Malaviya National Institute Of Technology Jaipur

Department of Computer Science and Technology

Software Requirements Specifications

TIMETABLE MANAGEMENT APPLICATION

Submitted by:
Vandita Goyal
2016ucp1004

Table Of Contents

Introduction

The main goal of the a Timetable Management System Software is to simplify the process of timetable management by enabling communication between **Students, Teachers and Administration**.
Communication between Students and their Teachers about when lectures are held is very important. Miscommunication can cause confusion and hence should be avoided. This system aims to remove that threat and streamline the process of timetable creation as well as rescheduling lectures.

Purpose

The main objective of this **Timetable Management System Software Requirement Specification** (SRS) is to provide a base for the foundation of the project. It gives a comprehensive view of how the system is supposed to work and what is to be expected by the end users. Client's expectation and requirements are analyzed to produce specific unambiguous functional and non-functional requirements, so they can be used by development team with clear understanding to build a system as per end user needs. This SRS for Timetable Management System can also be used for future as basis for detailed understanding on how project was started. It provides a blueprint for upcoming new developers and maintenance teams to assist in maintaining and modifying this project as per required changeability.

Scope of Project

This Timetable Management System will have three end-users: **Student, Teacher** and **Administration.** This will consist of a Timetable checking system and DBMS server. The **Students** as well as the **Teachers** will be able to check for their next lecture, check their attendance and get notifications for changes in their timetable. The **Teachers** will also be able to request to reschedule lectures and cancel them if necessary. **Administration** will be able to provide the students and teachers with their timetables and approve or deny the rescheduling requests on the basis of availability of lecture halls.

Definitions, Acronyms and Abbreviations

SRS	Software Requirements Specifications
FR	Functional Requirements
NFR	Non Functional Requirements
TTMS	Timetable Management System

TABLE 1: Definitions, Acronyms and Abbreviations

References

1. Creately
2. Concept Draw
3. Tutorials Point
4. Slide Share
5. Quora
6. Database Systems – Elmasri Navathe
7. Software Engineering – Pressman

Overview

The remaining sections of this documentations describes the overall descriptions which includes product perspective and functions, characteristics of users. It also consists of Assumptions, and Constraints. Overall description is listed in section 2. Section 3 includes Specific Requirements which consists of Functional and Nonfunctional requirements, External Interface Requirements, Software System Attributes, Performance Requirements, Capacity Requirements, Availability Requirements and Safety Requirements etc.

Overall Description

Product Functions

Some of the general functions of the Timetable Management System include the following:

1. Student and Teacher Registration and Allotment of the appropriate timetable
2. Display of the next lecture
3. Notification informing about the change in any lecture
4. Manage various timetables
5. Displays the attendance in each subject for each student

User Characteristics

1. Administration

Administration has super access to everything in the timetable management system. Administration is solely responsible for managing the college resources and staff (teacher). Administration is required to have experience on managing timetables and availability of resources previously, and have basic knowledge of database and application server.

2. Teacher

Teachers are a vital part of the system. They have less access to the system as compared to the Administration. They have access to view the next lecture in their timetable and put a request for an extra lecture or for rescheduling a lecture. Teachers are required to be able to use the web UI interface

3. Student

Students are vital part of the system. They have access to view their timetable and information about their next lecture. Student should at least be capable to use the web UI interface.

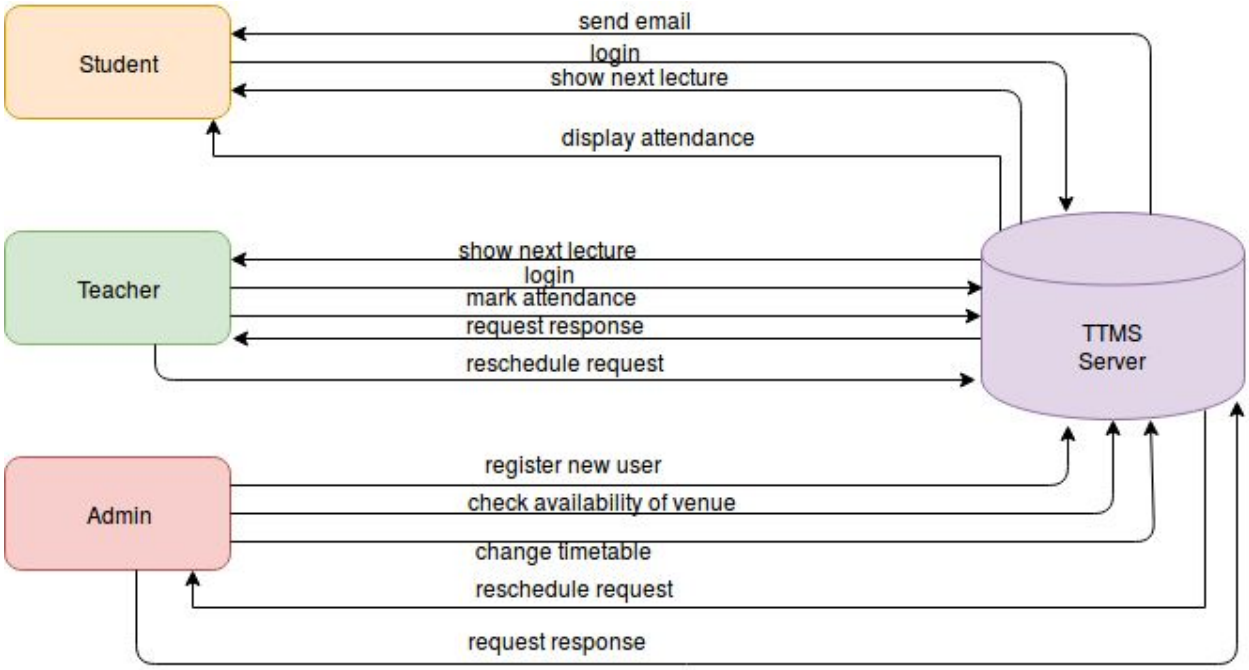


FIGURE 1: System Architecture

Constraints

1. **Memory:** System will have only 10GB space of data server.

2. **Language Requirement:** Software must be only in English.

3. **Budget Constraint:** Due to limited budget, Timetable Management System is intended to be very simple and just to perform basic functionalities. UI is going to be very simple.

4**. Implementation Constraint:** Application should be based on Node JS, JavaScript and CSS.

5. **Reliability Requirements:** System should sync frequently to backup server in order to avoid the data loss during failure, so it can be recovered.

Assumptions and Dependencies

It is assumed that system developed will work perfectly if all the listed requirements are met.

Specific Requirements

A. External Interfaces Requirements

1. User Interfaces

The user interface for system shall be compatible with any type of web browser such as Mozilla Firefox, Google Chrome, and Internet Explorer.

2. Software Interfaces (OS – Windows)

Web Server: Node Server
Database Server: MySQL
Development End: HTML, XML, CSS, JavaScript

3. Hardware Interfaces

Server side

Monitor	Processor	RAM	Disk Space
1024x768	Intel or AMD 2GHZ	4GB	10GB

Client Side

Monitor	Processor	RAM	Disk Space
1024x768	Intel or AMD 2GHZ	512 MB	4GB

TABLE 2: Hardware Requirements

4. Communication Interfaces

The System shall be using HTTP/HTTPS for communication over Internet and for intranet communications, it shall use TCP/IP protocol.

B. Functional Requirements

1. Registration

FR1. The Administration should be able to register the Student and Teacher with their details
FR2. The system should record the basic details like Name, Email, Password, DOB etc. of the customer into the database.
FR3. The system shall send verification message to email.

2. Login

FR4. The system should verify the Student's (and or Teacher's) registration number and password against the values stored in the database.
FR5. After login, customer should be redirected to the Home Page.

3. Student and Teacher Access

FR6. The system should enable **Student** and **Teacher** to check for the details of the next lecture.
FR7. The system should display the timetable for each **Student** and **Teacher**.
FR8. The system should allow the **Teacher** to mark the absence of each **Student**.
FR9. The system should record the number of absentees into database.
FR10. The system should display the attendance for each **Student** for each subject.
FR11. The system should allow the **Teacher** to request for rescheduling a lecture or scheduling an extra lecture.
FR12. The system should send a notification to the **Student** informing them of any changes in the schedule.

4. Admin Access

FR12. The system should allow **Administration** to register a **Student** and a **Teacher**
FR13. The system should assign a timetable to **Student** and **Teacher** automatically
FR14. The system should allow **Administration** to review and accept/reject rescheduling requests
FR13. The system should enable full access to **Administration**.

C. Performance Requirements

NF1. Data in database should be updated within 2 seconds.
NF2. Query results must return results within 5 seconds.
NF3. Load time of UI should not take more than 2 seconds.
NF4. Login Validation should be done within 3 seconds.
NF5. Response to **Student** and **Teacher** inquiry must be done within 5 minutes.

D. Security Requirements

NF6. All external communications between the data's server and client must be encrypted.
NF7. All data must be stored, protected or protectively marked..

E. Safety Requirements

NF9. Database should be backed up every hour.
NF10. Under failure, system should be able to come back at normal operation under an hour.

F. Software System Attributes

- **Correctness:** This system should satisfy the normal Timetable Management operations precisely to fulfil the end user objectives.

- **Efficiency:** Enough resources to be implemented to achieve the particular task efficiently without any hassle.

- **Flexibility:** System should be flexible enough to provide space to add new features and to handle them conveniently.

- **Integrity:** System should focus on securing the **Student** and **Teacher** information and avoid data losses as much as possible.

- **Portability:** The system should run on any Windows environment.

- **Usability:** The system should provide user manual at every level.

- **Testability:** The system should be able to be tested to confirm the performance and clients specifications.

- **Maintainability:** The system should be maintainable.

Appendices

Data Dictionary

Regno: String {Any Valid Character }*
Teacherid: String {Any Valid Character}*
Password : String {Any Valid Character }*

Data Flow Diagrams

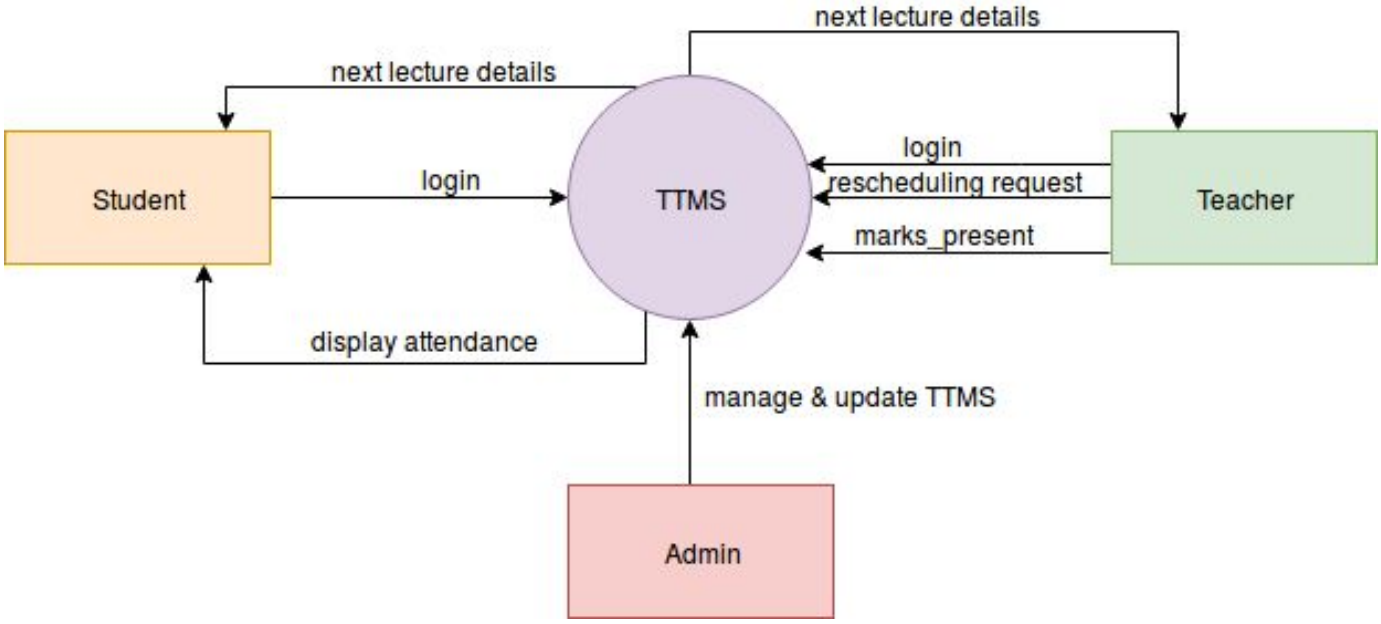


FIGURE 2: DFD Level 0

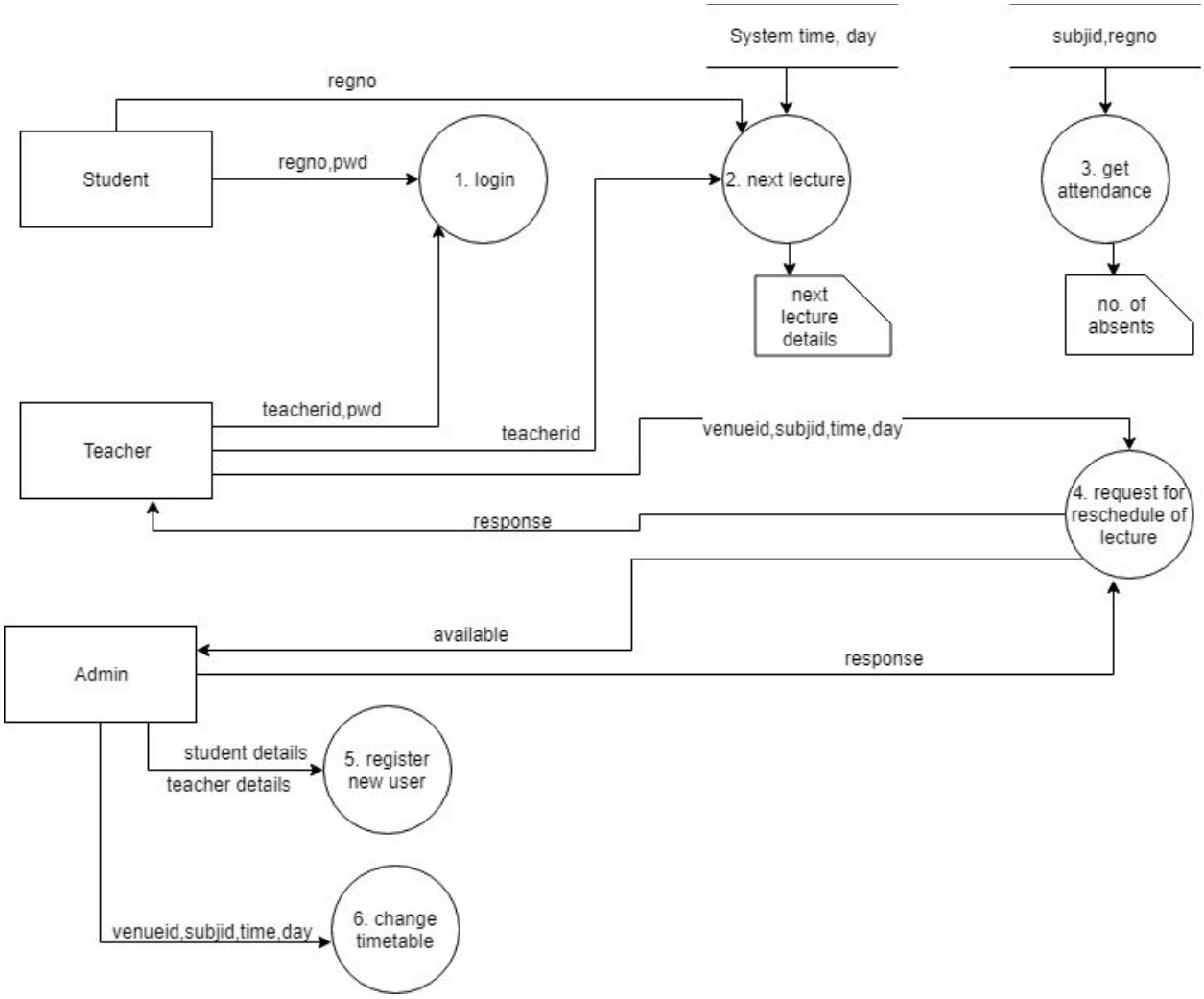


FIGURE 3: DFD Level 1

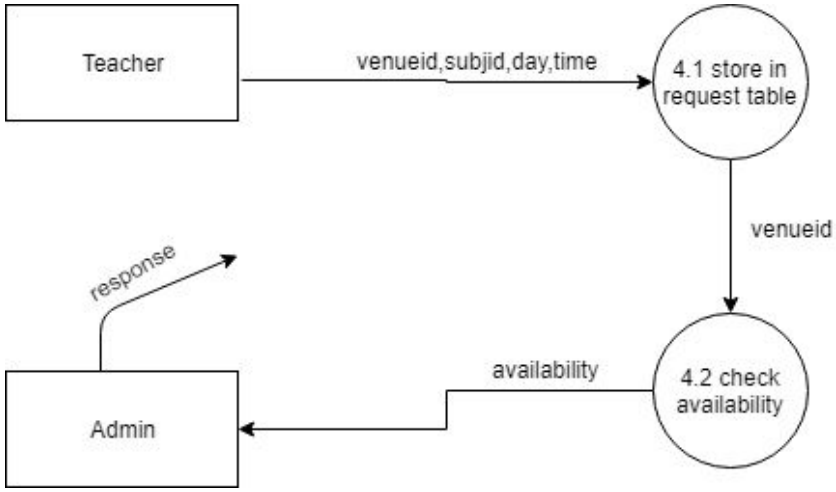


FIGURE 4: DFD Level 2

Use Case Diagram

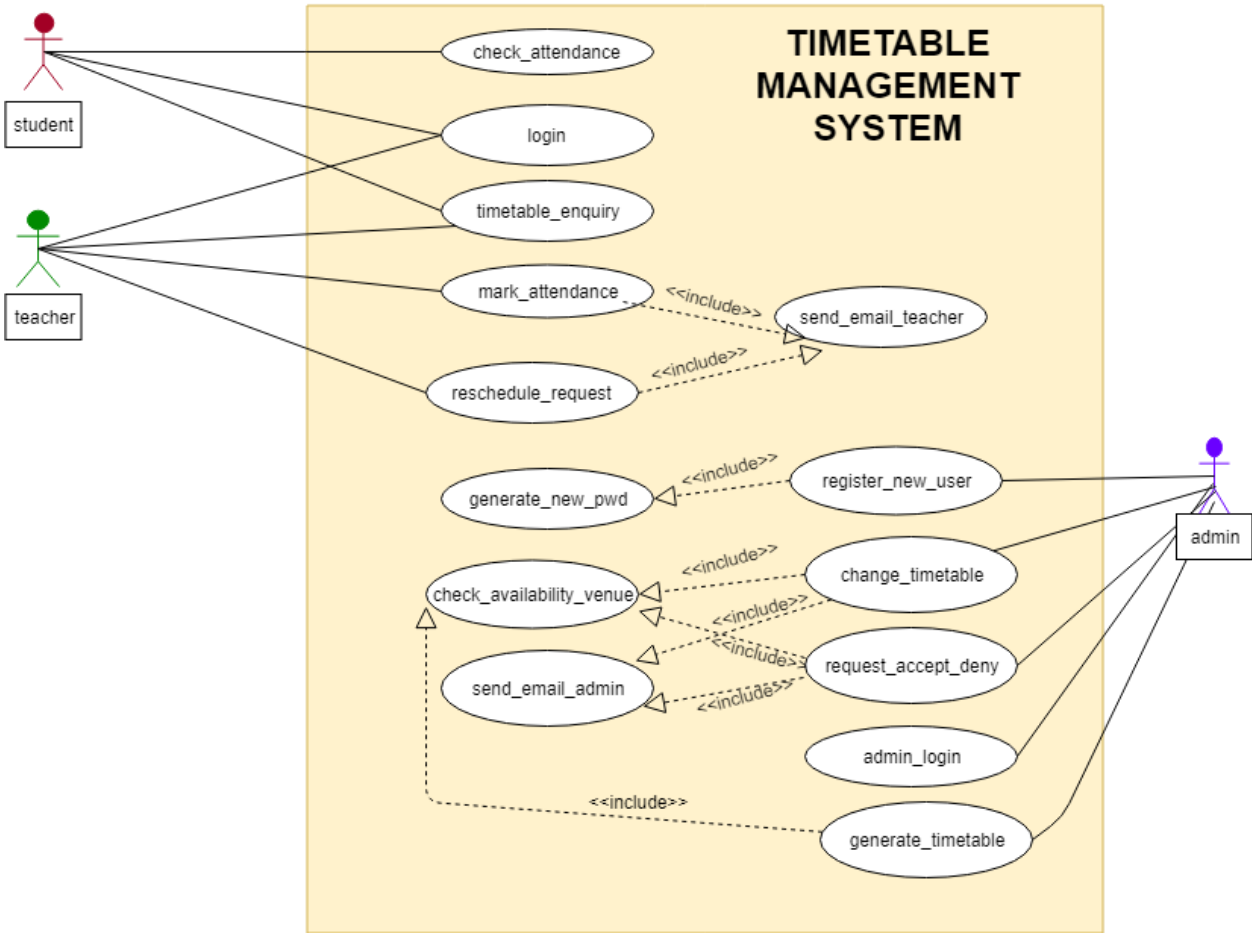


FIGURE 5: Use case diagram

ER Diagram

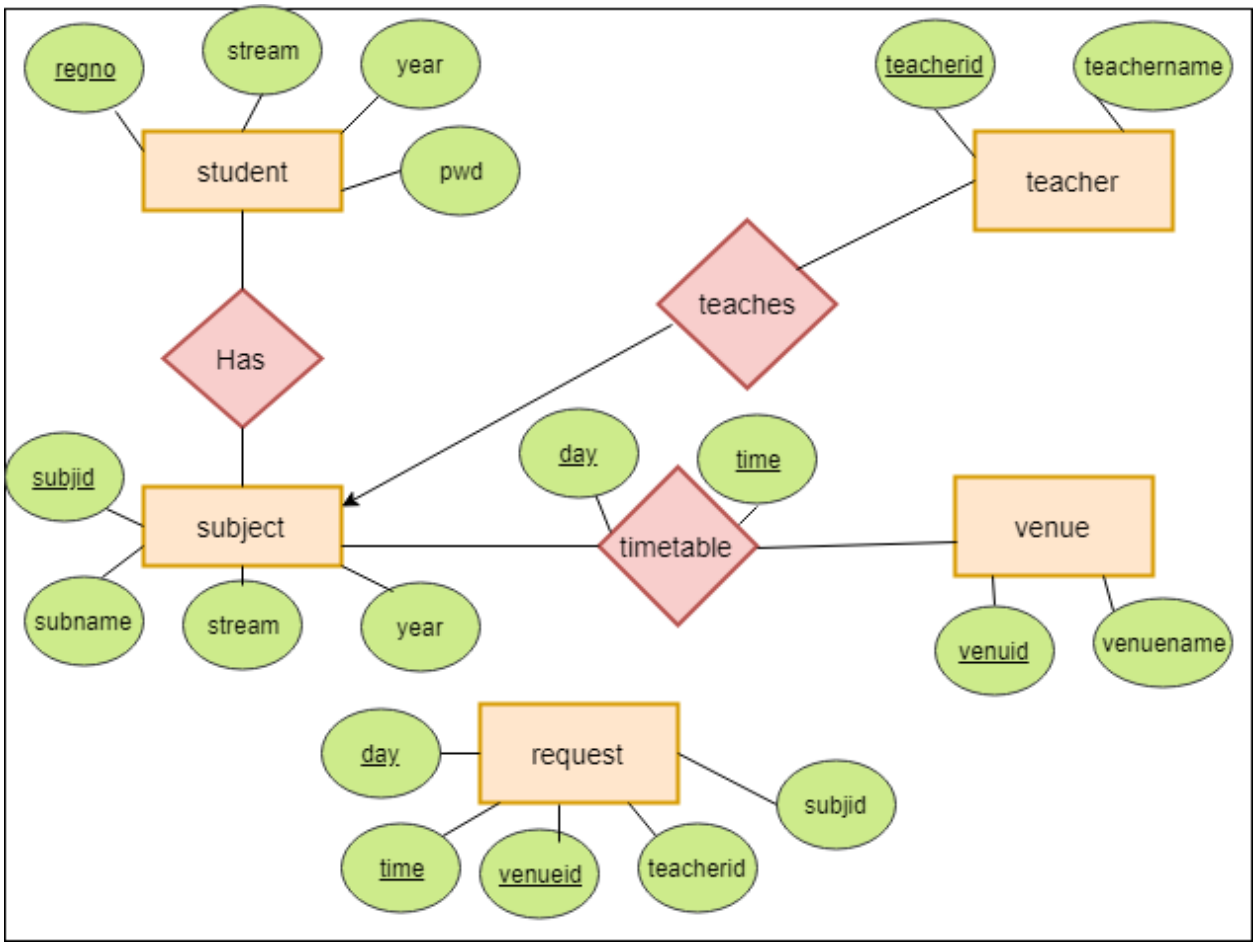


FIGURE 6: ER Diagram

Class Diagram

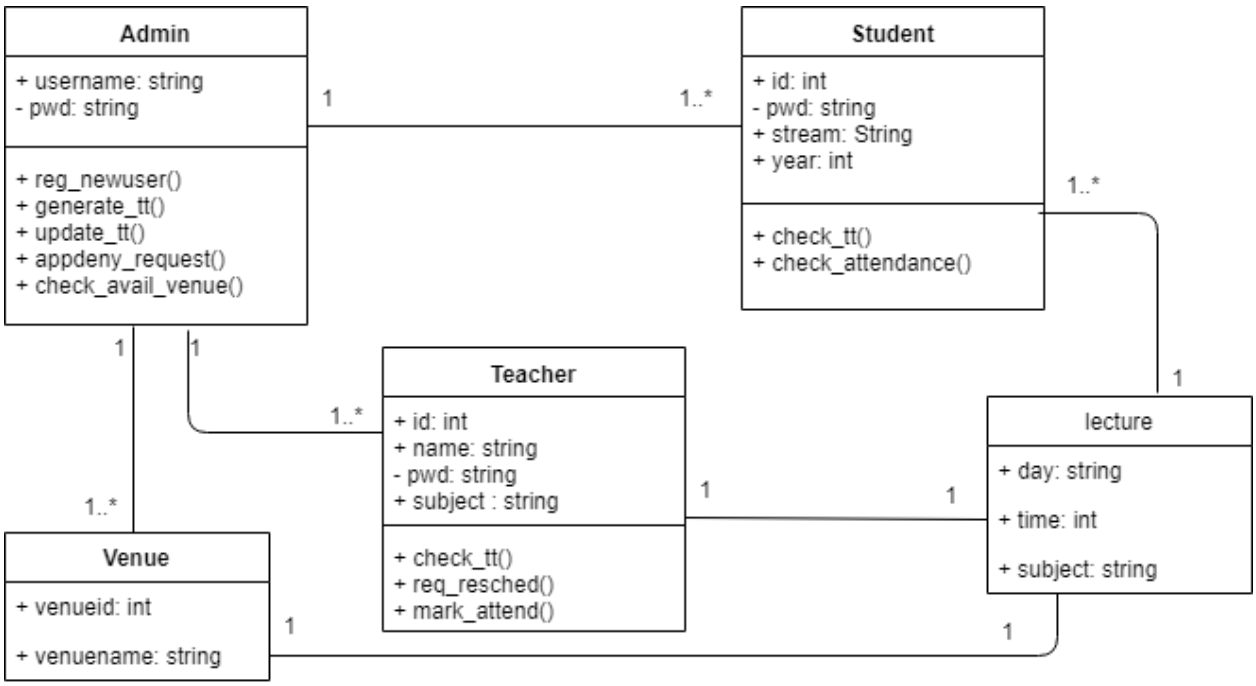


FIGURE 7: Class Diagram

Sequence Diagram

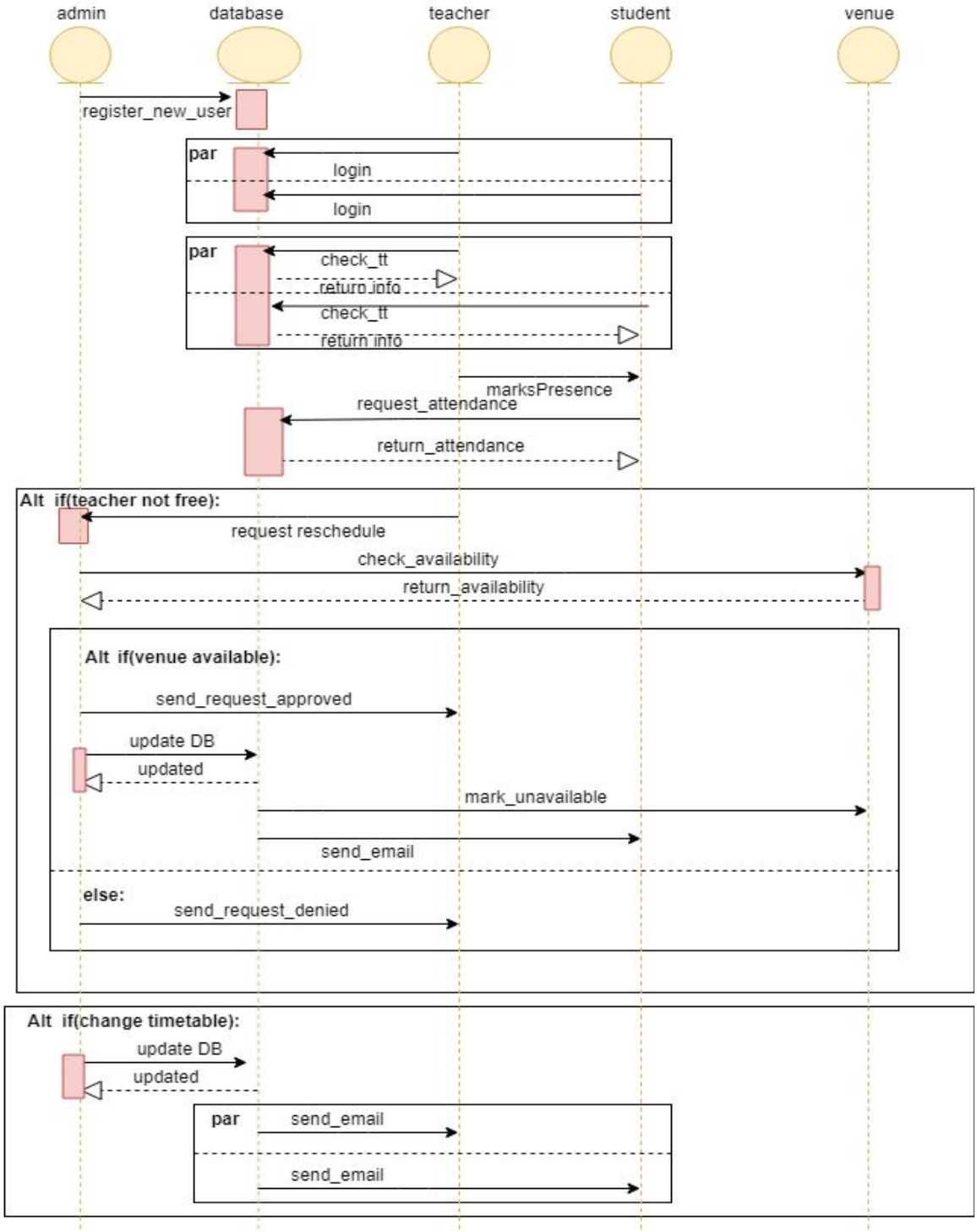


FIGURE 8: Sequence Diagram

Activity Diagram

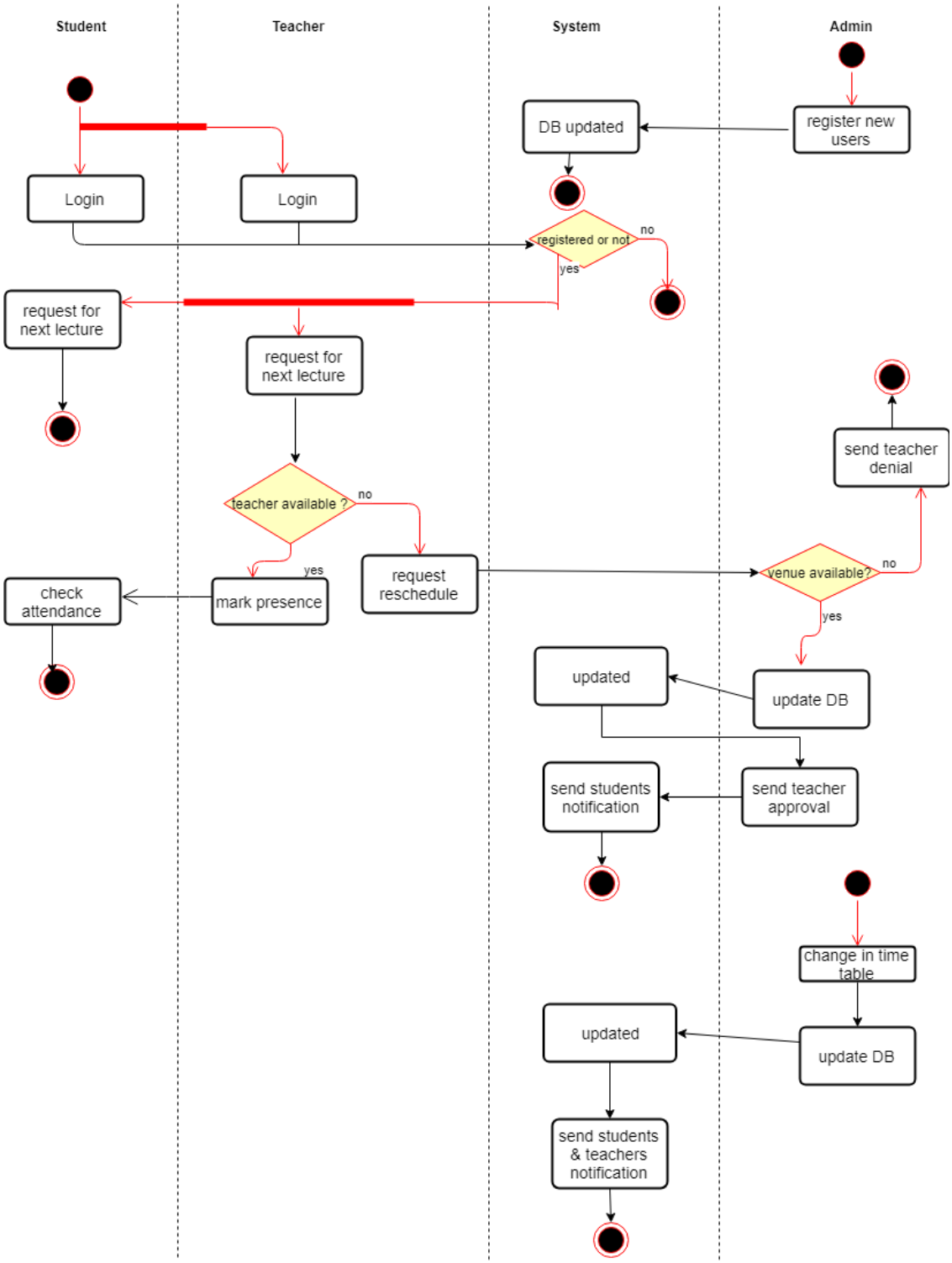


FIGURE 9: Activity Diagram

Algorithms

1. User login

student_login( )

{

var user = new Student( ); **// User is the data model**

user.findOne(regno:req.body.regno) // Find the user from the database

select(regno password)

If (any value==NULL)

{

Output message: Make sure all fields are provided

}

Else if (regno not found)

{

Output Message User not found

}

Else

{

Compare the entered password and the password saved in the database

if (passwords are matched)

{

Output Message Login Successful

Redirect to homepage

}

else

{

Output Message: Password entered is incorrect

}

}

}

teacher_login( )

{

var user = new Teacher( ); **// User is the data model**

user.findOne(teacherid:req.body.teacherid) // Find the user from the database

select(teacherid password)

```
        If (any value==NULL)
        {
                Output message: Make sure all fields are provided
        }

        Else if ( teacherid not found )
         {
                Output Message User not found
        }

        Else
        {
                Compare the entered password and the password saved in the database
                 if ( passwords are matched )
                {
                        Output Message Login Successful
                        Redirect to homepage
                 }
                else
                {
                        Output Message:Password entered is incorrect
                }
        }
}


admin_login( )
 {
         var user = new User( ); // User is the data model
        User.findOne(teacherid:req.body.teacherid) // Find the user from the database
        select(regno password)

        If (any value==NULL)
        {
                Output message: Make sure all fields are provided
        }

        Else
        {
                Compare the entered password and the password saved in the database
                 if ( passwords are matched )
                {
```

```
                    Output Message Login Successful
                    Redirect to homepage
                }
                else
                {
                    Output Message:Password entered is incorrect
                }
        }
}
```

2. User Registration

student_register( )

```
{
        var user = new Student( ); // User is the data model
         user.regno = req.body.regno; //Entering all details about the user
        user.password = req.body.password;
        user.name=req.body.name;
         if ( any value == null or is empty ) // Checking all values are there or not
         {
                Output Message Make sure all fields are provided
         }
        else if ( validation errors == true )
        {
                Output Message Respective Validation Error
         }
         Else
         {
                Save the user in the database
                 Send an activation link to the user's email id
                Output Message User Created
         }
}
```

teacher_register( )

```
{
        var user = new Teacher( ); // User is the data model
         user.teacherid = req.body.regno; //Entering all details about the user
        user.password = req.body.password;
        user.name=req.body.name;
         if ( any value == null or is empty ) // Checking all values are there or not
         {
```

```
                Output Message Make sure all fields are provided
        }
        else if ( validation errors == true )
        {
                Output Message Respective Validation Error
        }
        Else
        {
                Save the user in the database
                Send an activation link to the user's email id
                Output Message User Created
        }
}
```

3. Send rescheduling request

```
req_resched( )
{
        var lecture = new Lecture( );
        var venue=new Venue();
        lecture.day = req.body.day;
        lecture.time = req.body.time;
        lecture.subject = req.body.subject;
        venue.venueid=req.body.venueid;
        if ( any value == null or is empty )
        {
                Output Message Make sure all fields are provided
        }
        else if ( validation errors == true )
        {
                Output Message Respective Validation Error
        }

        Else
        {
                Save the user in the database
                Output Message Request recorded
        }
}
```

4. Respond to rescheduling request

```
check_avail_venue(day,time,venueid)
{
        Find in database record for day,time,venueid
        If (none are found)
        {
                appdeny=True;
        }
        Else
        {
                appdeny=False;
        }
        Return appdeny
}
appdeny_request( )
{
        var lecture = new Lecture( );
        var venue=new Venue();
        select(day time subject venueid)
        lecture.day = day;
        lecture.time =time;
        lecture.subject = subject;
        venue.venueid=venueid;

        appdeny=check_avail_venue(lecture.day,lecture.time,venue.venueid)
        If (appdeny==True)
        {
                Send approval
                Record in database
                Send notification to students
        }
        Else
        {
                Send denial
        }
}
```

5. Check for next lecture

Check_nextlecture() (student)

```
{
        Var presentday=system.day
        Var presenttime=system.time
        Var student=new Student()
        Find in timetable record with day=presentday,time=closest to(presenttime),
student.stream,student.year

        if(found)
        {
                Display details
        }

        Else
        {
                Output message: No lecture soon
        }
}
```

Check_nextlecture() (teacher)

```
{
        Var presentday=system.day
        Var presenttime=system.time
        Var teacher=new Teacher()
        Find in timetable record with day=presentday,time=closest to(presenttime),
teacher.teacherid

        if(found)
        {
                Display details
        }

        Else
        {
                Output message: No lecture soon
        }
}
```

6. Check Attendance

Check_attendance

```
{
        Var student=new Student()
        Var subject=req.body.subject

         if ( any value == null or is empty ) // Checking all values are there or not
         {
                  Output Message Make sure all fields are provided
         }
        else if ( validation errors == true )
        {
                Output Message Respective Validation Error
         }

         Else
         {
                Display attendance
         }
}
```

Testing

1. Black Box Testing

Black box testing, also known as Behavioral **Testing**, is a software **testing** method in which the internal structure/design/implementation of the item being tested is not known to the tester.

1.1 Equivalence Class Partitioning

Equivalence partitioning or equivalence class partitioning (ECP) is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once.

For Rescheduling Request:
Day:

Invalid	Valid	Invalid
<Present day	>present day and <present day +7	>present day +7

Time:

Invalid	valid	Invalid
<9 am	>9 am and < 5pm	> 5 pm

1.2 Decision Table Testing

A decision table is a good way to deal with different combination inputs with their associated outputs and also called cause-effect table.

User login

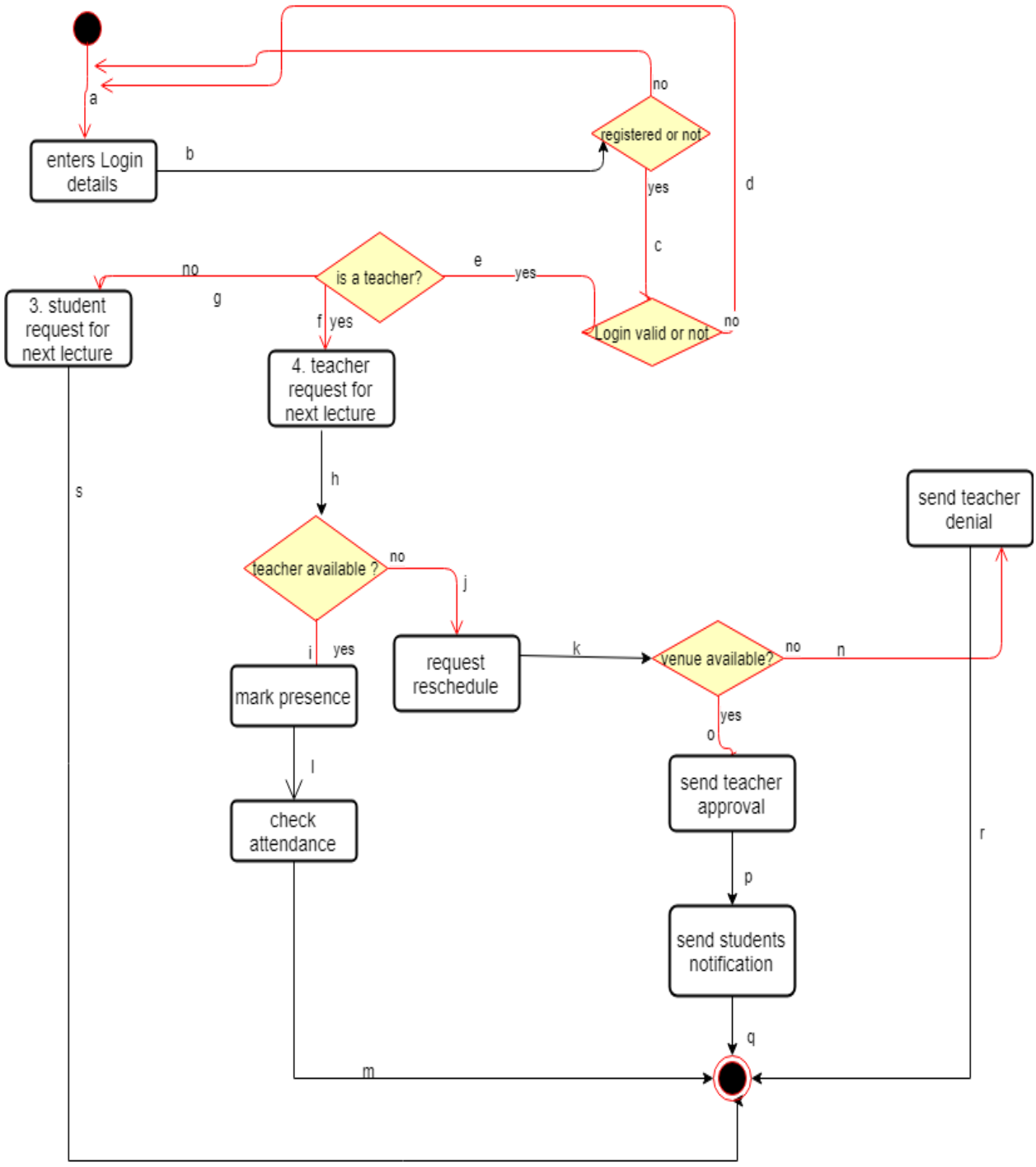
	col1	col2	col3	col4
regno/teacherid	invalid	valid	invalid	valid
password	invalid	invalid	valid	valid
login	Error message	Error message	Error message	successful

2. White Box Testing

White Box Testing is the testing of a software solution's internal coding and infrastructure. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability.

White Box Testing Includes:

1. Statement Coverage
2. Branch Coverage
3. Basis Path


a
enters Login details
b
registered or not
no
yes
d
c
is a teacher?
e
yes
no
g
3. student request for next lecture
f
yes
Login valid or not
no
4. teacher request for next lecture
h
s
send teacher denial
teacher available ?
no
j
i
yes
request reschedule
k
venue available?
no
n
mark presence
yes
o
l
send teacher approval
check attendance
r
p
send students notification
q
m

1. Statement Coverage Testing

Teacher:
Is teacher available: True or False
Test Case 1 : Invalid User, is a teacher, True
Test Case 2 : Valid User, is a student, True
Test Case 3 : Valid User, is a teacher, False
Test Case 4 : Valid User,is a student, False

2. Branch Coverage Testing

Teacher:
Is teacher available: True or False
Test Case 1 : Invalid User, is a teacher, True bcda
Test Case 2 : Valid User, is a student, True bcegs
Test Case 3 : Valid User, is a teacher, True bcefhilm
Test Case 4 : Valid User,is a teacher, False bcefhjkopq

3. Basis Path Testing

Cyclomatic Complexity V(G) of flow Graph:19-14+2=7
Edges: 19
Nodes: 14
Independent Paths To Reach End : 4
1. bcegs
2. bcefhilm
3. bcefhjknr
4. bcefhjkopq