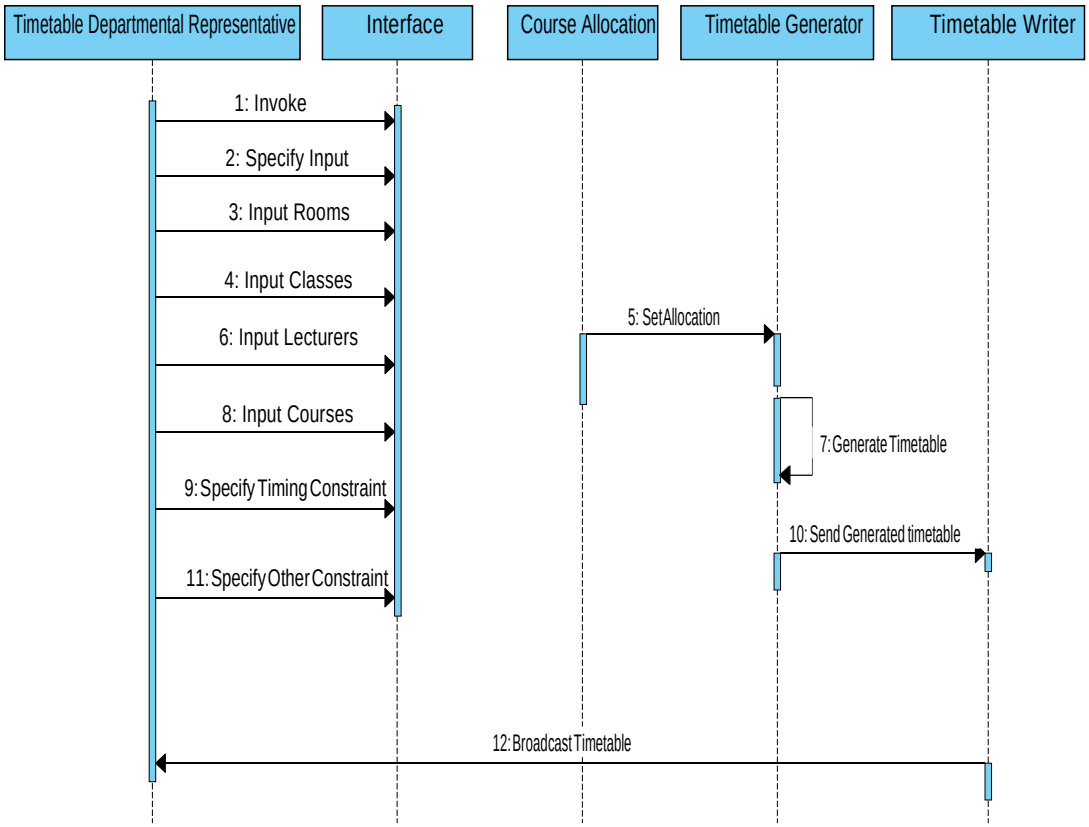

Figure 3.4: Sequence Diagram

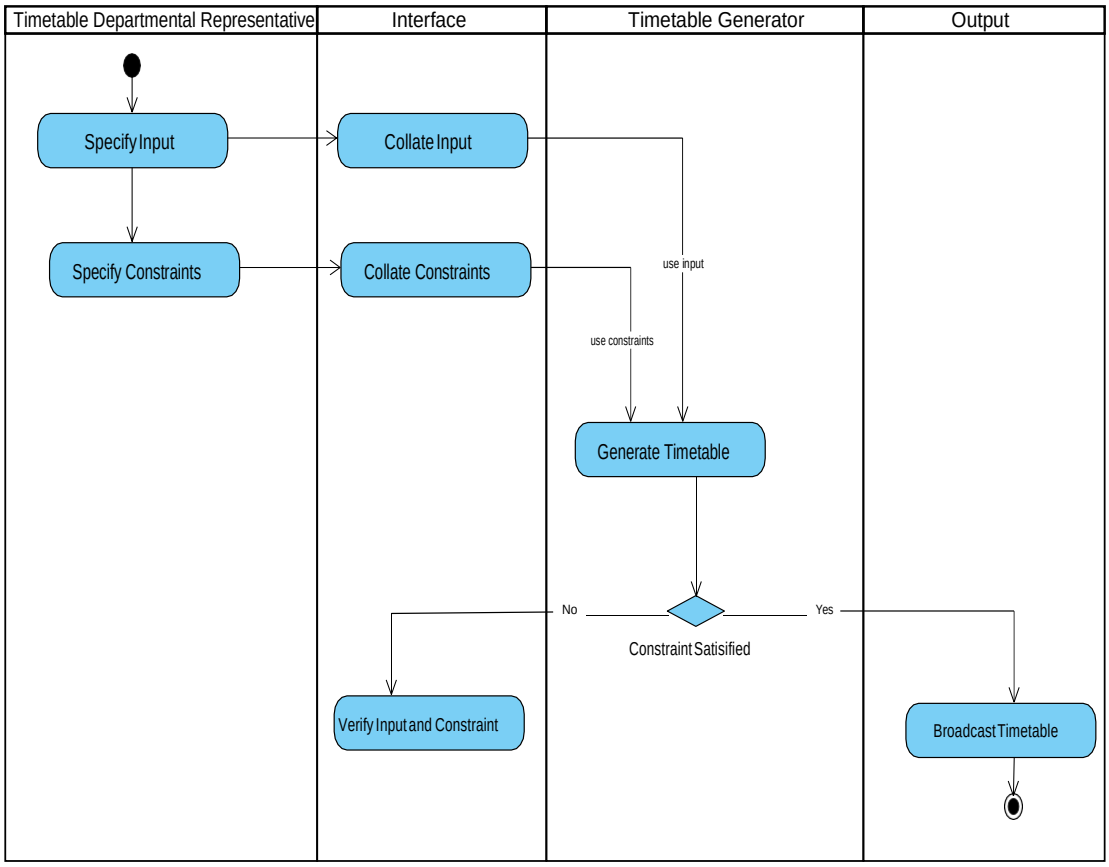

Figure 3.5: Activity Diagram

3.2.5 State Diagram

It is used for modeling a single object's complex behavior. They demonstrate the lifecycle of an object (i.e., the different states an object may assume) and events which cause the object to be transferred between states as shown in Figure 3.6.

3.3 System Implementation Tools

These are the implementation tools needed to carry out the development of an automated course scheduling system. The tools include: Hypertext Markup Language (HTML), Cascading Style Sheet (CSS), Hypertext Preprocessor, React JS, Node JS, Python, Mongodb, Swift, Kotlin and Postman.

a. **Hypertext Markup Language (HTML)** is the standard markup language for documents designed to be displayed in a web browser. Technologies like Cascading Style Sheets (CSS) and scripts like JavaScript can support it. HTML documents are downloaded from a remote server or a local database and returned to multimedia web sites via web browsers. HTML defines the Web page layout semantically and the records are initially used.

b. **Cascading Style Sheets** is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS, along with HTML and JavaScript, is a staple of the World Wide Web. CSS is designed to distinguish formats, colors , and fonts from the displays and text.

c. **Syntactically Awesome Style Sheets** is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets (CSS). The script language itself is SassScript. There are two syntaxes in Sass. The original grammar, known as the "indented grammar," uses Haml-like syntax. The code block and the new line characters for splitting rules are differentiated by indentation. "SCSS" (Sassy

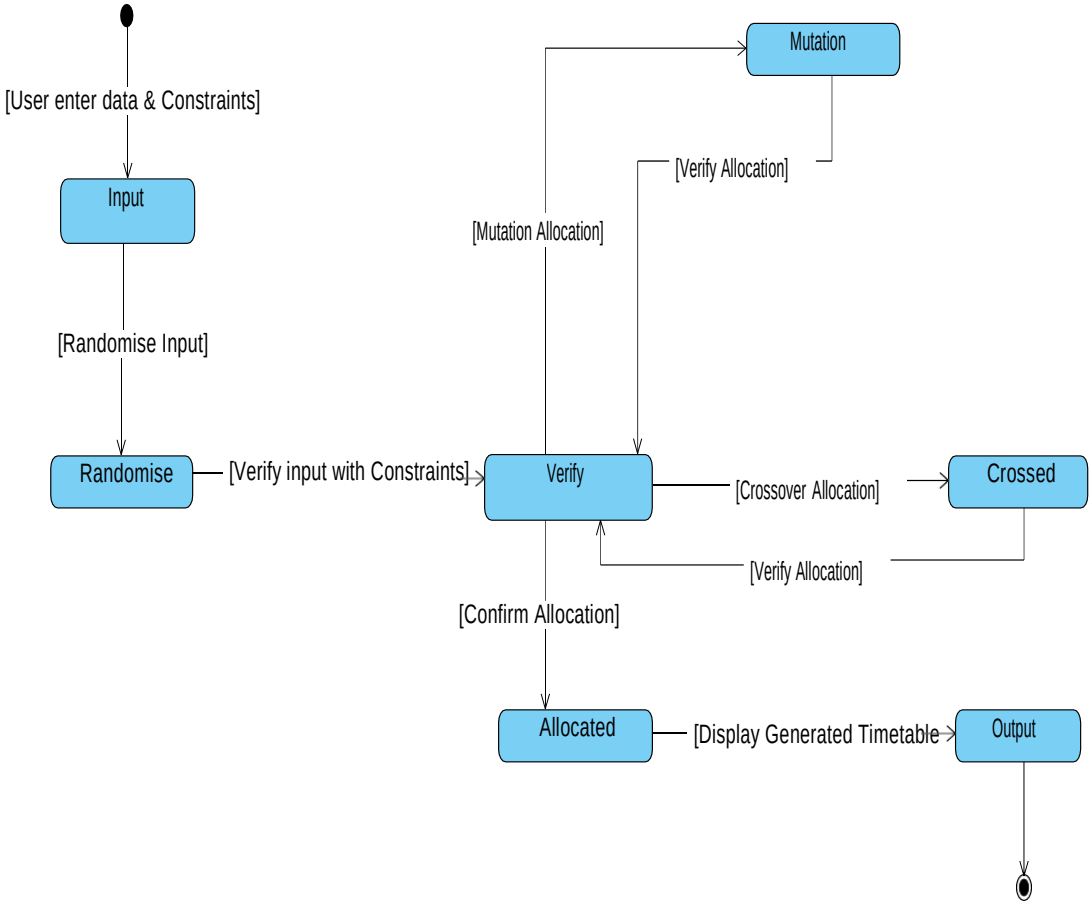

Figure 3.6: State Diagram

CSS) is the newest syntax which uses block formatting like CSS. It uses braces to mark blocks of text and semicolons in order to distinguish rules in a row. The package usually contains the extensions .sass and .scss in the indexed syntax and SCSS format.

d. **Syntactically Awesome Style Sheets** is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets (CSS). The script language itself is SassScript. There are two syntaxes in Sass. The original grammar, known as the "indented grammar," uses Haml-like syntax. The code block and newline characters for splitting rules are differentiated by indentation. "SCSS" (Sassy CSS) is the newest syntax which uses block formatting like CSS. It uses braces to mark blocks of text and semicolons in order to distinguish rules in a row. The package usually contains the extensions.sass and.scss in the indexed syntax and SCSS formats.

e. **React JS** is an open-source JavaScript library designed for user interfaces or UI modules. React is called a "react. "js or ReactJS." The reaction can be seen as a framework for creating single page and Smartphone Apps. Facebook is managed by a group of independent developers and businesses. However, the purpose of React is to render data only to DOMs, which makes it typically possible to use external libraries for state administration and routing Redux and the Router React respectively.

f. **Node JS** is an open-source, cross-platform, JavaScript runtime environment (Framework) that executes JavaScript code outside a web browser. Node.js lets developers code Command Line tools for JavaScript for scripting on a server-side – running server scripts to create complex content on web sites before submitting the file to the web browser of the user. Node.js therefore reflects a 'JavaScript

anywhere' paradigm uniting the development of web applications around a common programming language and not multiple languages for server and client scripts.

g. **Python** is a widely structured, universal language of programming. Its arrangement of language and object-oriented methodology was designed to assist programmers to build simple and functional codes for large and small projects. It is typed and waste stored dynamically. It supports many paradigms of programming, including structured programming (especially procedural), object-orientated and functional. Because of its vast standard library, Python is also defined as a *batteries included* script.

h. **MongoDB** is a document-based storage software that operates across platforms. MongoDB uses json-like documents with optional schemas, known as a nosql database application. MongoDB is developed under the public side of the server license (SSPL) by MongoDB inc. MongoDB is licensed.

i. **Swift** is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. for iOS, iPadOS, macOS, watchOS, tvOS, and Linux. Swift is intended to work with the Cocoa and Touch architectures of Apple and with a broad selection of current Objective C code written for products from Apple. It's built into the LLVM compiler architecture of the open source community and has been used in Xcode since version 6 in 2014. Apple's runtime libraries have been designed to run C, Objective-C , C++ and SWIFT within a single program.

j. **Kotlin** is a static cross-platform programming language for general purposes with a method of inference. Kotlin has been developed for complete interoperability with Java, while JVM 's regular library implementation of Kotlin focuses on the

Java Class library. Kotlin is primarily targeting the JVM, but also compiles to JavaScript (for example, for frontend web applications using React or Native code via LLVM). JetBrains funds language production expenses, while Kotlin Foundation preserves the Kotlin trademark. its preferred language for Android app developers.

k. **Postman** is a lightweight API platform that integrates rapidly into the CI / CD pipeline. In 2012, the API workflow for testing and improving began as a side project by Abhinav Asthana. it is now one of the most popular API testing development tools, developers can build, monitor, upload, and record APIs easily using this tool.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This section presents the results and the discussion of this study which involved the design and development of a personalized course scheduling system. The chapter presents the implemented database of the system using the JavaScript Object Notation (JSON) alongside the system interface that was implemented using web-based development technologies such as the HTML, CSS and React JavaScript for the development of the system interface. The system interface for the secondary user which is the student was implemented using Mobile technologies such as Kotlin and Swift. Testing and evaluation of system preformation's was also an integral part in this section.

4.2 Implementation of System Database for Course scheduling System

Figure 4.1 shows a description of the database that was implemented for this study in order to store and retrieve information required by the proposed system. As a result of this, a database was implemented called the coursekit which consisted of eleven (11) database collections which were required for managing the various types of information stored and manipulated by the system. The results of the implemented collection courses as shown in Figure 4.2 was used to manage information about the courses that are being taken in that institution. The collection holds an array of documents, a document in mongoDB is a data structure composed of field and value pairs.

The values of fields may include other documents, arrays and arrays of documents. A document in the courses collection had values and keys such as: _id, a

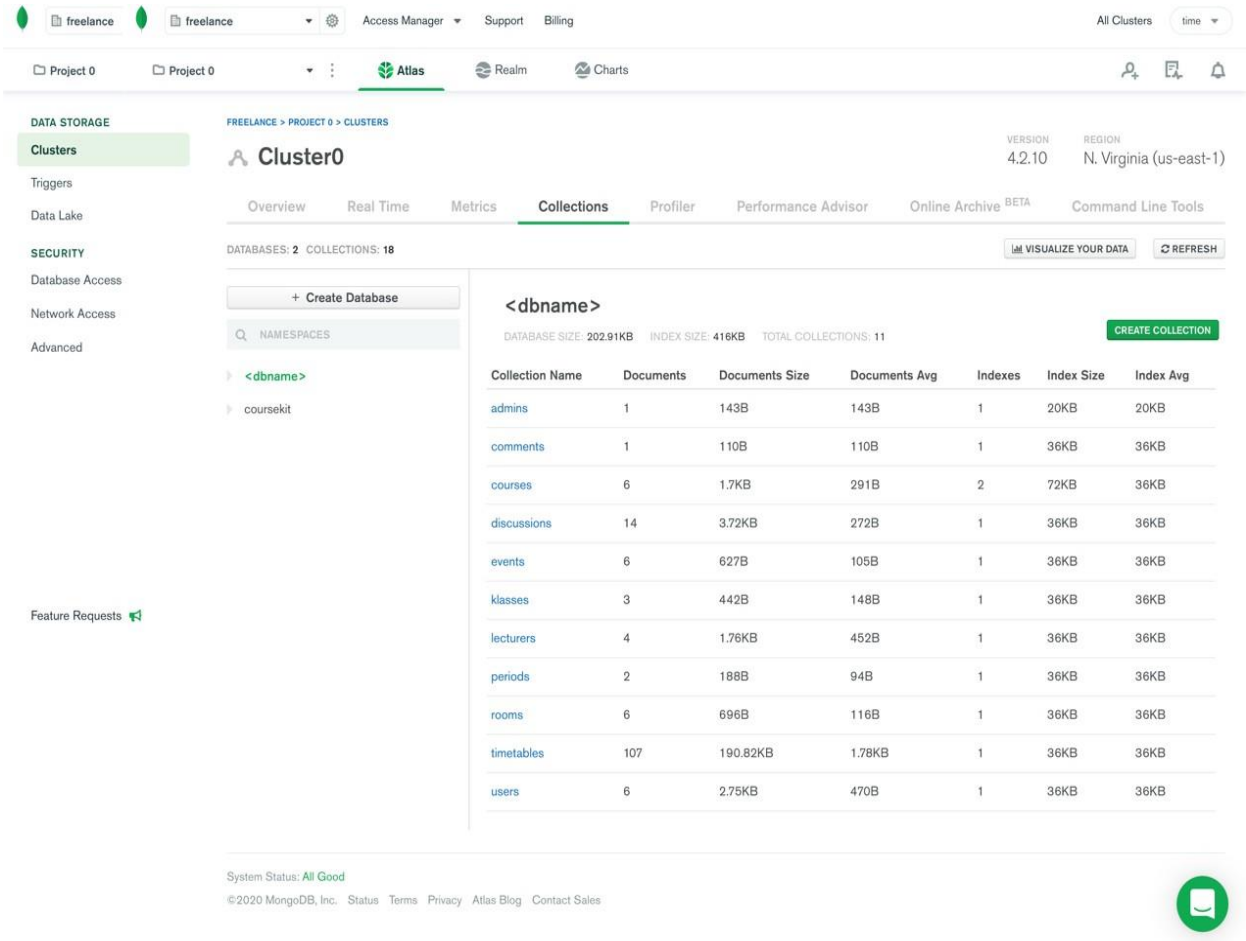

Figure 4.1 Database *coursekit* showing its collections

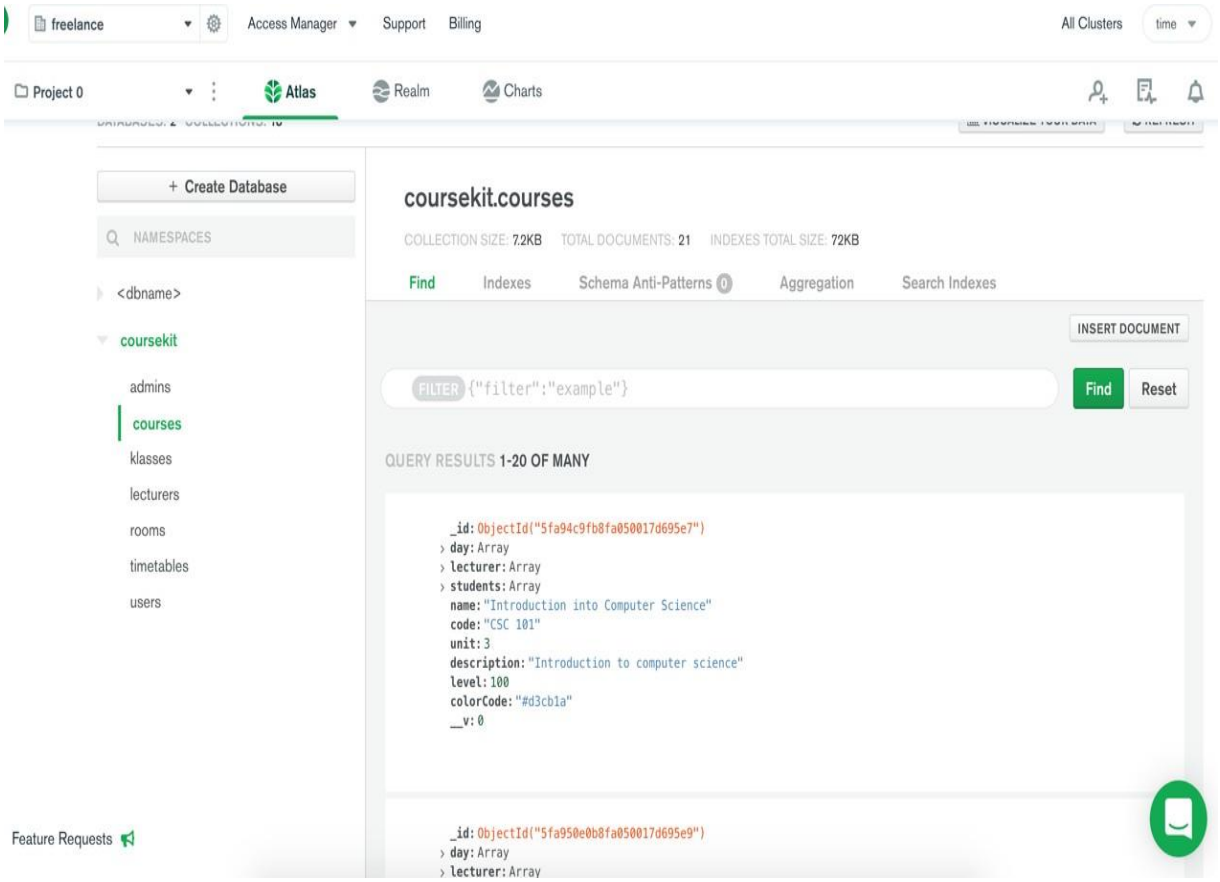

Figure 4.2: Collection *coursekit.courses* showing the list of courses and attributes

unique id for each document, name, the name of the course, code, unit, description, level and colorCode. The results of the implemented lecturer's collection as shown in Figure 4.3 was used to manage information about the lecturers that are in that institution. A document in the lecturer's collection has values, and keys such as: _id, courses, name, email, educational_bg, phone_no, office_no, ranking, degree, areaOfSpec, image etc. The results of the implemented courskit.rooms collection as shown in Figure 4.4 was used to manage information about the rooms that are in that institution. A document in the room's collection has values, and keys such as: _id, name, capacity, createdAt, and updatedAT

Figure 4.5 shows the result of the implementation of courskit.timetable collection which was used to manage information about the timetable for a department. A document in the timetables' collection has values and keys such as: _id, courses, uuid, createdAt, updateAt, _v, current_progress, name, session, and total progress. Figure 4.6 shows the result of the implementation of courskit.users collection which was used to manage information about the students in a department. A document in the users' collection has values and keys such as: _id, resetToken, image, maxUnit, minUnit, courses, firstname, lastname, email, dob, level, matric, password, role, createdAt, updateAt, _v, current_progress, name, session, and total progress.

The results of the implemented courskit.klasses collection as shown in Figure 4.7 was used to manage information about the classes in a department. A document in the classes's collection has values, and keys such as: _id, Courses, name, AcademicPeriod, Meeting, Population, UnavilableRooms , and _v. Figure 4.8 shows the result of the implementation of courskit.events collection which was used to manage information about the events in a department. A document in the events' collection has values and keys such as: _id, user, name, date, and time.

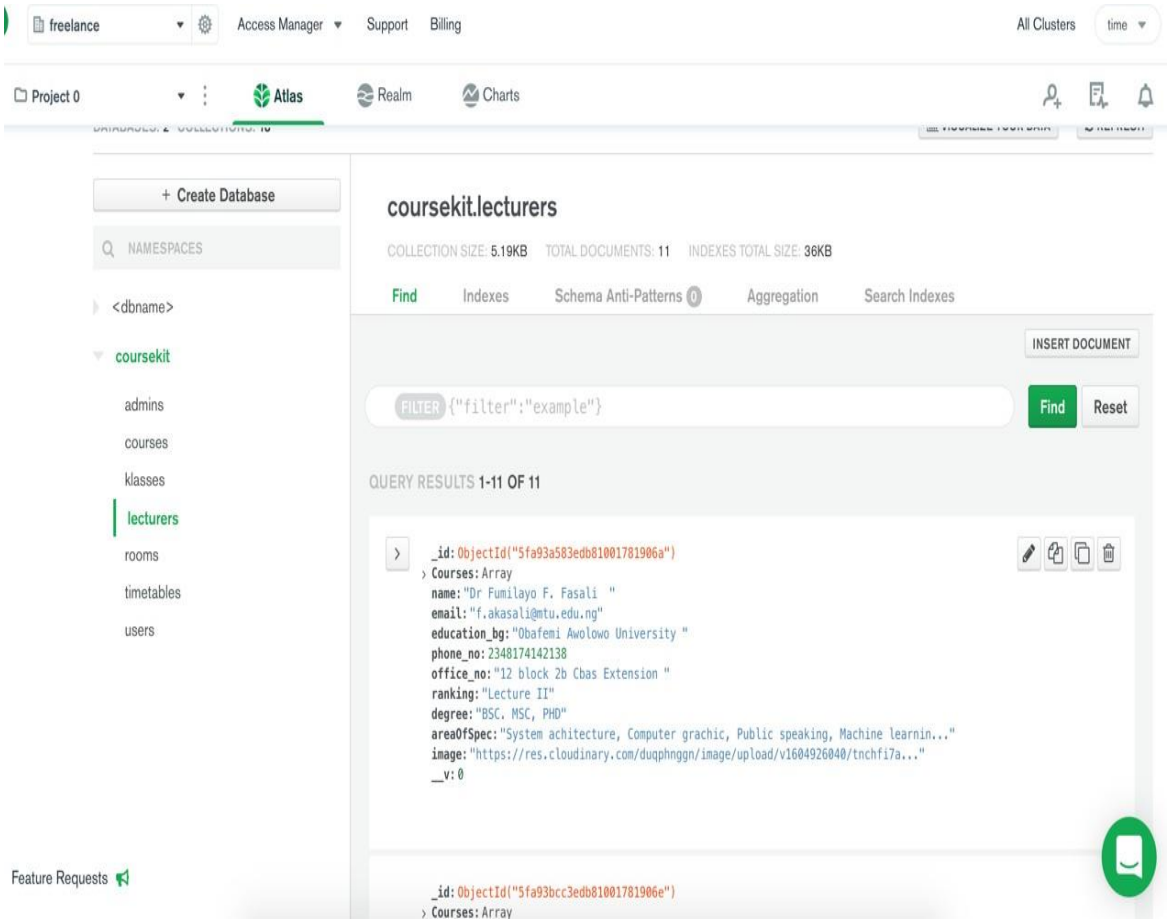

Figure 4.3: Collection coursekit.lecturers showing the list of lecturer and attributes

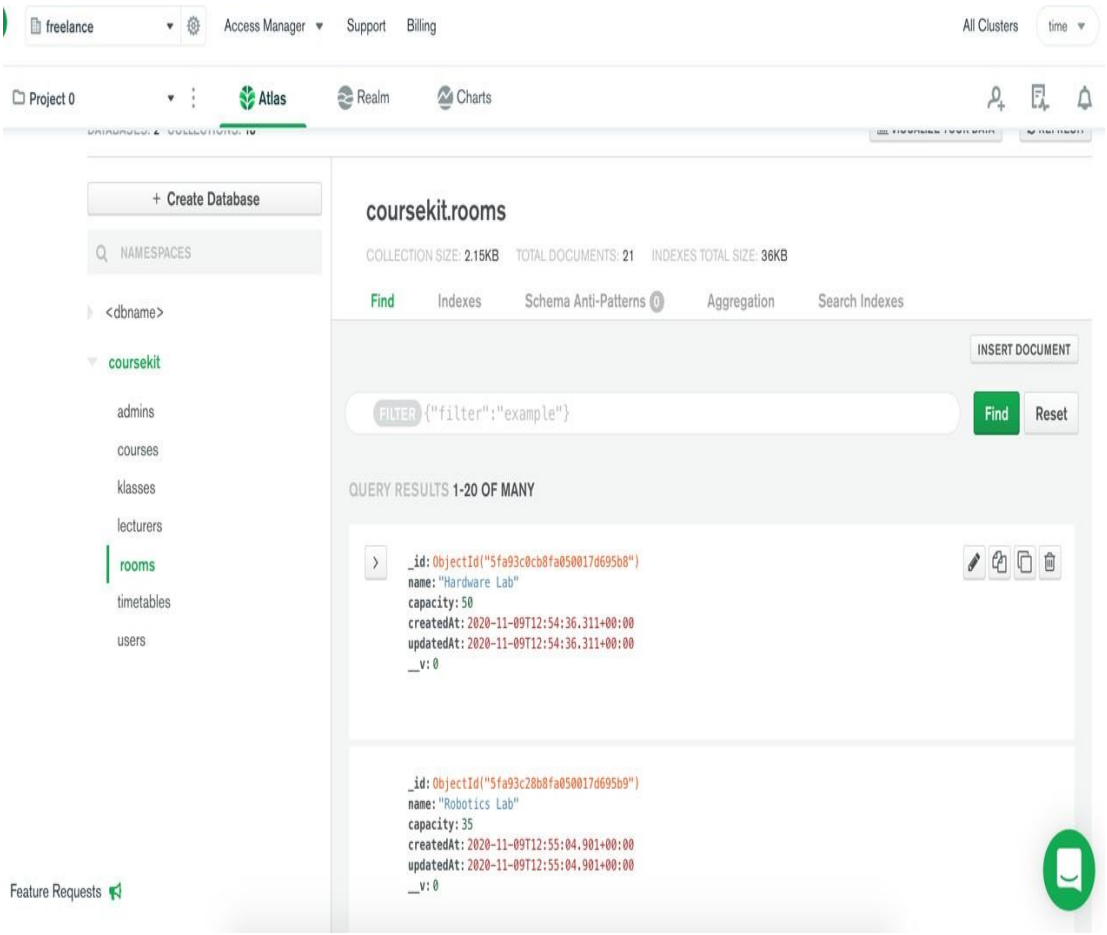

Figure 4.4: Collection *coursekit.rooms* showing the list of rooms and attributes

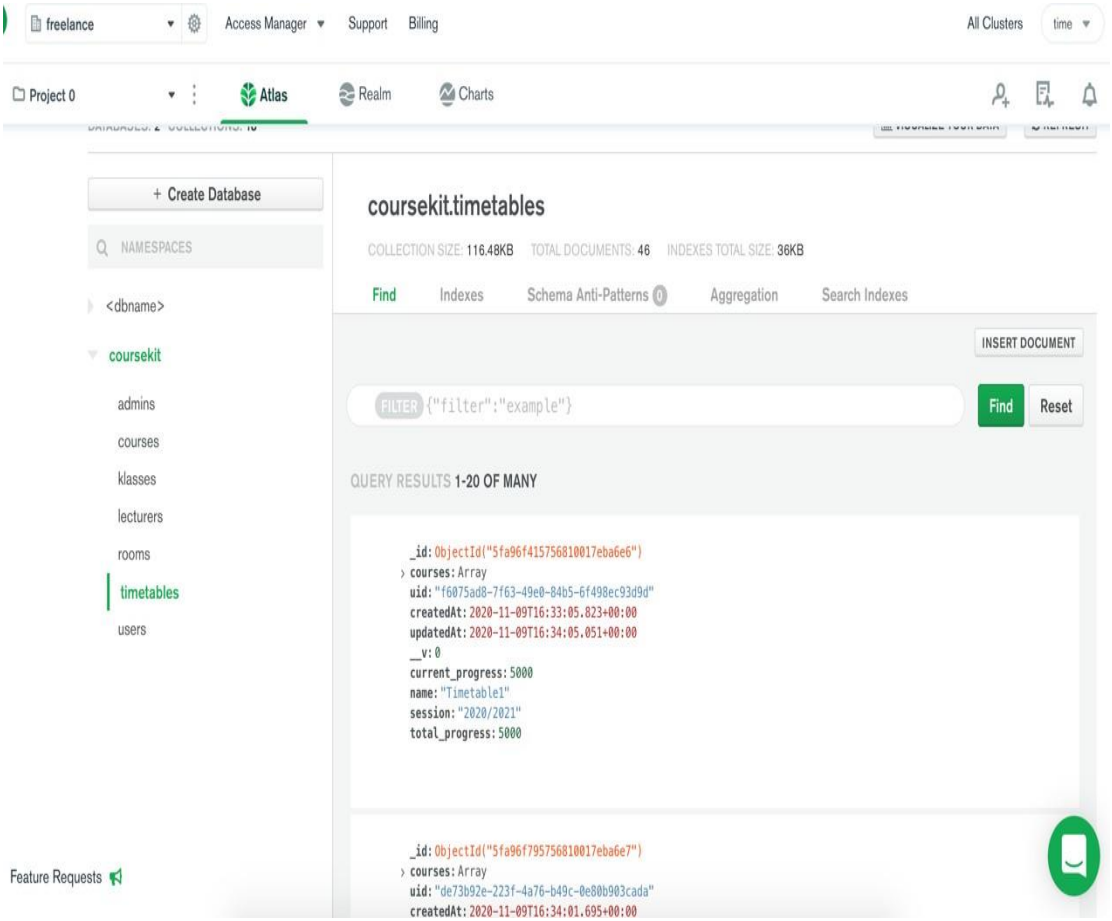

Figure 4.5: Collection *coursekit.timetables* showing the list of timetables and attributes

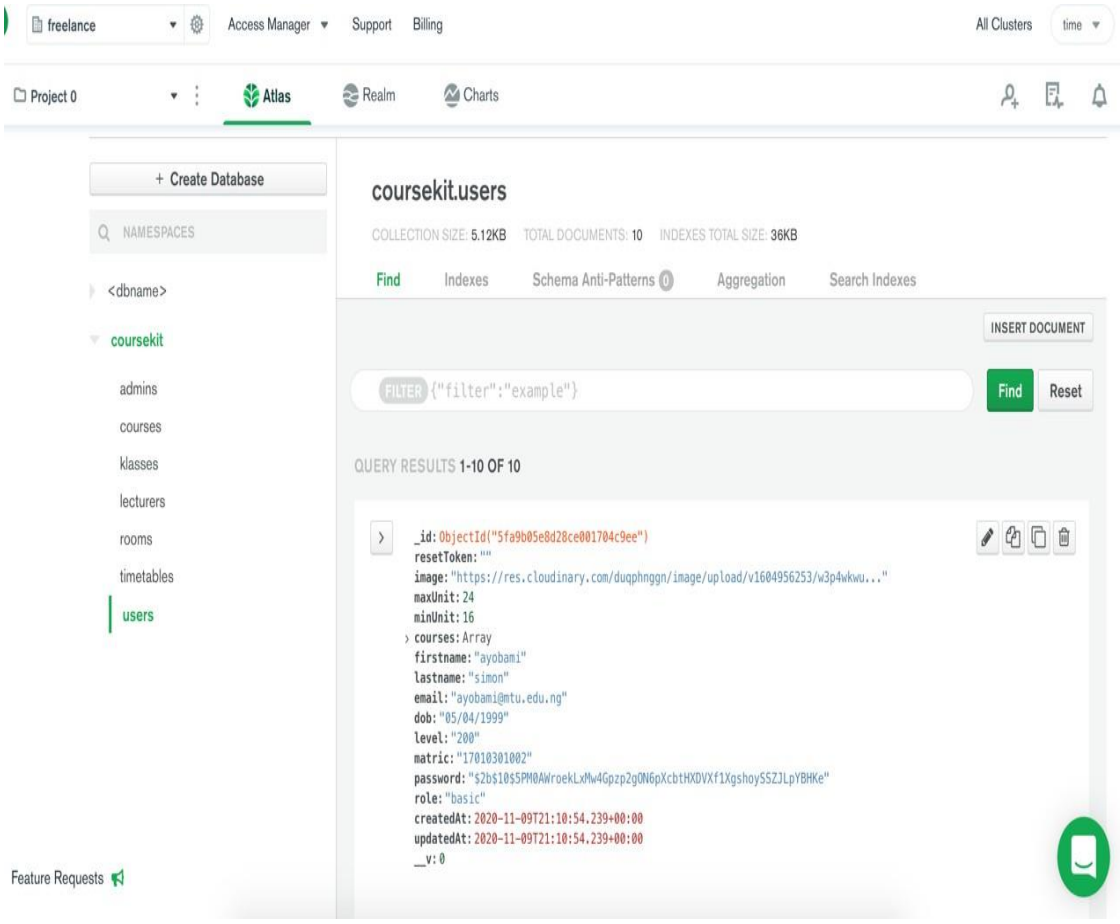

Figure 4.6 Collection *coursekit.users* showing the list of students and attributes

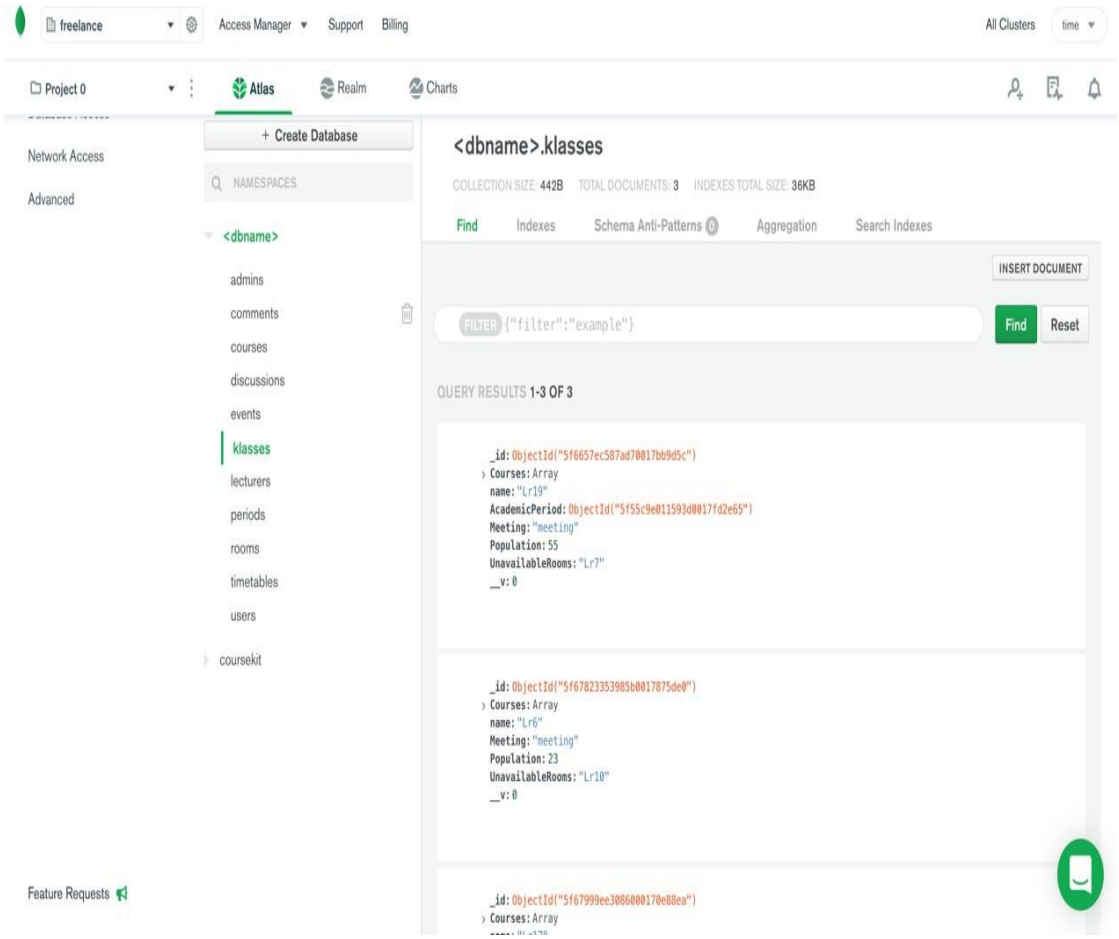

Figure 4.7: Collection *coursekit.Klasses* showing the list of clasess and attributes

Figure 4.9 shows the result of the implementation of courskit.discussions collection which was used to manage information about the discussion in a level. A document in the discussion's collection has values and keys such as: _id, user, comments, title, details, createdBy, createdAt, and updatedBy. Figure 4.10 shows the result of the implementation of courskit.comments collection which was used to manage information about the comments in a level. A document in the comment's collection has values and keys such as: _id, message, userId, createdAt, and updatedBy. Figure 4.11 shows the result of the implementation of courskit.admin collection which was used to manage information about the admins that are in an institution. A document in the comment's collection has values and keys such as: _id, image, adminNumber, and password.

4.3 Result of the User Interface of System Implementation

Following the presentation of the results of the implementation of the system database using Monogdb the results of the implementation of the system using web-based technologies are presented. Figure 4.12 shows the system Login page also called the Home Page open opening the system URL for departmental timetable representatives. The departmental timetable representatives are required to provide their admin number issued to them and passwords which is used by the system to determine the session which is to be created following user access authentication by the system. Figure 4.13 shows the interface of the dashboard of the system administrator upon providing his username and password to the system.

The results of this interface show the different information stored on the system so far such as number of lecture rooms, number of courses offered in a department and

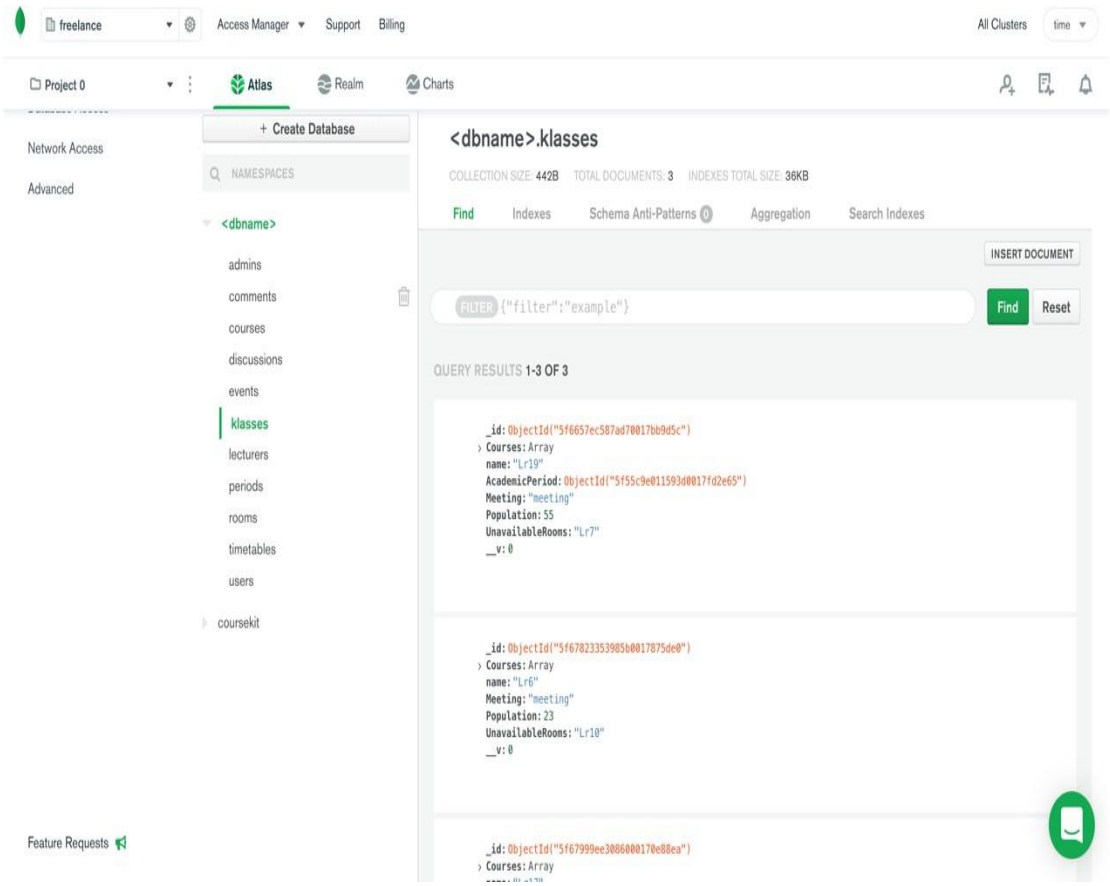

Figure 4.8: Collection coursekit.discussions showing the list of events and attributes

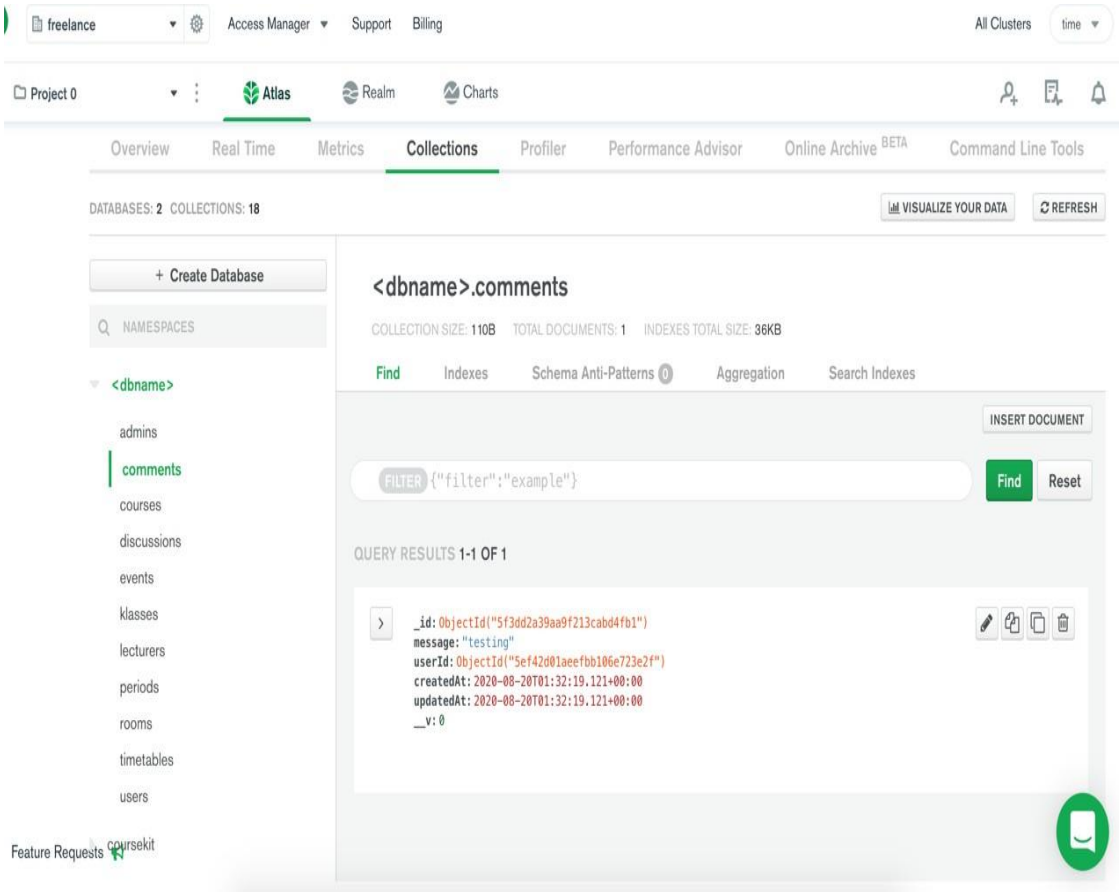

Figure 4.9: Collection coursekit.comments showing the list of comments and attributes

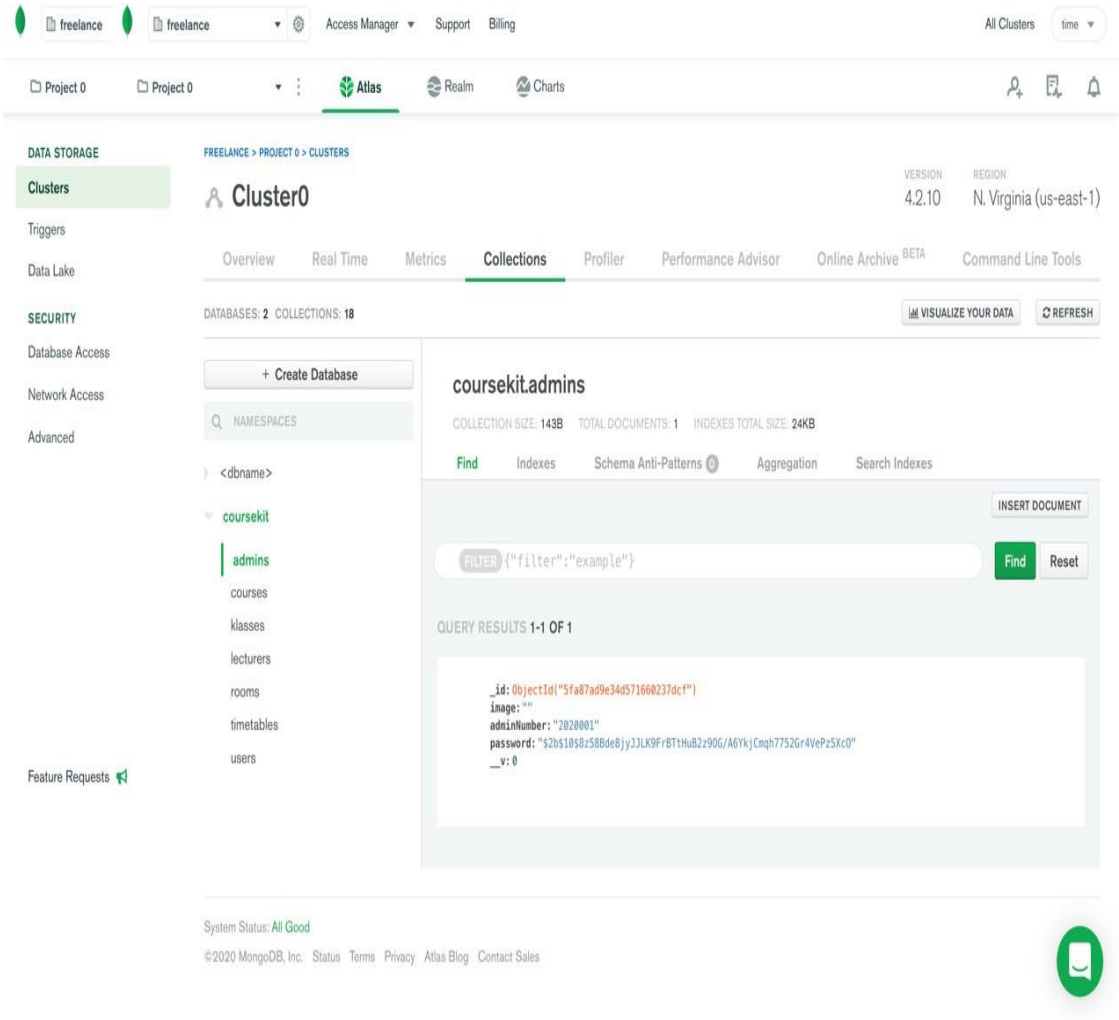

Figure 4. 10: Collection *coursekit.admin* showing the list of admin and attributes

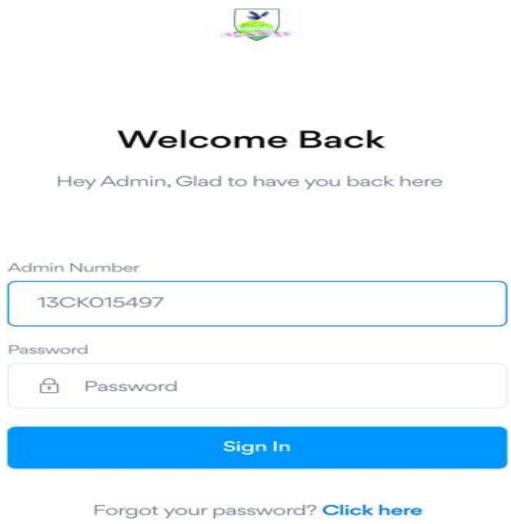

Figure 4.11: Screenshot of the Login Page of the Admin Interface

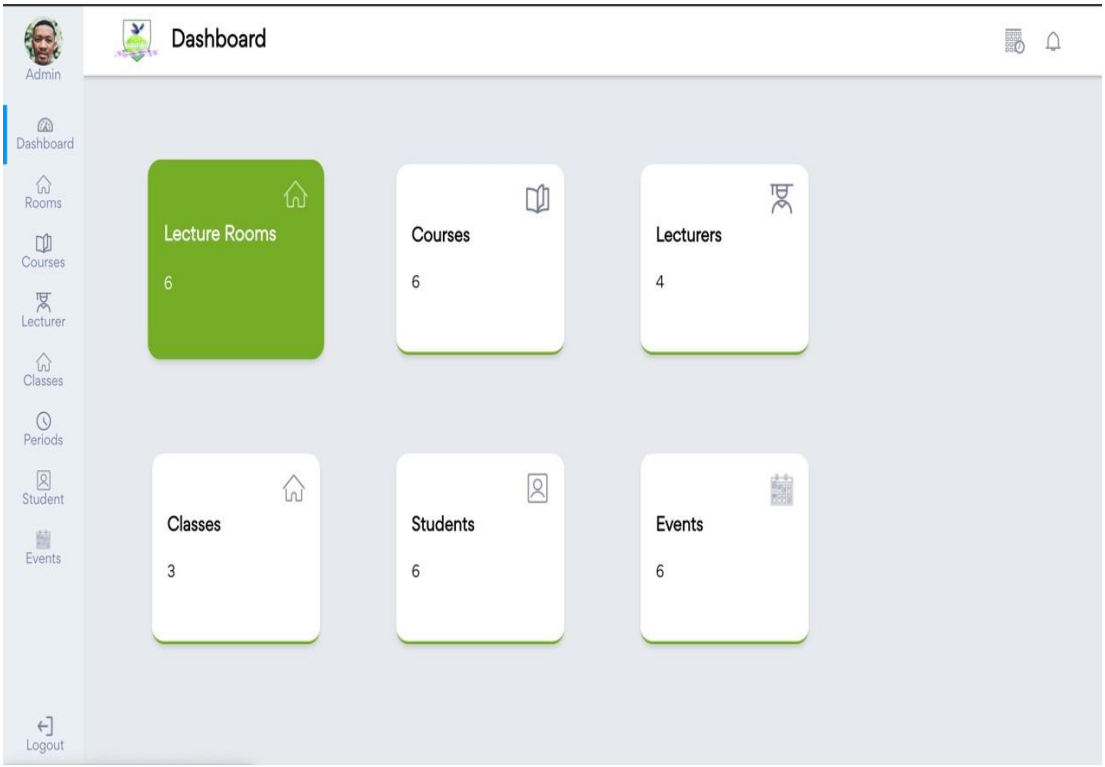

Figure 4.12: Screenshot of System Admin dashboard upon login

number of lecturers available, number of classes, number of students in a department and number of events.

Figure 4.14(a) shows the results of the interface when an administrator is to create a new room to the system which requires the name of the room and the capacity of the room. Figure 4.14(b) shows the result of the interface is required by the administrator for viewing information about existing room in a department. Thus, this interface shows the requirement of details such as name of the rooms, room capacity and action for editing and deleting an existing room.

Figure 4.15(a) shows the results of the interface when an administrator is to create a new course to the system which requires the name of the course, course code, course unit, level, and the capacity of the room. Figure 4.15(b) shows the result of the interface is required by the administrator for viewing information about existing courses in a department. Thus, this interface shows the requirement of details such as name of the course, course code, course unit, number of students offering a course and action for editing and deleting an existing course.

Figure 4.16(a) shows the results of the interface when an administrator is to create a new lecturer to the system which requires the name of the lecturer, image, email, education background, phone number, office number, ranking, degree and area of specification. Figure 4.16(b) shows the result of the interface is required by the administrator for viewing information about existing lecturer in a department. Thus, this interface shows the requirement of details such as name of the lecturer, courses handle, and action for editing and deleting an existing lecturer. Figure 4.16(c) shows the result of the interface is required by the administrator for viewing information about

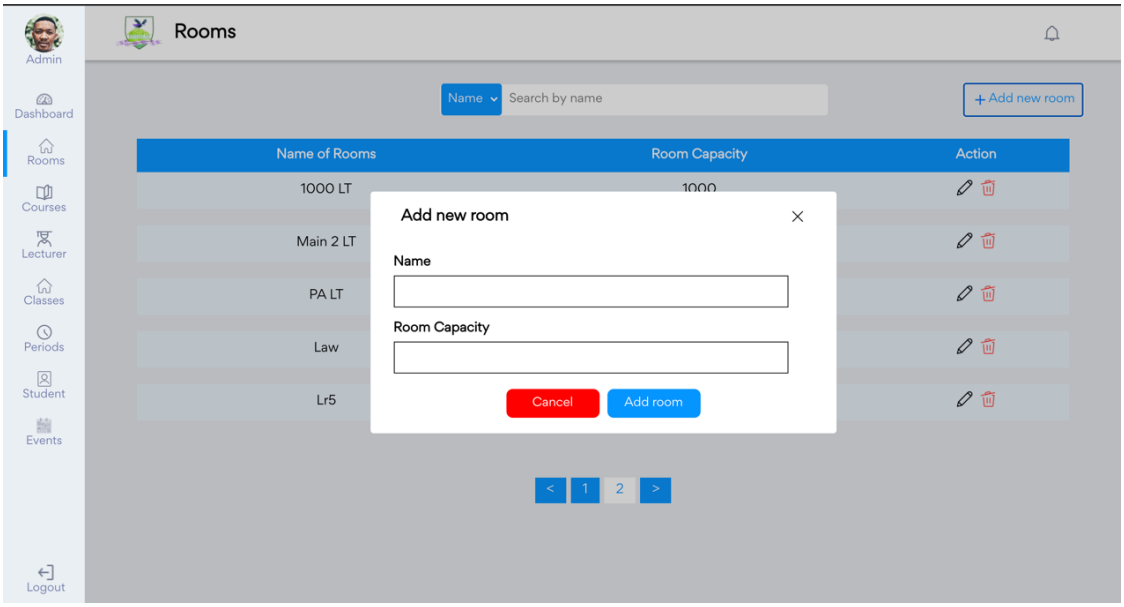

Figure 4.13(a): Screenshot of Admin Interface for creating a new Room

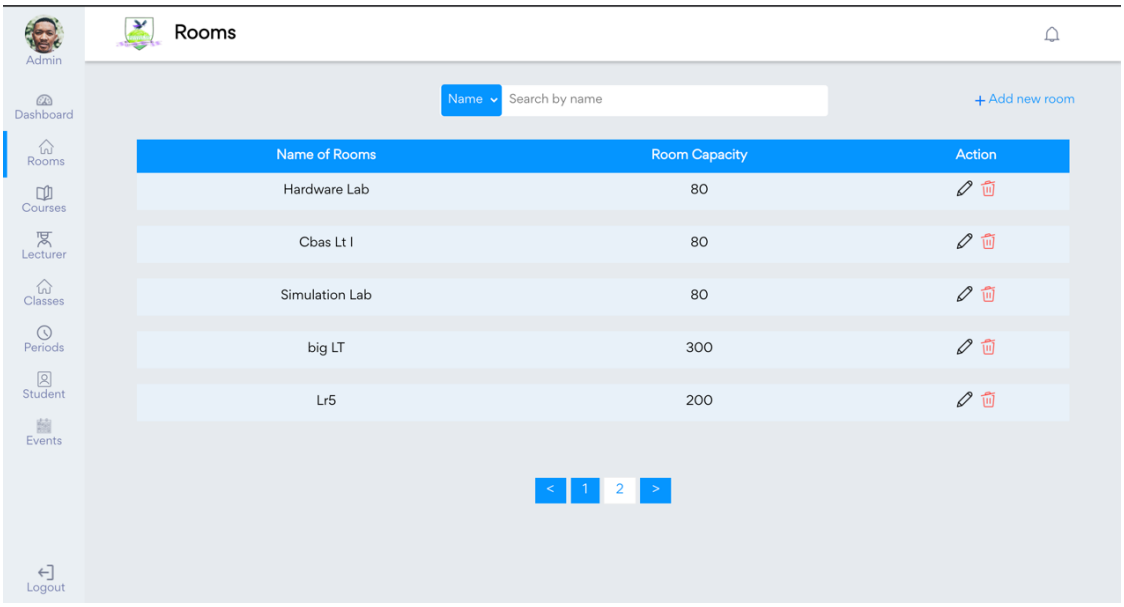

Figure 4.14(b): Screenshot of Admin Interface for managing existing Rooms

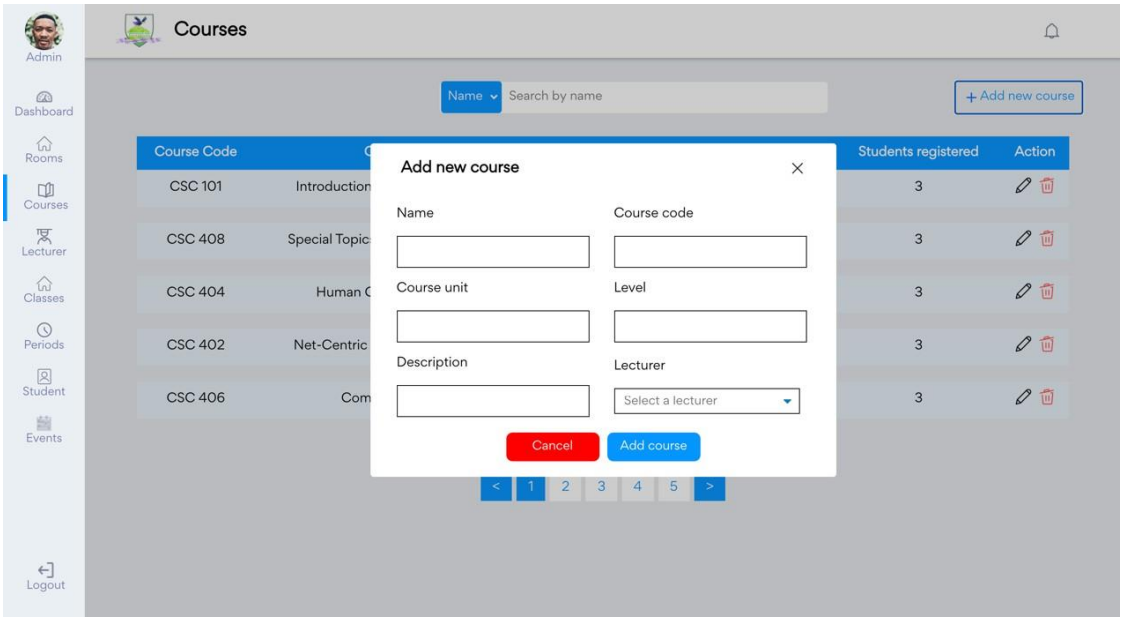

Figure 4.15(a): Screenshot of Admin Interface for creating a new Course

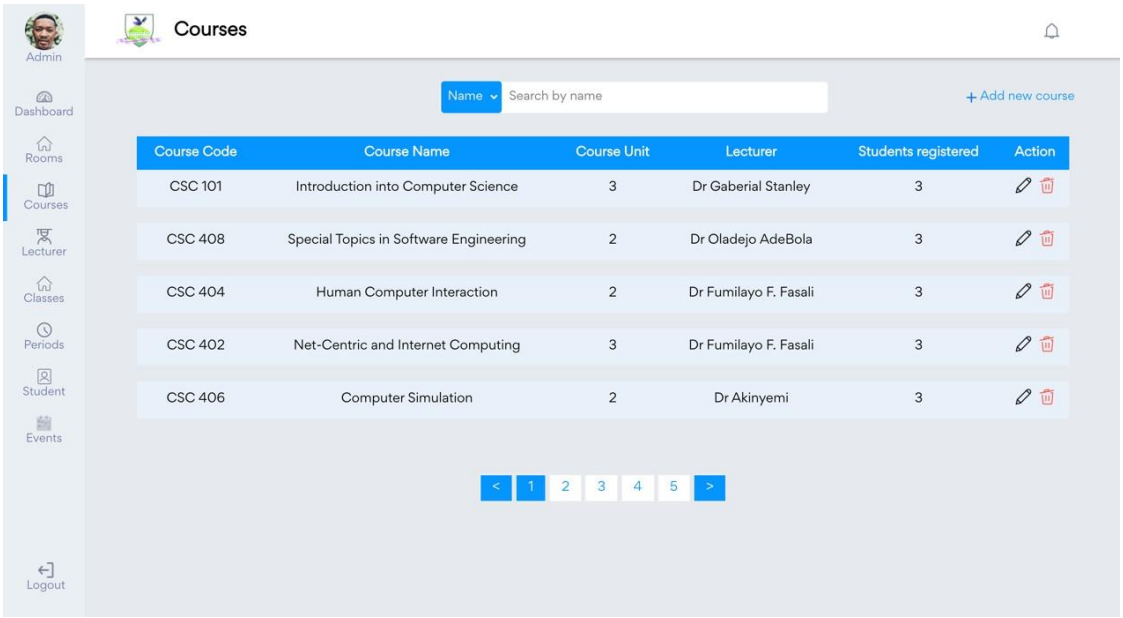

Figure 4.15(b): Screenshot of Admin Interface for managing existing Courses

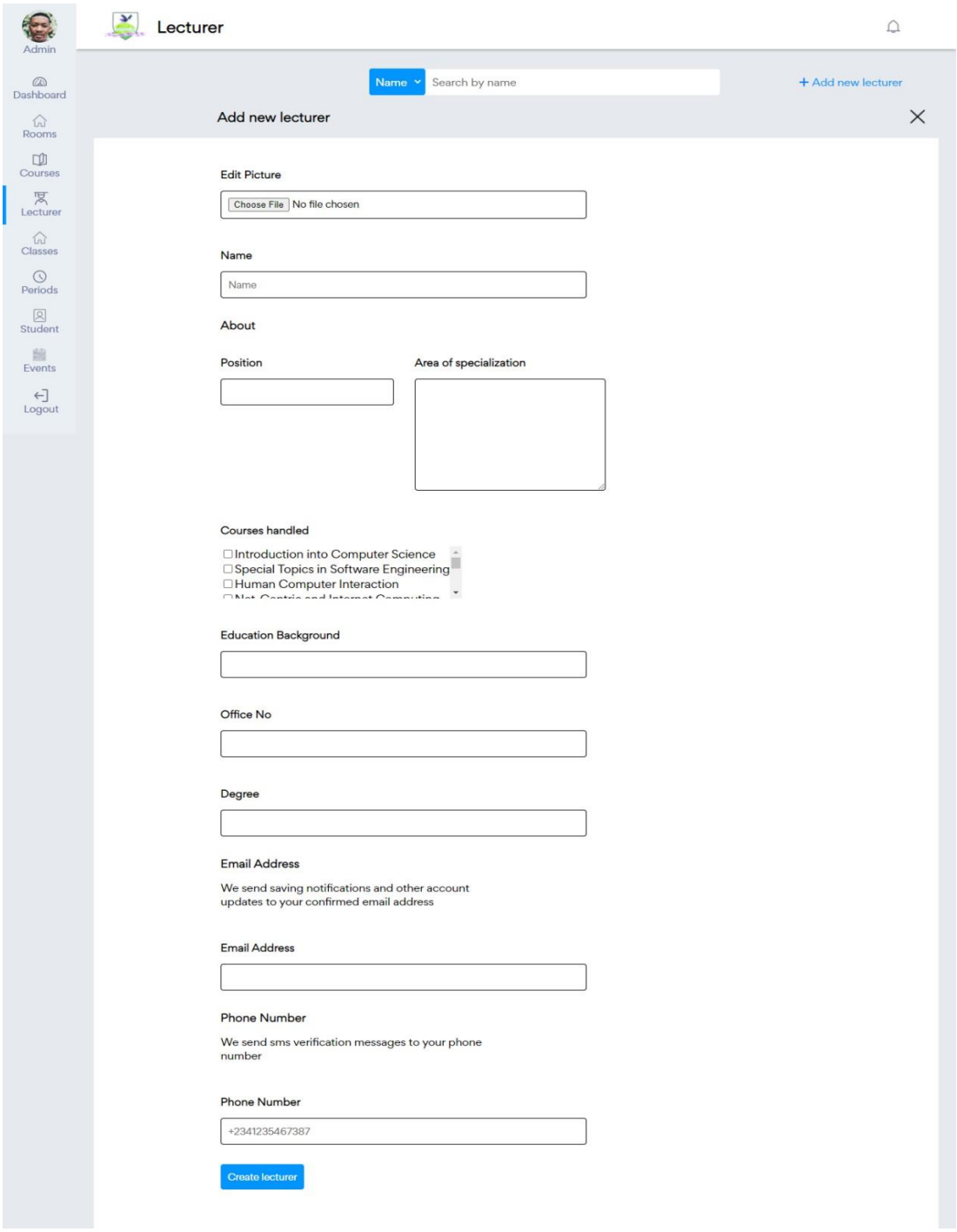

Figure 4.16(a): Screenshot of Admin Interface for creating a new Lecturer

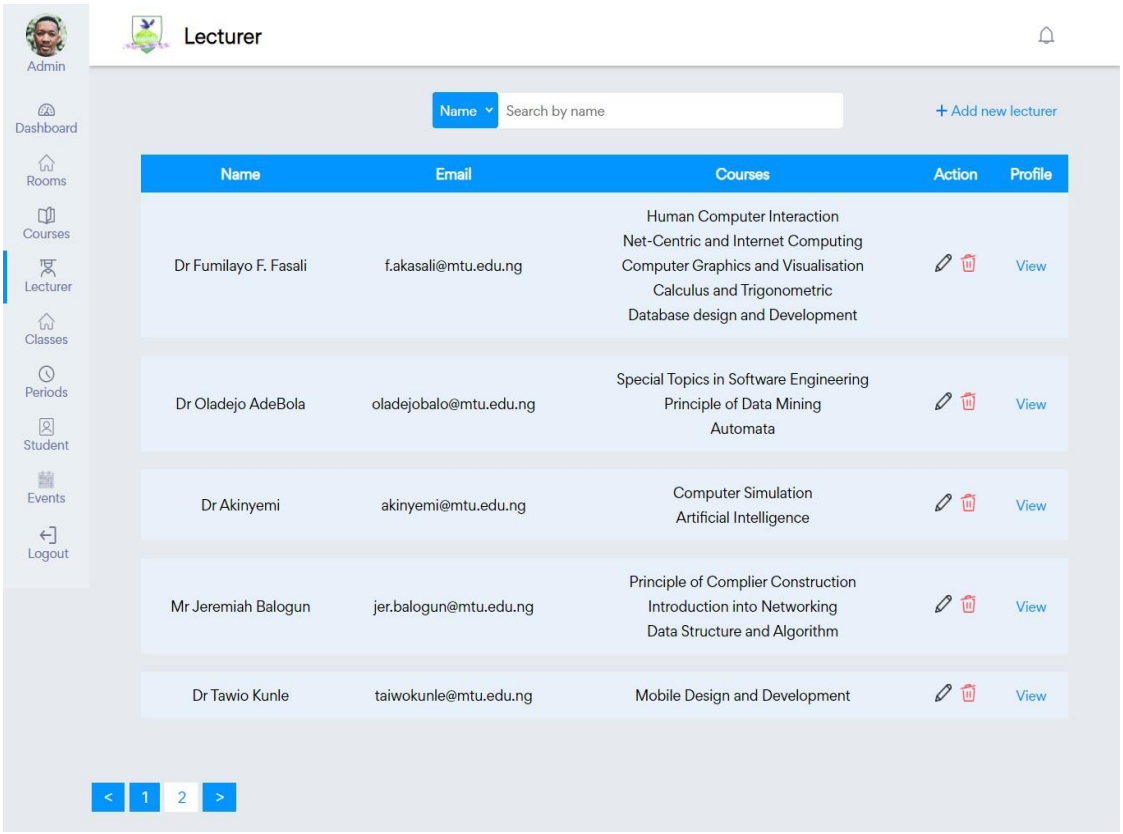

Figure 4.16(b): Screenshot of Admin Interface for managing existing Lecturers

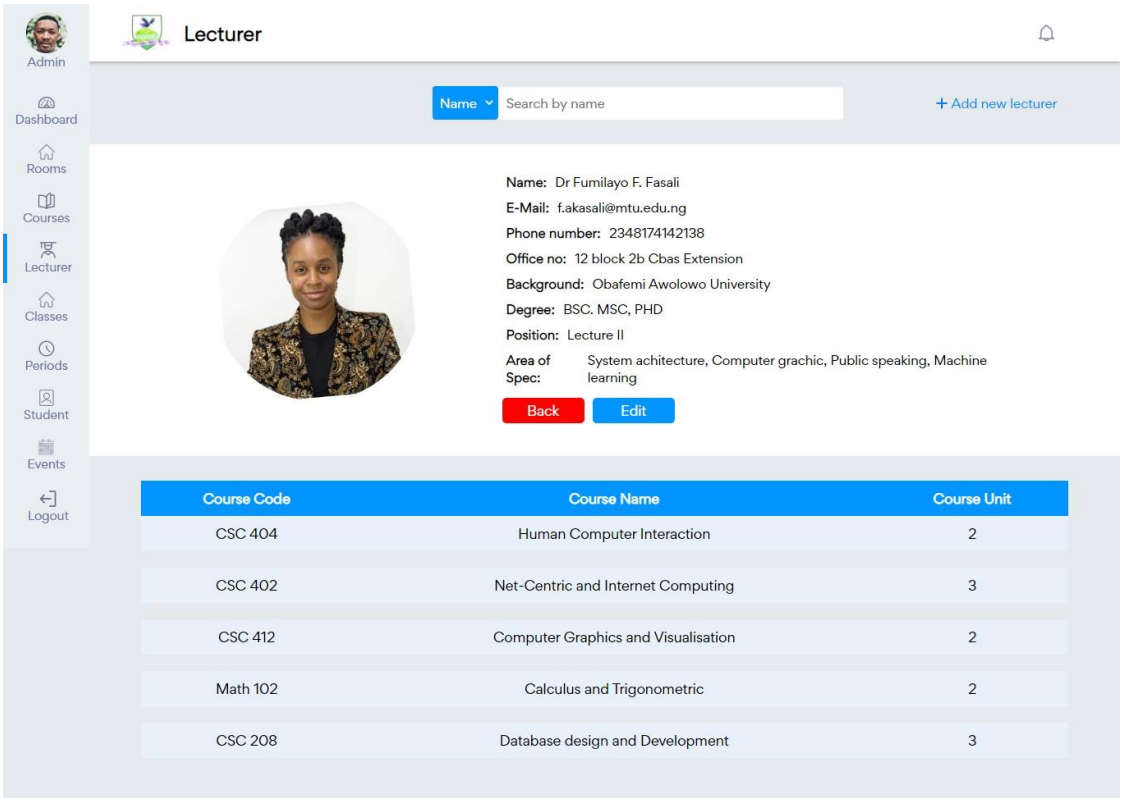

Figure 4.16(c): Screenshot of Admin Interface for viewing a lecturer information

a lecturer in a department. Thus, this interface shows the requirement of details such as name of the lecturer, image, educational background, area of specification, office number, phone number, email address, position of the lecturer and, courses handle.

Figure 4.17(a) shows the results of the interface when an administrator is to create a new class to the system which requires the name of the class, the course assigned for the class, the population for the class, and the unavailable lecture room. Figure 4.17(b) shows the result of the interface is required by the administrator for viewing information about existing classes in a department. Thus, this interface shows the requirement of details such as name of the class, class size, courses, unavailable rooms and action for editing and deleting an existing class.

Figure 4.18(a) shows the results of the interface when an administrator is to create a new student to the system which requires the name of the student, student image, full name, matriculation number, level, department, Date of birth and, email address. Figure 4.18(b) shows the result of the interface is required by the administrator for viewing information about existing student in a department. Thus, this interface shows the requirement of details such as name of the student, email, courses, view profile and action for editing and deleting an existing student. Figure 4.18(c) shows the result of the interface is required by the administrator for viewing information of student in a department. Thus, this interface shows the requirement of details such as name of the student, email, courses, view profile and action for editing and deleting an existing student.

Figure 4.19(a) shows the results of the interface when an administrator is to create a new timetable to the system which requires the name of the timetable, academic session, and selected days. Figure 4.19(b) shows the result of the interface is required by the administrator for viewing information of a timetable. Thus, this interface shows

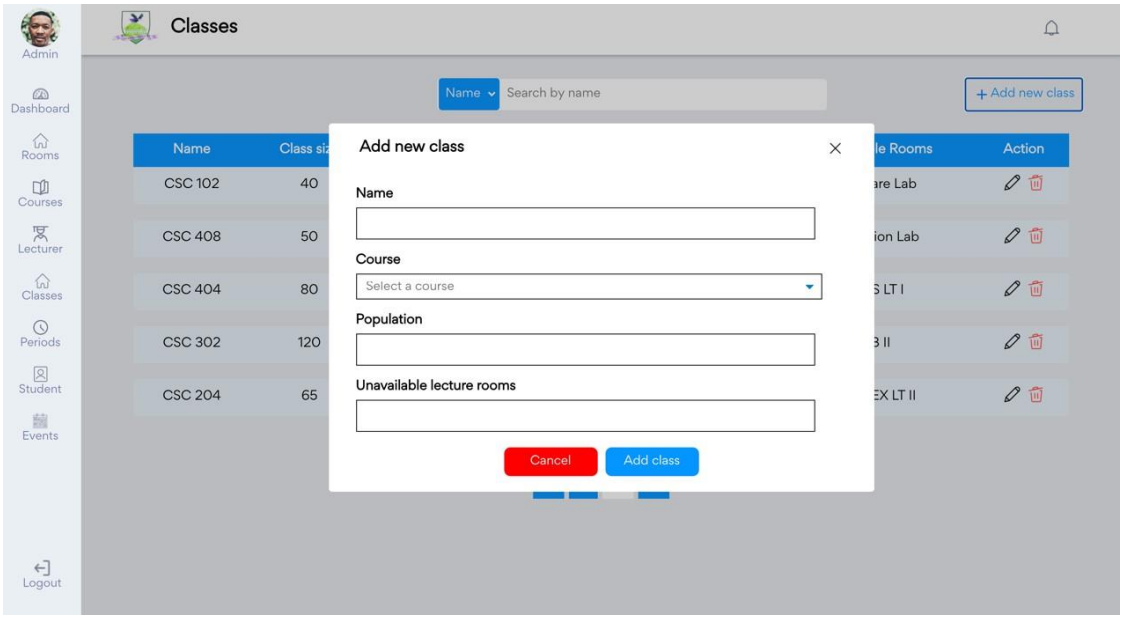

Figure 4.17(a): Screenshot of Admin Interface for creating a new Class

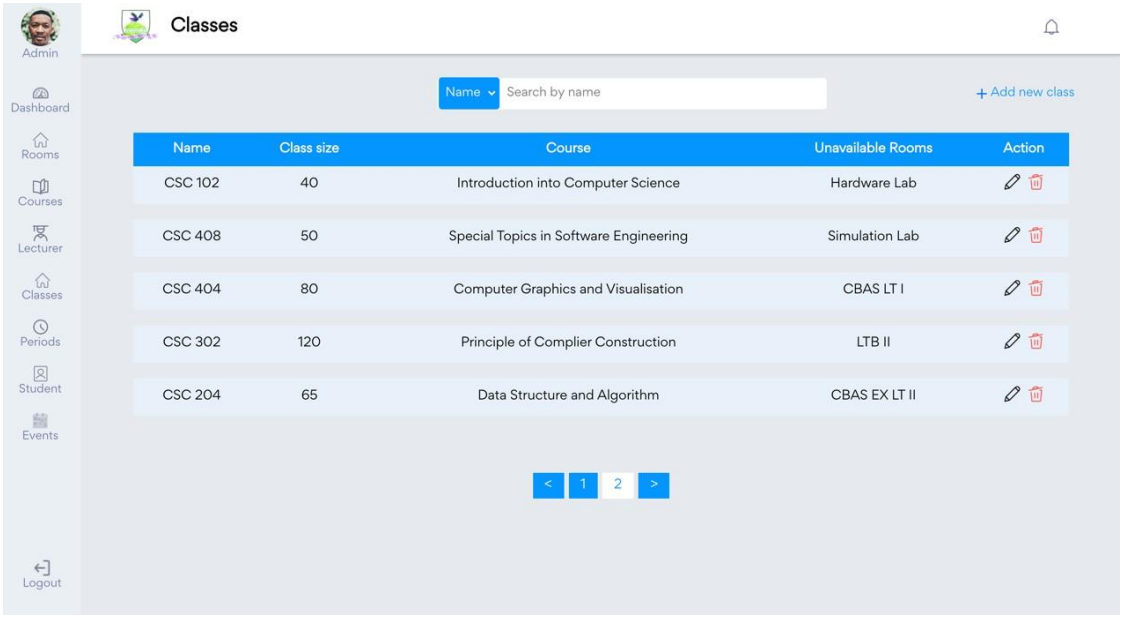

Figure 4.17(b): Screenshot of Admin Interface for managing existing classes

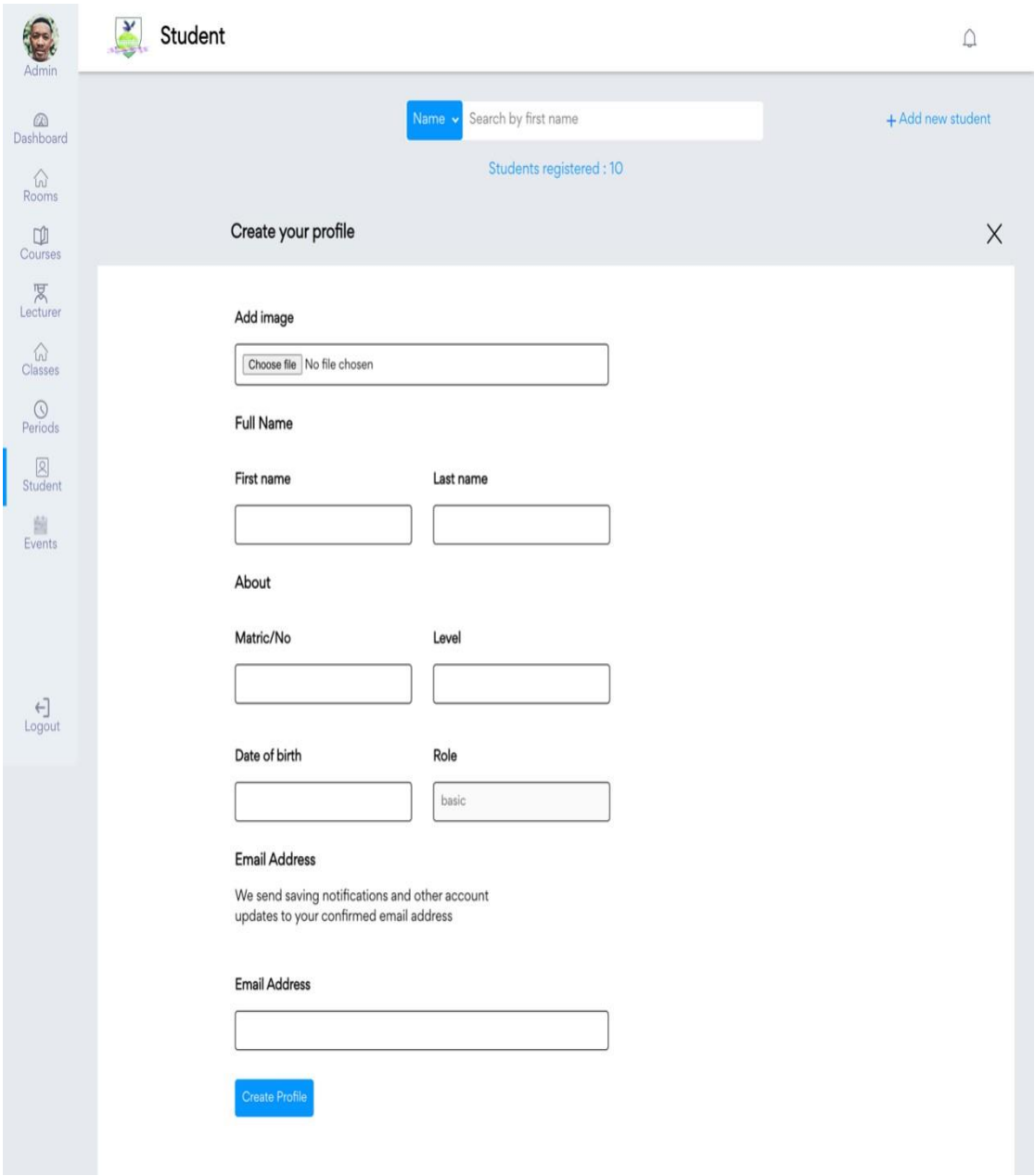

Figure 4.18(a): Screenshot of Admin Interface for creating a new Student

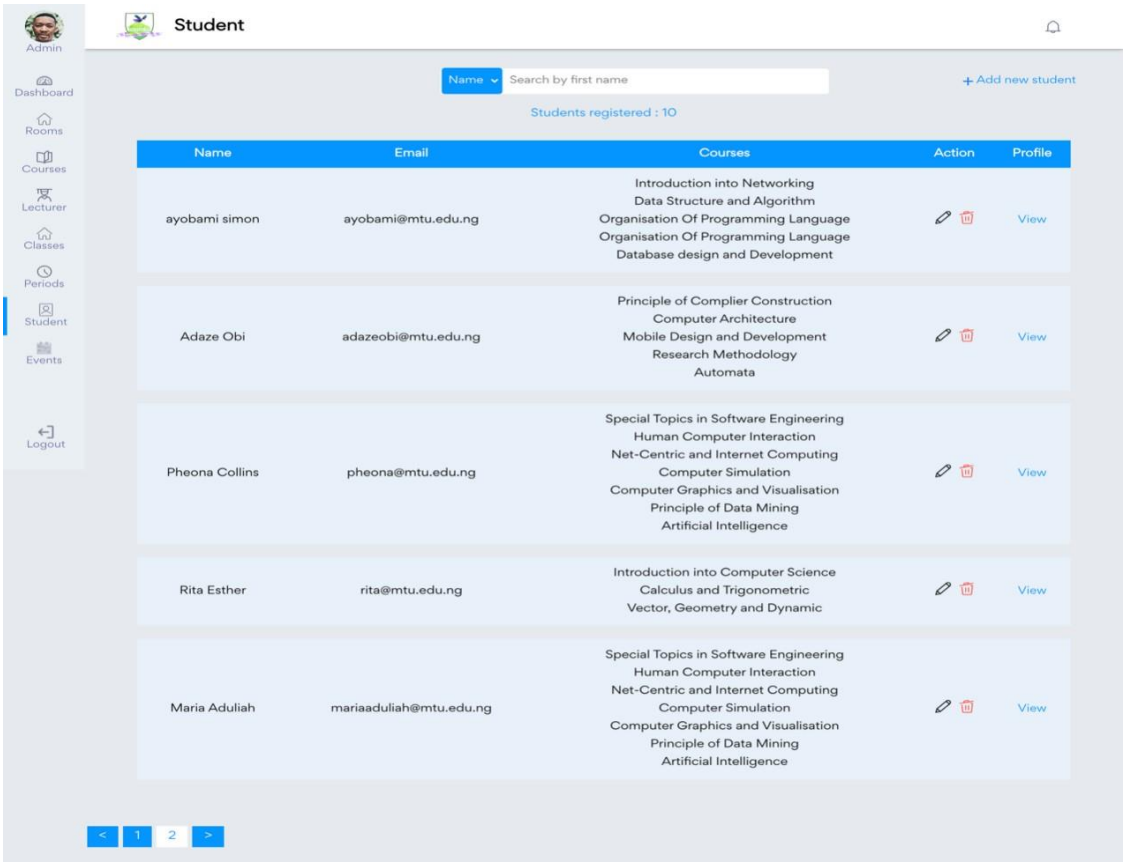

Figure 4.18(b): Screenshot of Admin Interface for managing existing Student

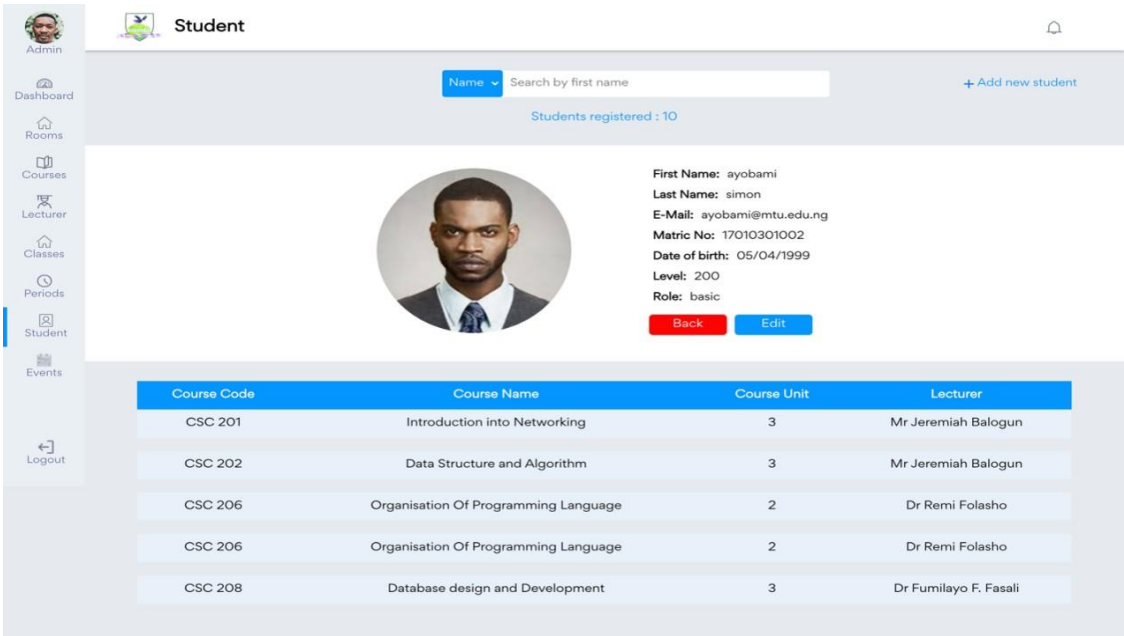

Figure 4.18(c): Screenshot of Admin Interface for view the Information of Student

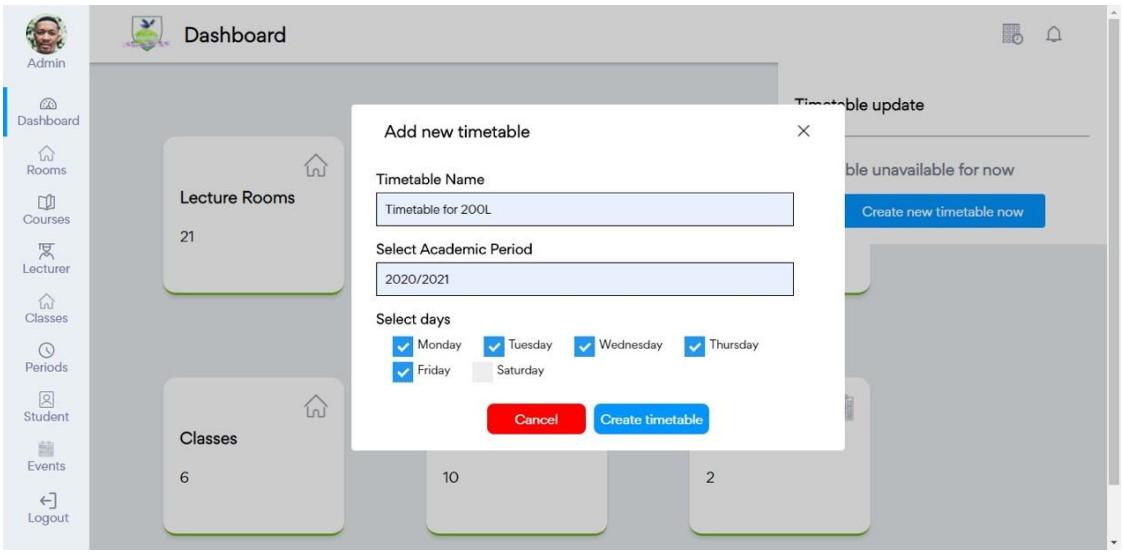

Figure 4.19(a): Screenshot of Admin Interface for creating a new Timetable

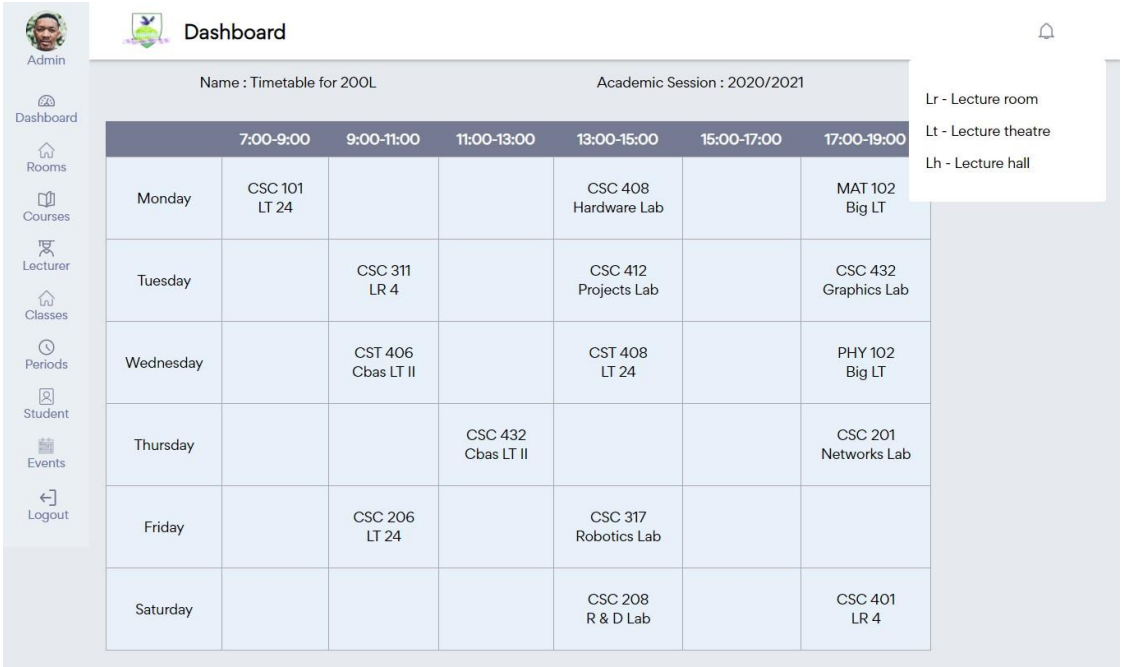

Figure 4.19(b): Screenshot of Admin Interface for managing existing Timetable

the requirement of details such as name of the timetable, academic session, days of the week, list of various courses in a department, duration of for a course and venue for a course.

Figure 4.20 shows the system login page also called the Home Page that give access to the student to use the mobile app. The student is are required to provide their school mail address or school Id and passwords issued to them (their first name by default). Figure 4.21(a) shows the result of the interface is required by the Student for resetting their password in a situation when they are unable to recollect their current password. Thus, this interface shows that the student is required to provide the school email to validate if that student exists or not. Figure 4.21(b) shows the result of the interface is required by the Student for creating a new password. Thus, this interface shows that the student is required to provide information such as generated password (token sent to their school mail), and new password.

The result of the interface in Figure 4.22(a) shows is required by a student for viewing information courses registered by a student. Thus, this interface shows the requirement of details such as name of the student, image of the student, course title, code, and unit. Figure 4.22(b) shows result of the interface required by a student for viewing information regarding a course. Thus, this interface shows the requirement of details such as course title, code, number of units for the course, lecturer handling the course, a description of the course, the venue of the of course, and time. Figure 4.22(c) shows result of the interface required by a student to add new courses. Thus, this interface shows the requirement of details such as name of the student, image of the student, a search field to find a course, the maximum unit and minimum unit by a student, course title, code and number of units for the course.

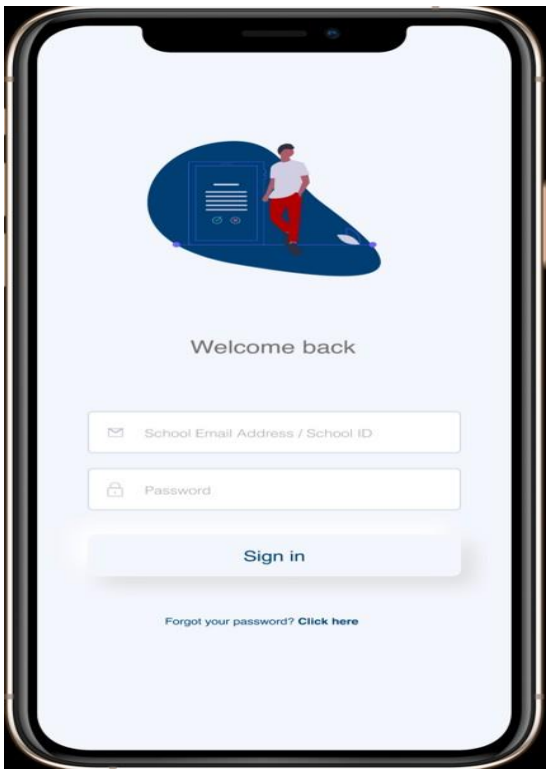

Figure 4.20: Screenshot of the Login page for Student Interface

(a) (b)

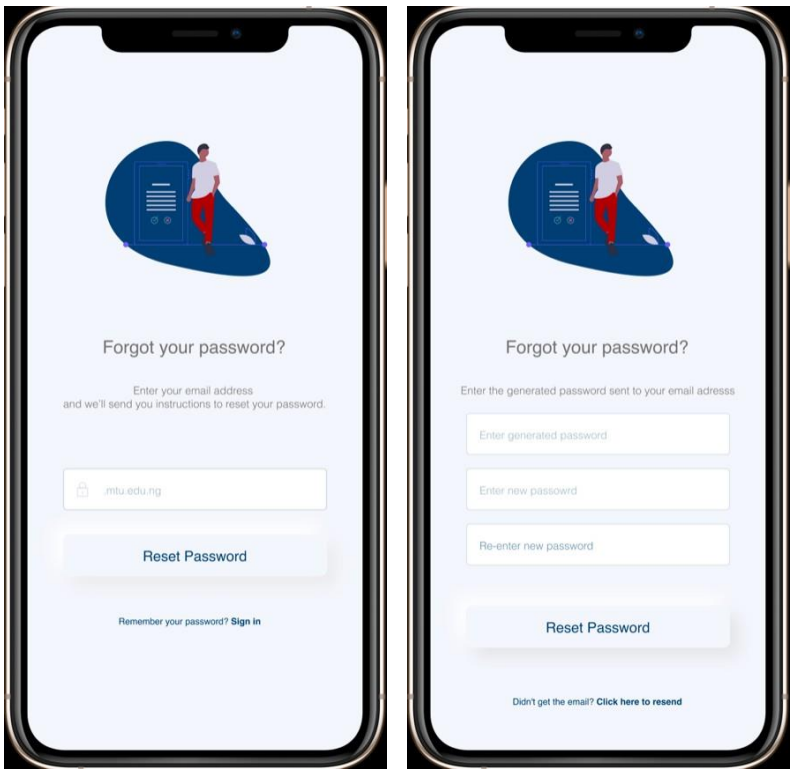

Figure 4.21 (a) and (b) Shows a Screenshot of the Respective Student interface for Resetting Password and Creating a New Password

(a) (b)

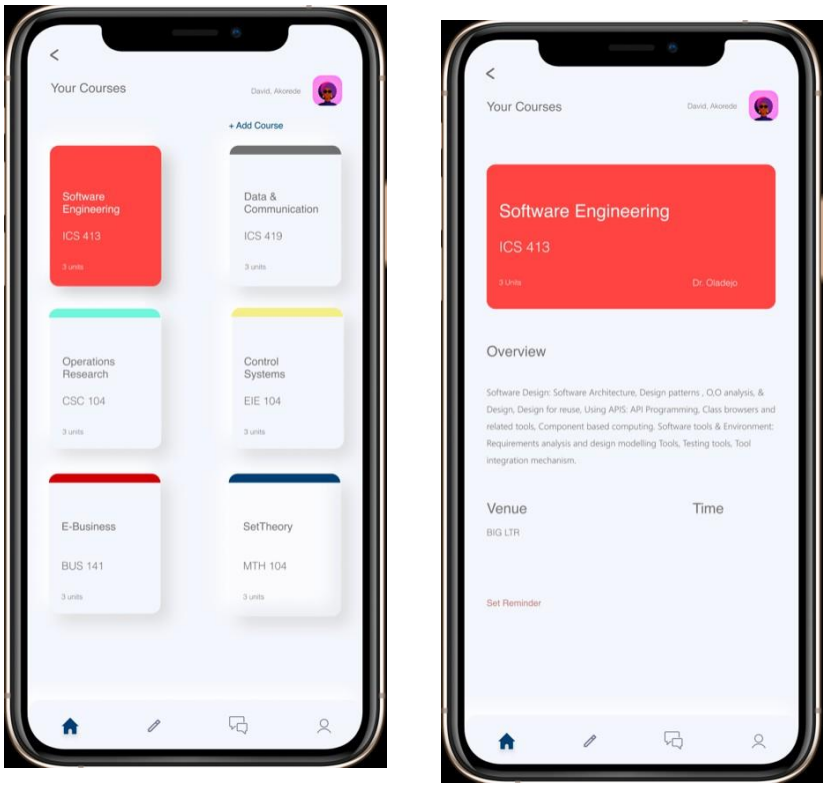

Figure 4.22(a) and (b): Shows a Screenshot of the Respective Student interface for viewing registered courses and details of a course.

Figure 4.23 shows result of the interface required by a student for viewing information regarding a lecturer handling a course. Thus, this interface shows the requirement of details such as lecturer image, name, educational background, telephone number, email address, office number, area of specialization and bio data. Figure 4.24(a) shows result of the interface required by a student for viewing information regarding course schedules based on registered courses. Thus, this interface shows the requirement of details such as the day the course is going to be held, course code, venue of the course, and time allocated for the course.

Figure 4.24(b) shows result of the interface required by a student for viewing information course schedules based on a particular day of the week. Thus, this interface shows the requirement of details such as the name of day of the week and the list of course that is going to be held on that day. Figure 4.24(c) shows result of the interface required by a student for setting a reminder for courses. Thus, this interface shows the requirement of details such as the list of course that is going to be held on a particular day and time for the alert.

Figure 4.25(a) shows result of the interface required by a student for viewing information no conversation is available. Figure 4.25(b) shows result of the interface required by a student for creating a new conversation based on the on the student level. Thus, this interface shows the requirement of details such as title of the conversation, and the detail of the conversation. Figure 4.25(c) shows result of the interface required by a student for viewing the list of conversation made the student current level. Thus, this interface shows the requirement of details such as title of the conversation the detail of the conversation, the name of user who made the created the conversation, the image of the user, and the number of replies, list of the replies.

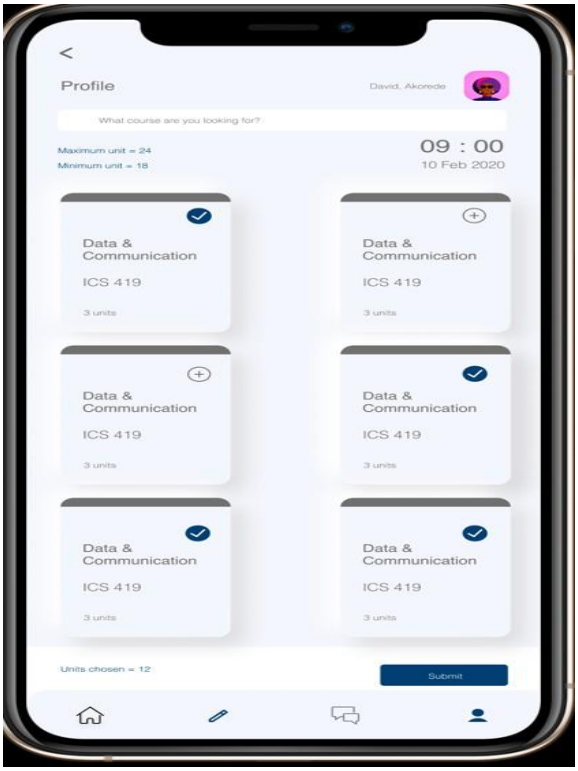

Figure 4.22(c): Shows a Screenshot of the Respective Student interface for Adding of a Course.

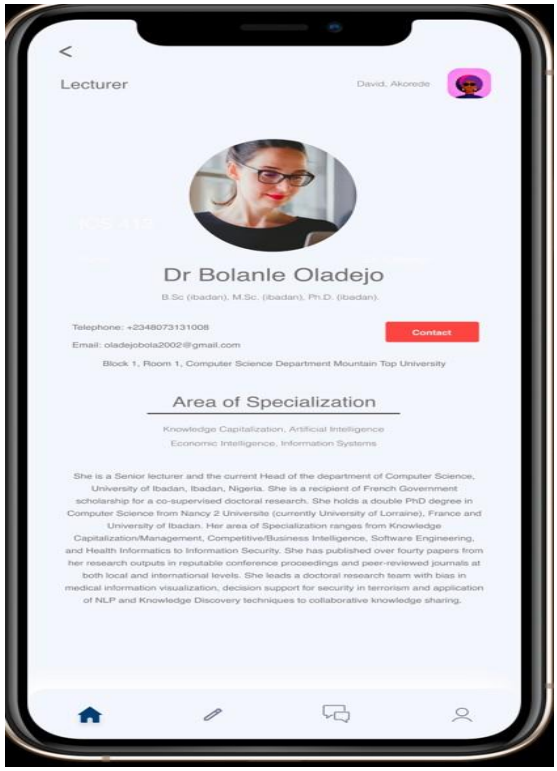

Figure 4.23: Shows a Screenshot of the Student Interface for viewing information about a lecturer handling a course.

(a) (b)

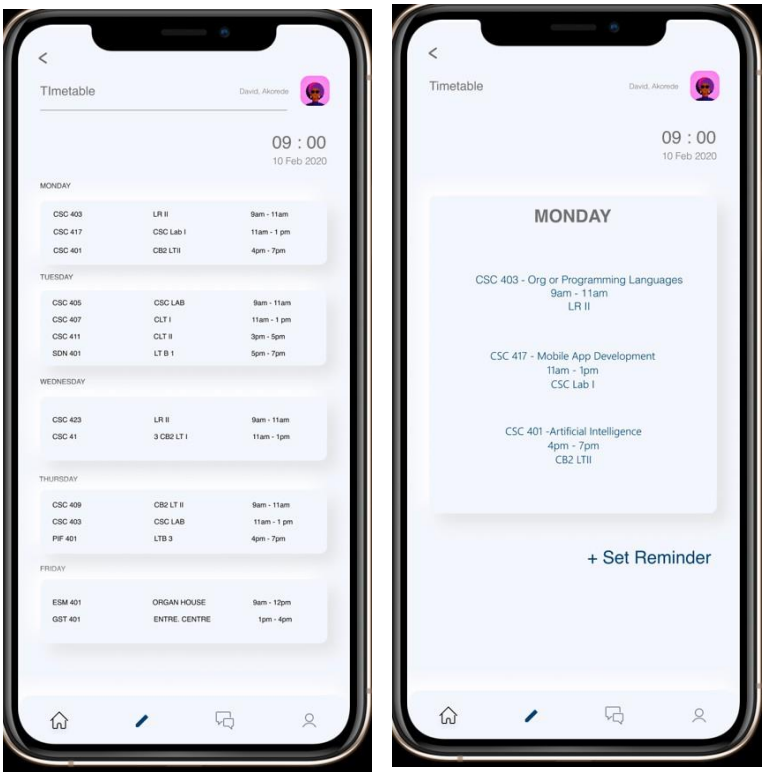

Figure 4.24(a) and (b): Shows a Screenshots of the Respective Student interface for viewing information of course schedule for a student based on registered course, and day of the week

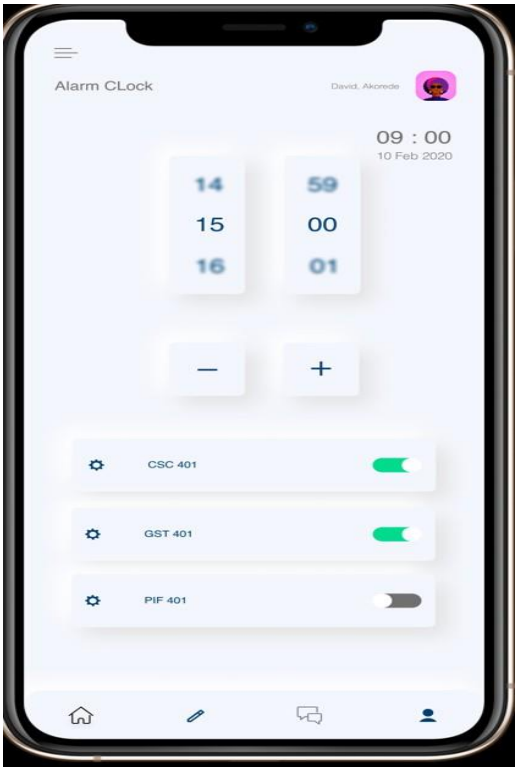

Figure 4.24 (c): Shows a Screenshot of the Student interface for setting reminder

(a) (b)

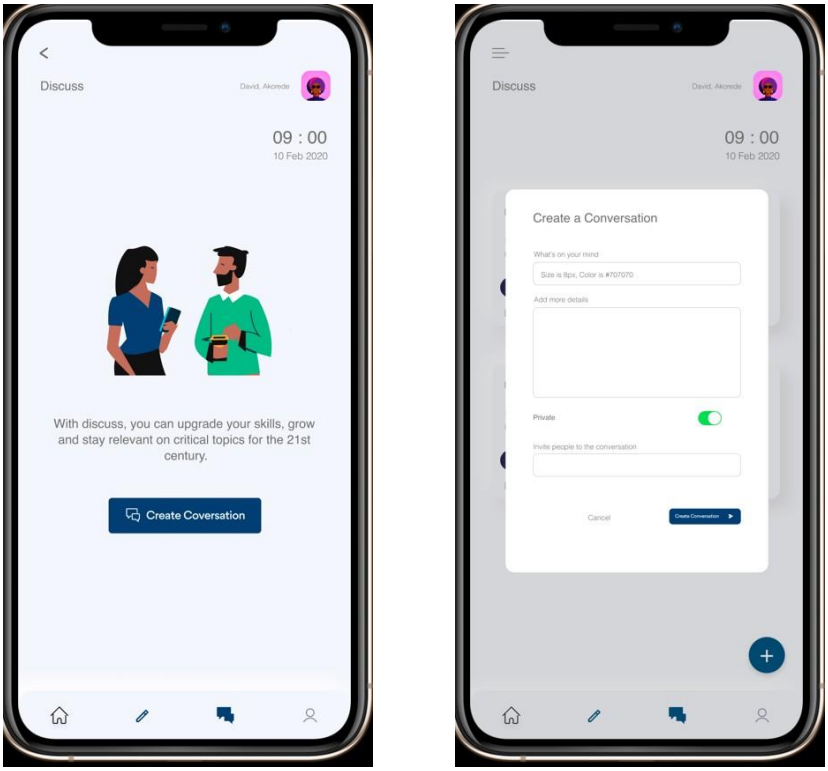

Figure 4.25 (a) and (b) shows a Screenshot the Respective Student interface when no conversation is available and creating a conversation.

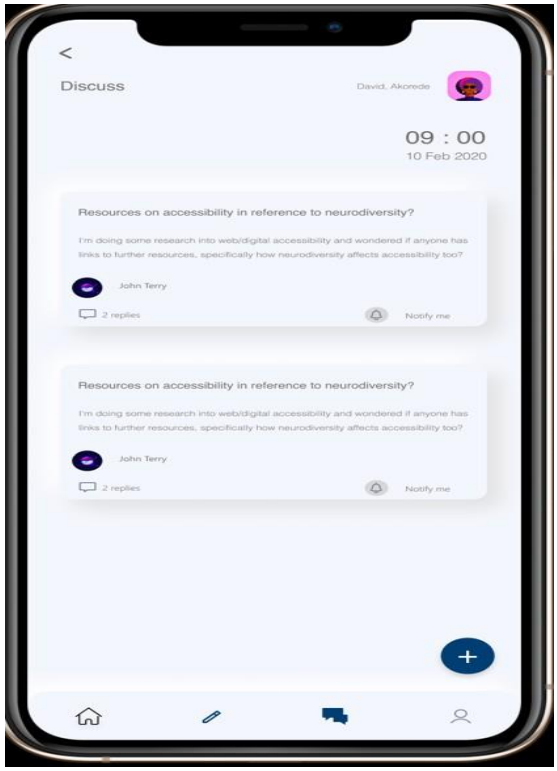

Figure 4.25(c) shows a Screenshot the Respective Student interface viewing a conversation.

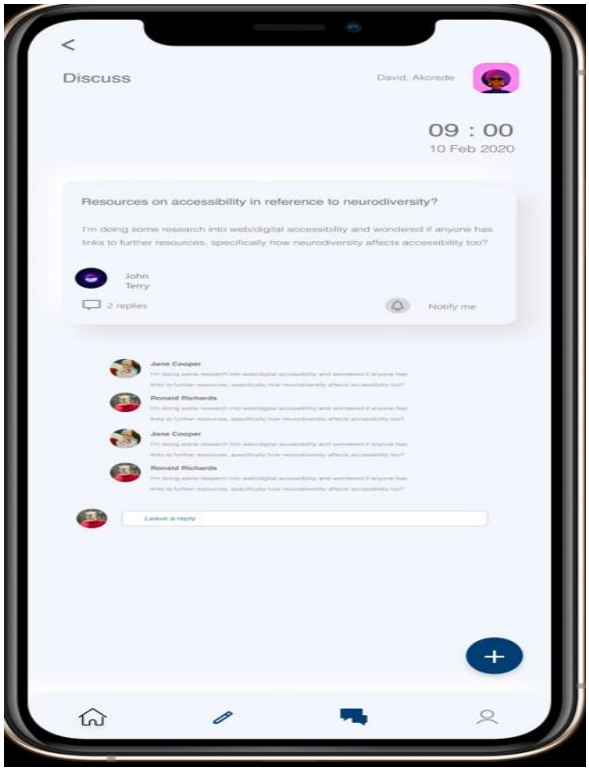

Figure 4.25 (d) shows a Screenshot the Respective Student interface for commenting on a conversation.

The result of the interface in Figure 4.26(a) is required by a student for viewing information regarding a student profile. Thus, this interface shows the requirement of details such as full name of the student, image of the student, level, date of birth, and list of courses offered by a student. The result of the interface in Figure 4.26(b) is required by a student for editing their profile. Thus, this interface shows the requirement of details such as full name of the student, image of the student, level, date of birth, and password. Thus, this interface shows the requirement of details such as title of the conversation the detail of the conversation, the name of user who made the created the conversation, the image of the user, and the number of replies. Figure 4.26(d) shows result of the interface required by a student for commenting on a conversation of interest.

4.4 Discussion of Result

The results of the study as presented, shows the different expectations of this study based on the objectives that were stated in the earlier chapters of this study. The results of the identification of the user and system requirements allowed for the identification of the different users of the proposed system such as primary and secondary user of the system. The results showed that the primary user was responsible for creating course scheduling and managing information regarding a department. which in return lead slow response time, the results showed that the secondary users can only access the system using their school email or Id and passwords provided by the system administrator of the system. The result also shows that the primary user is can view course schedules based on their registered courses, set reminder and make create a conversation.

(a) (b)

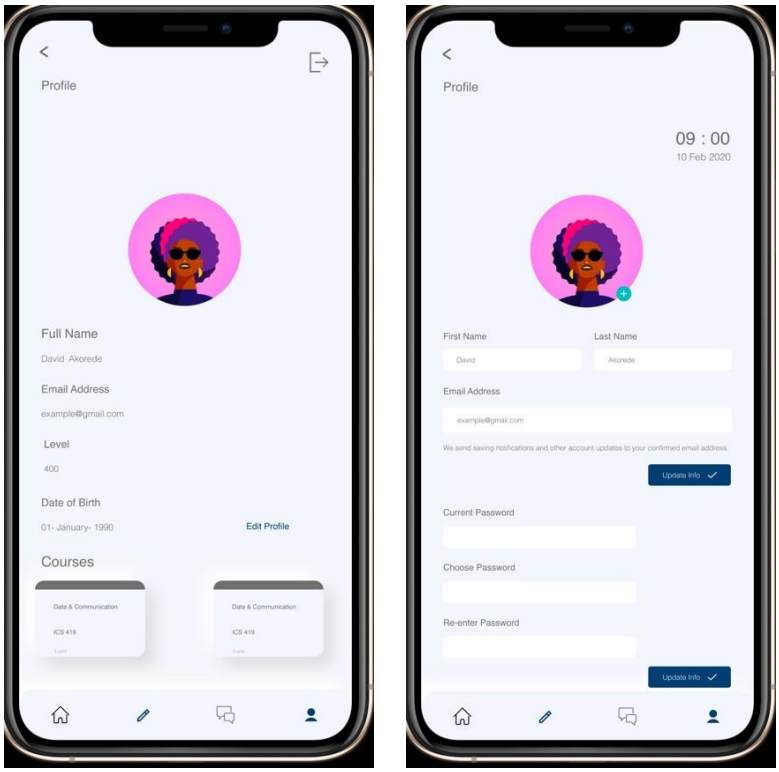

Figure 4.26 (a), and (b) a Screenshot shows the Respective Student interface for viewing and editing Student Profile.

The result shows that with the help of genetic algorithm the system was able to schedule course efficient based on the class size. Although when the number of mutation iteration was increase from 5,000 to 10,000 the accuracy of fitness increased exponentially the web server in generating the course schedules as a result of low memory size and processing speed of size of the web server. However, the system was able to performed exceptionally well even when number of data input was increased with a benchmark of 5,000 of iteration.

The results of the system implementation showed the database of system implemented using Mongodb. For each collection in the database, a number of documents was used for storing and retrieving of information. Also, the system database information regarding location were restricted only to Computer Science and Mathematics department. The results of the web and mobile system interface showed that the system was able to provide interfaces that were compliant with the system and user requirements that were identified in this study. The system implementation allowed users of various roles to perform their various duties using the system thus, removing the challenges such as the manual approach of scheduling course and cluster of course schedule.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Summary

This study developed a personalized course scheduling system that enables a timetable departmental representative created course schedules and student view course schedules on registered course. The study identified the user and system requirements that were required to be met by the system. The user and system requirements were identified alongside the hardware and software requirements of the system. The requirements of the system were also specified using unified modeling languages using use-case diagrams for user requirements specification and class diagrams for data modeling specification. The system was implemented using Web 3.0 technologies such as HTML, CSS and React JS for the web layout and data movement in and out of web interface forms from and to system database. MongodB was also used to implement the system database and python for creating the genetic algorithm for this study.

5.2 Conclusion

In conclusion, this study has designed and implemented a system to solve the currently challenge faced in scheduling of courses in academic institutions. The study was able to identify the respective user and system requirements of the system and appropriate designs were used to specify these requirements provided by the users using use-case and class diagrams. The system database was implemented in order to suit the mechanisms and inner workings of the proposed system.

5.3 Recommendation

This study recommends that further work on this study can be focused creating a personalized interface for academic staff for viewing course handled so that they that will help be able the organize of task and allocation of time and resources efficiently.

Reference

Muhammad, S.H.1, Galadanci, B.S.1, Mustapha, A.1 and Yahaya, A.S2. (2017). DESIGN AND IMPLEMENTATION OF AN ANDROID AND WEB-BASED UNIVERSITY TIMETABLE CUSTOMIZATION SYSTEM. Bayero Journal of Pure and Applied Science, 7.

Gilberto Rivera, Luis Cisneros, Patricia Sánchez-Solís, Nelson Rangel-Valdez and Jorge Rodas-Osollo. (2019, 09 21). Genetic Algorithm for Scheduling Optimization Considering Heterogeneous Containers: A Real-World Case Study, 16.

Celiz, E. A. (2018). UNIVERSITY TIMETABLE SCHEDULING USING METAHEURISTIC ADAPTIVE-ELITIST GENETIC ALGORITHM. 70.

Sandhu, K. S. (2003). Automating Class Schedule Generation in the Context of a University Timetabling Information System. 202.

Onuwa, N. I. (2015). DESIGN AND IMPLEMENTATION OF MOBILE BASED STUDENTS TIMETABLE MANAGEMENT SYSTEM. 89.

Herath, A. K. (2017). Genetic Algorithm For University Course Timetabling Problem. 47.

Sommerville, I. (2011). SOFTWARE ENGINEERING (Vol. 9). (M. H. Marcia Horton, Ed.) Addison-Wesley.

Jeffries, R. E. (2011, 03 16). What is Extreme Programming. From ronjeffries.com: https://ronjeffries.com/xprog/what-is-extreme-programming/ m well, D. (2013, 10 8). When should extreme programming be used. From Extreme programming: http://www.extremeprogramming.org/ (n.d.).

Wren, A. 1996. Scheduling, Timetabling and Rostering - A Special Relationship? In Proc. of 1st Practice and Theory of Automated Timetabling, 46-75

Rohini V (2016). Scheduling System for Univearsity.

Asif Ansari, and Prof Sachin Bojewar (2014). Genetic Algorithm to Generate the Automatic Time-Table – An OverView

Media Mark (3.2.12). "Sass - Syntactically Awesome Stylesheets". Sass-lang.com. Retrieved 2014-02-23.

Sass: Syntactically Awesome Style Sheets". sass-lang.com. Archived from the originalon 2013-09-01.

CSS developer guide". Mozilla Developer Network. Archived from the original on 2015-09-25. Retrieved 2015-09-24.

The MIT License". Open Source Initiative. 17 September 2018. Retrieved 17 September 2018.

RoomModel.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const mongoosePaginate = require('mongoose-paginate-v2');


const RoomSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    capacity: {
        type: Number,
        required: true
    },
},
 { timestamps: true }
)

RoomSchema.plugin(mongoosePaginate);

const Room = mongoose.model('room', RoomSchema);

module.exports = { Room };
```

LecturerModel.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema
const mongoosePaginate = require('mongoose-paginate-v2');


const LecturerSchema = new Schema({
    name: {
        type: String,
        required: true,
    },
    image: {
        type: String
    },
    ranking: {
        type: String
    },
    degree: {
        type: String
    },
    office_no: {
        type: String,
        required: true
    },
    phone_no: {
        type: Number,
        required: true
    },
    email: {
        type: String,
        required: true,
    },
    areaOfSpec: {
```

```
        type: String,
        required: true
    },
    education_bg: {
        type: String,
        required: true
    },
    unavailablePeriods: {
        type: String,
    },
    Courses: [{ type: Schema.Types.ObjectId, ref: 'Course', default:
null }]
})

LecturerSchema.plugin(mongoosePaginate);

const Lecturer = mongoose.model("Lecturer", LecturerSchema)
```

CouresModel.js

```
const mongoose = require('mongoose');
const mongoosastic = require('mongoosastic')
const Schema = mongoose.Schema
const mongoosePaginate = require('mongoose-paginate-v2');
const aggregatePaginate = require('mongoose-aggregate-paginate-v2');


const colorValidator = (v) => (/^#([0-9a-f]{3}){1,2}$/i).test(v)

const CourseSchema = new Schema({
    name: {
        type: String,
        required: true,
    },
    code: {
        type: String,
        required: true,
    },
    unit: {
        type: Number,
        required: true
    },
    day: {
        type: [String]
    },
    description: {
        type: String
    },
    level: {
        type: Number
    },
    colorCode: {
        type: String,
        validator: [colorValidator, 'Invalid color'],
        required: true
    },
    lecturer: [{type: Schema.Types.ObjectId, ref: 'Lecturer' }],
    students: [{type: Schema.Types.ObjectId, ref: 'User'}],
    venue: {type: Schema.Types.ObjectId, ref: 'room'},
    time: {
        type: String,
    },
```

```
})

//CourseSchema.plugin(mongoosastic)
CourseSchema.plugin(aggregatePaginate);
CourseSchema.plugin(mongoosePaginate);

CourseSchema.index({
    name: "text", code: "text"
})

const Course = mongoose.model("Course", CourseSchema)



module.exports = Course
```

StudentModel.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const Schema = mongoose.Schema;
const SALT_WORK_FACTOR = 10
const mongoosePaginate = require('mongoose-paginate-v2');
const aggregatePaginate = require('mongoose-aggregate-paginate-v2');

const UserSchema = new Schema({
    firstname: {
        type: String,
        required: true
    },
    lastname: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    dob: {
        type: String,
        required: true
    },
    level: {
        type: String,
        required: true
    },
    matric: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    role: {
        type: String,
        enum: ["basic", "admin"]
    },
    resetToken: {
        type: String,
        default: ''
    },
```

```
    image: {
        type: String,
        default: ''
    },
    maxUnit: {
        type: Number,
        default: 24
    },
    minUnit: {
        type: Number,
        default: 16
    },
    selectedUnit: {
        type: Number,
    },
    courses: [{ type: Schema.Types.ObjectId, ref: 'Course' }]
},
 { timestamps: true }
)
```

TimetableModel.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const TimetableSchema = new Schema({
    courses: {
        type: Array
    },
    name: {
        type: String
    },
    session: {
        type: String
    },
    uid: {
        type: String
    },
    current_progress: {
        type: Number
    },
    total_progress: {
        type: Number
    },
},
 { timestamps: true }
)

const Timetable = mongoose.model('timetable', TimetableSchema);

module.exports =  Timetable
```

```
/* eslint-disable array-callback-return */
import React,{useState} from "react"
import {Link} from "react-router-dom"
import { CSSTransition, TransitionGroup } from "react-transition-
group";
import "./room.css"
import "../../global/global.css"
import plus from "../../images/plus.svg"
import bin from "../../images/bin.png"
import pen from "../../images/pencil 1.png"
import cross from "../../images/close.png"
import logo from "../../images/Logo.png"
import axios from "axios"
import spinner from "../../images/spinner.gif"
import checkg from "../../images/checkg.png"
import checkr from "../../images/checkr.png"
import checkb from "../../images/checkb.png"
import ReactPaginate from "react-paginate"
import { useQuery, useMutation , queryCache } from "react-query"

const getRooms = (page, {pageNo, search}) => {

        return axios.get('https://tbe-node-
deploy.herokuapp.com/Admin/room', {
            headers: {},
            params: {perPage: 5, page: pageNo, searchQuery: search}
        })
        .then((response) => {
            var rooms = response.data?.data.docs
            var pages = response.data?.data.totalPages
            return {rooms, pages}
        })
 }

const createRoom = (roomData) => {
        let data = JSON.stringify(roomData);

        return axios.post('https://tbe-node-
deploy.herokuapp.com/Admin/room', data, {
            headers: {
                'Content-Type': 'application/json'
            }
        })
        .then((response) => {
            return response;
        })
        .then((error)=> {
            return error
        })
}
```

Lecturer.js

```
const createLect = (datum) => {
        let data = new FormData();

        // Getting values
        var file = document.getElementById("fileInput").files[0]
```

```
        var name = document.querySelector("#namec").value
        var edubg = document.querySelector("#edubgc").value
        var pos = document.querySelector("#posc").value
        var email = document.querySelector("#emailc").value
        var phone = document.querySelector("#phonec").value
        var aos = document.querySelector("#aosc").value
        var officeno = document.querySelector("#officenoc").value
        var degree = document.querySelector("#degreec").value
        data.append('name', name);
        data.append('email', email);
        data.append('education_bg', edubg);
        data.append('phone_no', phone);
        data.append('office_no', officeno);
        data.append('ranking', pos);
        data.append('degree', degree);
        data.append('areaOfSpec', aos);
        data.append('image', file);
        var array = []

        for (var i = 0; i < datum.length; i++) {
            array.push(datum[i])
            data.append(`Courses[${datum.indexOf(datum[i])}]`,
datum[i])
        }

        console.log(...data)

        return axios.post('https://tbe-node-
deploy.herokuapp.com/Admin/lecturer/image', data, {
            headers: {
                'Content-Type': 'multipart/form-data'
            }
        })
        .then((response) => {
            return response;
        })
        .then((error)=> {
            return error
        })
    }

    const getCourses = () => {

        return axios.get('https://tbe-node-
deploy.herokuapp.com/Admin/getCourse', {
            headers: {},
            params: { page: 1, searchQuery: ''}
        })
        .then((response) => {
            return response.data.data.docs
        })
    }
const getLect = (page, {pageNo, search}) => {

        return axios.get('https://tbe-node-
deploy.herokuapp.com/Admin/getlecturer', {
            headers: {},
            params: {perPage: 5, page: pageNo, searchQuery: search}
        })
        .then((response) => {
            var lect = response.data?.data.docs
```

```
            var pages = response.data?.data.totalPages
            return {lect, pages}
        })
    }
```

Course.js

```
onst createCourse = (finalDataObj) => {

    let data = JSON.stringify(finalDataObj);

    return axios.post('https://tbe-node-
deploy.herokuapp.com/Admin/course', data, {
        headers: {
            'Content-Type': 'application/json'
        }
    })
    .then((response) => {
        return response;
    })
    .then((error)=> {
        return error
    })
}

const getCourses = (page, {pageNo, search}) => {

    return axios.get('https://tbe-node-
deploy.herokuapp.com/Admin/getCourse', {
        headers: {},
        params: {perPage: 5, page: pageNo, searchQuery: search}
    })
    .then((response) => {
        var courses = response.data?.data.docs
        var pages = response.data?.data.totalPages
        return {courses, pages}
    })
}
```

Student.js

```
const createStud = () => {
        let data = new FormData();

        var file = document.getElementById("fileInput").files[0]
        var firstname = document.querySelector("#firstname").value
        var lastname = document.querySelector("#lastname").value
        var dob = document.querySelector("#dob").value
        var email = document.querySelector("#email").value
        var matric = document.querySelector("#matric").value
        var role = document.querySelector("#role").value
        var level = document.querySelector("#level").value

        data.append('firstname', firstname);
        data.append('lastname', lastname);
        data.append('email', email);
        data.append('dob', dob);
        data.append('matric', matric);
        data.append('role', role);
        data.append('image', file);
```

```
        data.append('level', level)

        return axios.post('https://tbe-node-
deploy.herokuapp.com/signup', data, {
            headers: {
                'Content-Type': 'multipart/form-data'
            }
        })
        .then((response) => {
            return response;
        })
        .then((error)=> {
            return error
        })
}

const getStudentsL = () => {

    return axios.get('https://tbe-node-
deploy.herokuapp.com/Admin/students/all', {
        headers: {},
        params: { page: 1, searchQuery: ''}
    })
    .then((response) => {
        return response.data.data.docs
    })
}
```

Timetable.js

```
const Dashboard = (props) => {
    const [timetableFn, {status}] = useMutation(postTimetable, {
        onSuccess: () => {
            console.log("created")
            setCreated(true)
            setModalOut(false)
            setTimeout(() => {
                // setCreated(true)
            }, 10);
            // setCreated(false)
        },
        onError: (error) => {
            console.log({...error})
        }
    })


    function uuidFn() {
        return 'xxxxxxxx-xxxx-4xxx-yxxx-
xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
            var r = Math.random() * 16 | 0, v = c == 'x' ? r : (r &
    const [uuid, setUuid] = useState(uuidFn())
    const [int, setInt] = useState(2000)
    const [enable, setEnable] = useState(false)

    const tResponse = useQuery(['tResponse', {uuid}], getTRes, {
```

```
        refetchOnWindowFocus: false,
        refetchInterval: int,
        enabled: enable
    })

    const coursesT = useQuery('coursesT', getCoursesT, {
        refetchOnWindowFocus: false
    })

    console.log(tResponse.data?.data.data)

    const roomsT = useQuery('roomsT', getRoomsT, {
        refetchOnWindowFocus: false
    })

    const {isLoading, data} = useQuery('lengths', getLength, {
        refetchOnWindowFocus: false
    })

    const [modalOut, setModalOut] = useState(false)
    const [updateOut, setUpdateOut] = useState(false)
    const [created, setCreated] = useState(false)
    const [showT, setShowT] = useState(false)



    const submit = (e) => {

        e.preventDefault()
        const datum =
[...document.querySelectorAll('input[type=checkbox]:checked')].map(e
=> e.value);

        timetableFn({roomsT, coursesT, datum, uuid})

        setEnable(true)
        tResponse.refetch()
    }

    setInterval(()=> {
        if(tResponse.data?.data.data?.current_progress === 5000){
            setInt(-1)
        }
    }, 1000)

    const cProgress = tResponse.data?.data.data?.current_progress;
    const tProgress = tResponse.data?.data.data?.total_progress;
    const tDate = new
Date(tResponse.data?.data.data?.updatedAt.substring(0,10)).toDateStri
ng();
    const tTime =
tResponse.data?.data.data?.updatedAt.substring(11,16);


    var monday;

    const [mon1, setMon1] = useState([])
    const [mon2, setMon2] = useState([])
    const [mon3, setMon3] = useState([])
    const [mon4, setMon4] = useState([])
```

```
    var tuesday;

    const [tue1, setTue1] = useState([])
    const [tue2, setTue2] = useState([])
    const [tue3, setTue3] = useState([])
    const [tue4, setTue4] = useState([])

    var wednesday;

    const [wed1, setWed1] = useState([])
    const [wed2, setWed2] = useState([])
    const [wed3, setWed3] = useState([])
    const [wed4, setWed4] = useState([])

    var thursday;

    const [thur1, setThur1] = useState([])
    const [thur2, setThur2] = useState([])
    const [thur3, setThur3] = useState([])
    const [thur4, setThur4] = useState([])

    var friday;

    const [fri1, setFri1] = useState([])
    const [fri2, setFri2] = useState([])
    const [fri3, setFri3] = useState([])
    const [fri4, setFri4] = useState([])

    var saturday;

    const [sat1, setSat1] = useState([])
    const [sat2, setSat2] = useState([])
    const [sat3, setSat3] = useState([])
    const [sat4, setSat4] = useState([])

    const arrangeT = () => {
        // eslint-disable-next-line no-unused-expressions


console.log(JSON.stringify(tResponse.data?.data.data?.courses))

        monday = tResponse.data?.data.data?.courses.filter((course)
=> {


tResponse.data?.data.data?.courses.filter((course) => {
            return course.assignedDay === 'wednesday'
```

```
=> {


=> {
```

```
        }
    })

    tuesday.map((tue)=> {
        if(tue.startHour === 9){
            setTue1(tue.name)
        }else if(tue.startHour === 11){
            setTue2(tue.name)
        }else if(tue.startHour === 13){
            setTue3(tue.name)
        }else if(tue.startHour === 15){
            setTue4(tue.name)
        }
    })

    wednesday.map((wed)=> {
        if(wed.startHour === 9){
            setWed1(wed.name)
        }else if(wed.startHour === 11){
            setWed2(wed.name)
        }else if(wed.startHour === 13){
            setWed3(wed.name)
        }else if(wed.startHour === 15){
            setWed4(wed.name)
        }
    })

    thursday.map((thur)=> {
        if(thur.startHour === 9){
            setThur1(thur.name)
        }else if(thur.startHour === 11){
            setThur2(thur.name)
        }else if(thur.startHour === 13){
            setThur3(thur.name)
```

```
            }else if(thur.startHour === 15){
                setThur4(thur.name)
            }
        })

        friday.map((fri)=> {
            if(fri.startHour === 9){
                setFri1(fri.name)
            }else if(fri.startHour === 11){
                setFri2(fri.name)
            }else if(fri.startHour === 13){
                setFri3(fri.name)
            }else if(fri.startHour === 15){
                setFri4(fri.name)
            }
        })

        saturday.map((sat)=> {
            if(sat.startHour === 9){
                setSat1(sat.name)
            }else if(sat.startHour === 11){
                setSat2(sat.name)
            }else if(sat.startHour === 13){
                setSat3(sat.name)
            }else if(sat.startHour === 15){
                setSat4(sat.name)
            }
        })
    }
```

RoomController.js

```
const { query } = require("express");
const express = require("express");
const mongoose = require("mongoose");
const { Room } = require("../models/room");
require('mongodb');

/ create a Room
const createRoom = async (req, res) => {
  const { name, capacity } = req.body;

  try {
    const existingroom = await Room.findOne({ name: req.body.name
}).exec();
    if (existingroom) {
      return res.status(401).json({
        message: "name already taken",
      });
    }

    const room = new Room({
      name,
      capacity,
    });
    await room.save();
    return res.json({
      success: true,
      data: room,
    });
  } catch (error) {
    return res.status(500).json({
      error: "There was an error.",
      success: false,
    });
  }
};

// get all rooms
const GetAllRooms = async (req, res) => {
  try{
    const { page, perPage, searchQuery, sort } = req.query;
    let sortQuery = { }
    switch (sort) {
      case "name": {
        sortQuery.name = 1;
        break;
      }

      case "capacity": {
        sortQuery.capacity = 1;
        break;
      }

      default: {
        sortQuery.name = 1;
      }
    }
    const options = {
```

```
      page: parseInt(page, 10) || 1,
      limit: parseInt(perPage, 10) || 10,
    };
    const rooms = await Room.paginate({ name:  new
RegExp(`^${searchQuery}`, "i")}, options);
    if (rooms) {
      return res.status(200).json({
        success: true,
        data: rooms,
      });
    } else {
      return res.status(404).json({
        error: "no rooms found",
      });
    }
  } catch (error) {
    return res.status(500).json({
      error: "error",
      success: false,
    });
  }
};
```

LecturerController.js

```
const express = require("express");
const mongoose = require("mongoose");
const Lecturer = require("../models/lecturer");

// create a lecturer
const createLecturer = async (req, res) => {


  const lecturer = await Lecturer.create({
    name: req.body.name,
    email: req.body.email,
    unavailablePeriods: req.body.unavailablePeriods,
    courses: req.body.courses,
    education_bg: req.body.education_bg,
    phone_no: req.body.phone_no,
    office_no: req.body.office_no,
    ranking: req.body.ranking,
    degree: req.body.degree,
    areaOfSpec: req.body.areaOfSpec,
    image: req.file.path,
  });

  if (!lecturer) {
    res.status(400).json({
      success: false,
      message: "lecturer not created",
    });
  }
  res.json({
    success: true,
    message: "lecturer created",
    data: lecturer,
  });
};
// get all lecturers
const getAllLecturer = async (req, res) => {
  try {
    const { page, perPage, searchQuery } = req.query;
```

```
    const options = {
      page: parseInt(page, 10) || 1,
      limit: parseInt(perPage, 50) || 50,
      populate: [{path: 'Courses'}]
    };
    const data = await Lecturer.paginate({ name:  new
RegExp(`^${searchQuery}`, "i")}, options);
    if (!data) {
      res.status(404).json({
        success: false,
        message: "not found",
      });
      return;
    } else {
      res.status(200).json({
        success: true,
        data,
      });
    }
  } catch (error) {
    console.log(error);
    res.status(400).json(error);
  }
};
```

CourseController.js

```
const express = require("express");
const mongoose = require("mongoose");
const generateColor = require("generate-color");
const Course = require("../models/course");
const User = require("../models/user");
const Lecturer = require('../models/lecturer');

var myAggregate = Course.aggregate([{$lookup: {from: "lecturers",
localField: "lecturer", foreignField: "_id", as: "lecturer"}},
{$addFields:{ number: {$size: { "$ifNull": [ "$students", [] ]
}}}}]);

// create a course
const createCourse = async (req, res) => {
  const {
    name,
    code,
    unit,
    time,
    day,
    venue,
    description,
    lecturer,
    level,
  } = req.body;

  try {
    const existingcourse = await Course.findOne({ name: req.body.name
}).exec();
    if (existingcourse) {
      return res.status(401).json({
        message: "name already taken",
      });
    }
```

```
    const colorCode = generateColor.default();

    const course = new Course({
      name,
      code,
      unit,
      day,
      time,
      venue,
      description,
      level,
      lecturer,
      colorCode,
    });
    await course.save();

    const data = await Course.findOne({ name: req.body.name
}).populate(
      "lecturer"
    );

    return res.json({
      success: true,
      data: data,
    });
  } catch (error) {
    return res.status(500).json({
      error: "There was an error.",
      success: false,
    });
  }
};

// get all courses
const GetAllCourses = async (req, res) => {

  const { page, perPage, searchQuery } = req.query;
  const options = {
    page: parseInt(page, 10) || 1,
    limit: parseInt(perPage, 10) || 50,
    populate: [{path: "venue"}, {path: "lecturer"}]
  };
  const courses = await Course.aggregatePaginate(myAggregate,
options)
  if (courses) {
    return res.status(200).json({
      success: true,
      data: courses,
    });
  } else {
    return res.status(404).json({
      error: "no course found",
    });
  }
};

// mobile get all courses
const GetCourseAll = async (req, res) => {
  const course = await Course.find({})
    .populate("venue")
    .populate("lecturer");
```

```
  if (course) {
    return res.status(200).json({
      success: true,
      data: course,
    });
  } else {
    return res.status(404).json({
      error: "no course found",
    });
  }
};
```

StudentController.js

```
const express = require("express");
const mongoose = require("mongoose");
const User = require("../models/user");
const Notifications = require("../models/user");
const Course = require("../models/course");
const jwt = require("jsonwebtoken");
const { secret } = require("../config/helper");
const { transporter } = require("../config/nodemailer");
const { nanoid, customAlphabet } = require("nanoid");
const bcrypt = require("bcrypt");
const Pusher = require("pusher");
var _ = require("lodash");
const { nextTick } = require("process");
const { error } = require("console");
const { use } = require("bcrypt/promises");
const { json } = require("express");
const Socket = require("../index");
require("dotenv").config();

const login = async (req, res) => {
  const { matric, password } = req.body;
  try {
    const user = await User.findOne({ matric: req.body.matric
}).exec();
    if (!user) {
      return res.status(404).json({
        error: "user not found",
        success: false,
      });
    }

    user.comparePassword(req.body.password, (err, match) => {
      if (!match) {
        return response
          .status(400)
          .send({ message: "The password is invalid" });
      }
    });

    const token = jwt.sign({ _id: user._id }, secret);
    return res.json({
      success: true,
      data: token,
    });
  } catch (err) {
    return res.status(500).json({
      err: "error login in",
      success: false,
```

```
    });
  }
};

const signup = async (req, res) => {

  try {
    const existingUser = await User.findOne({ matric: req.body.matric
}).exec();
    if (existingUser) {
      return res.status(401).json({
        error: "account alrady exists",
        success: false,
      });
    }
    const course = await Course.find({ level: req.body.level
}).populate("lecturer");
    console.log(course);

    const user = new User({
      firstname: req.body.firstname,
      lastname: req.body.lastname,
      email: req.body.email,
      dob: req.body.dob,
      level: req.body.level,
      matric: req.body.matric,
      password: req.body.firstname,
      role: req.body.role,
      courses: course,
      image: req.file.path
    });
```

TimetableController.js

```
const { model } = require("mongoose");
const axios = require("axios");
const Course = require("../models/course");
const Lecturer = require("../models/lecturer");
const { Room } = require("../models/room");
const { response } = require("express");
const Timetable = require("../models/tinetable");
const { error } = require("winston");

const sendTimetabledata = async (req, res) => {
  console.log(req.body);

  const timetable = new Timetable({
    uid: req.body.timetableId,
  });

  const existing = await timetable.find({ session: req.body.session})
  if (existing) {
    return res.status(401).json({
      error: "session already has timetable"
    });
  } else {
    await timetable.save();
  }

  const data = req.body;

  var config = {
```

```
    method: "get",
    url: "https://coursekit-timetable.herokuapp.com/generate/",
    headers: {
      "Content-Type": "application/json",
    },
    data: data,
  };

  axios(config)
    .then(function (response) {
      console.log(JSON.stringify(response.data));
      const result = JSON.stringify(response.data);
      return res.json({
        data: result,
      });
    })
    .catch(function (error) {
      console.log(error);
    });
};

const receivedata = async (req, res) => {
  const data = req.body;
  console.log(req.query.current_progress);
  console.log(data);

  await Timetable.findOneAndUpdate(
    { uid: req.query.timetableId },
    {
      current_progress: req.query.current_progress,
      total_progress: req.query.total_progress,
      name: req.body.timetableName,
      session: req.body.academicSession,
      courses: req.body.courses,
    },
    (err, timetable) => {
      if (err) {
        console.log(err);
      } else
        res.status(200).json({
          data: timetable,
        });
    }
  );
};

const progress = async (req, res) => {
  const timetable = await Timetable.findOne({ uid:
req.query.timetableId });
  if (!timetable) {
    return res.json({
      error: "error",
    });
  } else {
    return res.json({
      data: timetable,
    });
  }
};

const allTimetable = async (req, res) => {
```

```
  const data = await Timetable.find({});
  if (!data) {
    return res.status(404).json({
      error: "timetables not found",
    });
  } else {
    return res.status(200).json({
      data: data,
    });
  }
};

module.exports = { sendTimetabledata, receivedata, progress,
allTimetable };
```