A Project Report on

LEAVE MANAGEMENT SYSTEM FOR SCHOOL

Submitted in partial fulfilment of requirements to

Exposys Data Labs

For the award of internship Certificate

Submitted by

Boggaram Sushma Sri

R.V.R. & J.C. College of Engineering

Guntur

Andhra Pradesh

TABLE OF CONTENTS

Table of contents ……ii

List of Figures ……iv

List of Tables ……..v

Abbreviations ……..vi

1. INTRODUCTION ……1

1.1 Background ……..2

1.2 Problem Statement ……3

1.3 Significance of Work……..4

1.4 Objective ……...6

2. SYSTEM ANALYSIS ……7

2.1 Requirement Specification ……8

2.1.1 Requirement Elicitation……9

2.1.2 User Requirements ……11

2.1.3 System Requirements ……12

2.1.4 Non-Functional Requirements……13

2.2 Use-Case View……..14

2.2.1 Identification of Actors……14

2.2.2 Identification of Use Cases……18

2.2.3 Use Case Diagram……25

2.2.4 Flow of Events ……..32

2.2.5 Prototypes for Application……40

2.2.6 Activity Diagram……42
2.3 Logical View ……..51
2.3.1 Identification of Analysis Classes……..51
2.3.2 Identify responsibilities of Classes……53
2.3.3 Use Case Realizations……55
2.3.4 Sequence Diagrams……..56
2.3.5 Collaboration Diagrams……61
2.3.6 Identification of Attributes of classes……..65
2.3.7 Identification of relationships……67
2.3.8 UML Class Diagram……70
2.3.9 UML State Chart Diagram……76

3. SYSTEM DESIGN……83
3.1 Refining Attributes & methods……84
3.2 Implementation Diagrams……89
3.2.1 Component Diagrams……89
3.2.2 Deployment Diagram……93
3.3 Architecture of the System……..95
3.4 Work Flow……..97
3.4.1 Admin Module……98
3.4.2 Teacher Module……99
3.4.3 Student Module……..100
3.5 Data Base Design……101
4. IMPLEMENTATION……102
4.1 Technologies Used……103
4.2 Implementation of Technical Service Layer……105
5. TESTING……114
5.1 Introduction to Testing……116

5.2 Test cases……119
6. RESULTS……120
7. CONCLUSION……140
8. REFERENCES……142

LIST OF FIGURES

1. Basic Use Case Diagram……29
2. Use Case diagram for Leave Management System……30
3. Use Case Diagram with stereotypes……31
4. Login Prototype…… 40
5. Registration Prototype……41
6. Activity Diagram……48
7. Activity Diagram for Change Password……49
8. Activity diagram for Apply leave……50
9. Sequence Diagram for Login……58
10. Sequence diagram for Apply Leave……59
11. Sequence diagram for Change Password……60
12. Collaboration Diagram for Login……62
13. Collaboration diagram for Apply Leave……63
14. Collaboration Diagram……64
15. Class Diagram for Leave Management System……74
16. Class diagram for Manage Leave……75
17. Class diagram for Apply Leave……75
18. State Chart Diagram for Apply Leave……80
19. State Chart diagram for Manage Leave……81
20. State Chart diagram for Change Password……82
21. Component Diagram for Leave Management System……91
22. Component Diagram for Manage Leaves……92
23. Deployment Diagram for Leave Management System……94

24. 3-Tier Architecture of System……95
25. User Interfaces of Teachers' Login……..134
26. User Interfaces of Students' Login……..137
27. User Interfaces of Admin Login……..140

LIST OF TABLES

1. Requirement Elicitation……..11
2. Description of Actors……..17
3. Description of Use Cases……..24
4. Object Visibility……..85

ABBREVIATIONS

1. HTML – Hyper Text Markup Language
2. CSS – Cascading Style Sheet
3. PHP – Hypertext PreProcessor
4. SQL – Structured Query Language

1

INTRODUCTION

1.1 Background

A **leave management system**, also known as LMS, is a platform that enables a organisation to easily and correctly allocate, track and grant **leave** as well as allow for users to request and track their own **leave**.

In modern practice there are many kinds of leave available to employees like sick, casual, paid and compassionate. Tied together with the different leave structures of organisations and the requirements to request and grant leave can often allocate too much time and resources to manage it.

Today's leave management system is more often found in the form of a web based application due to its easy accessibility and ability to run on almost any device and operating system that has a web browser. Leave management systems make it easy for an user to request leave from their own desk or even from home if sick.

The request for leave will then reflect to the relevant superior for approval, if the leave request is denied, a reason must be entered into the leave management system and the applicant will be notified. This is particularly beneficial to schools.

Having a detailed record of leave is essential for different classes in Schools and highlights the leave managements system's ability to keep a history of leave means details and specific reports can be drawn to identify users(teachers/students) personally, that perhaps are consistently going over their allocated leave or that workaholic that has accumulated too much leave. The data in the leave management system can also be extracted and used to speed up the running of payroll or wages in a school.

A leave management system automates the entire process revolved around leave within a school, saving time and resources by letting employees focus on the important tasks before them and eliminating the traditional need to record and file leave documents.

1.2 Problem Statement

The Easy Leave Management System is an internet based application that can be accessed throughout the organization or a specified group. The periodic crediting of leave is also automated. There are features like cancellation of leave, report generators etc in this tool.

There are registered people in the system. Some are approvers. An approver can also be a requestor. In an organization, the hierarchy could be Students / Teachers / Vice-Principal / Principal etc.

A user should be able to login to the system through the first page of the application, change the password, apply for leave specifying the from and to dates, and approve/ reject the leave applications that are submitted to him / her.

As soon as a leave application / cancellation request / approval / rejection / password-change is made by the person, an automatic email should be sent to the person. The number of days of leave (as per the assumed leave policy) should be automatically credited to everybody. The processing leave should be approved by the system automatically after some days.

1.3 Significance of the Work

In the existing Leave Record Management System, every School follows manual procedure in which faculty enters information in a record book. At the end of each month, Administrator calculates leave of every member which is a time taking process and there are chances of losing data or errors in the records.

The staff needs to submit their leaves manually to their respective authorities. This increases the paperwork & maintaining the records becomes tedious. In schools if a student wants to apply leave he/she have to write a leave letter and submit to respective teacher.

For a single teacher it is difficult to maintain all leave records of all students in a class and some times leads to loss of data or errors. It also increase the paper work. Finally maintaining of all leave records of students and faculty in a school manually is very difficult.

The purpose of Leave Management System for School is to automate the existing manual system by the help of computerized equipments and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. The required software and hardware are easily available and easy to work with.

Leave Management System, as described above, can lead to error free, secure, reliable and fast management system. It can assist the user to concentrate on their other activities rather to concentrate on the record keeping. Thus it will help organization in better utilization of resources. The organization can maintain computerized records without redundant entries. That means that one need not be distracted by information that is not relevant, while being able to reach the information.

The aim is to automate its existing manual system by the help of computerized equipments and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. Basically the project describes how to manage for good performance and better services for the clients.

Leave Management System is a web application for a school which is used for applying leaves by students and teachers and maintain those all leave records easily. A school can maintain all records of students and staff easily by having a particular centralized Database System.

To automate the system of applying leave by students and teachers a web based application is required. A student or a teacher can apply leave in advance by specifying cause of leave and respective authority can approve or reject leave based on requirements. This will help students/teachers to apply leave easily and check their leave status. School also get benefitted by easy maintenance of all records of students and staff leaves.

1.4 Objective

The main objective of the Project on Leave Management System for School is to manage the details of Students, Employees, Leave. It manages all the information about Student / Employee, Leave Type, Leave History. The project is built in such a way that an administrative can add users. A teacher can apply leave which is checked by administrative. A student can apply leave which is managed by teacher. The purpose of the project is to build an application program to reduce the manual work for managing the all leave records of teachers and students.

2

SYSTEM ANALYSIS

2.1 Requirement Specification

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed.

In software engineering, such requirements are often called functional specifications. Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

2.1.1 Requirements Elicitation

RID	REQUIREMENTS	REQUIREMENT NATURE	Moscow
R1	Intuitive graphical user interface to be offered	Functional	Should Have
R2	Leave Management should run in real time	Functional	Must Have
R3	Different roles of employees should be allowed	Functional	Must Have
R4	Information of different roles of employees should be maintained by administrator	Functional	Must Have
R5	Information of students should be maintained by administrator	Functional	Must Have
R6	Administrator has to assign a unique username and password to each employee and student	Functional	Must Have
R7	Login mechanism is imperative	Functional	Must Have
R8	Teacher shall be able to change password	Functional	Could Have
R9	Student shall be able to change password	Functional	Could Have
R10	Both student and teacher shall be able to make leave requests	Functional	Must Have

R11	Display leave balance for each teacher	Functional	Must Have
R12	Display the leave balance for each student	Functional	Must Have
R13	Display leave status of applied leave requests for each teacher	Functional	Could Have
R14	Display leave status of applied leave requests for each student	Functional	Could Have
R15	Display leave history of each teacher	Functional	Must Have
R16	Display leave history of each student	Functional	Must Have
R17	Teacher can cancel his/ her leave request before the leave is verified	Functional	Want to Have
R18	Student can cancel his/her leave request before the leave is verified	Functional	Want to Have
R19	Teacher shall be able to approve or reject a leave request of student	Functional	Must Have
R20	Administrator shall be able to approve or reject a leave request of teacher	Functional	Must Have
R21	Teacher has to maintain the leave history of students of his/ her class	Functional	Should Have
R22	Administrator has to maintain leave details of all teachers	Functional	Must Have

R23	Database interface should be provided	Functional	Must Have
R24	Giving help on different features of the system	Non-Functional	Want to Have

Table 1: Requirement Elicitation

2.1.2 User Requirements

1. User Friendly and easy to understand
2. Fast in Execution(Less execution Time)
3. Maintenance of Records

2.1.3 System Requirements

Hardware requirements

Minimum	:	Pentium IV Processor with 1.2GHZ
Hard Disk	:	500GB
Ram	:	2GB
Display	:	LCD Monitor

Platform Specification

Operating system	:	Any Operating System after WINDOWS 2000
Front End	:	HTML, CSS
Database	:	PHP, MYSQL Database

2.1.4 Non-Functional Requirements

1.Execution qualities:

a. Effciency: The state or quality of being efficient, i.e., able to accomplish something with the least waste of time and effort; competency in performance.

2. Evolution qualities:

a. Testability: The means by which the presence, quality, or genuineness of anything is determined.

b. Extensibility: To enlarge the scope of, or make more comprehensive, as operations, influence etc.

c. Scalability: The ability of something, especially a computer system, to adapt to increased demands.

2.2 Use Case View

2.2.1 Identification of Actors

Actors represent system users. They are NOT part of the system. They represent anyone or anything that interacts with (input to or receive output from) the system.

An actor is someone or something that:

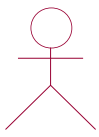
- Interacts with or uses the system.
- Provides input to and receives information from the system.
- Is external to the system and has no control over the use cases.
- Actors are discovered by examining:
- Who directly uses the system
- Who is responsible for maintaining the system
- External hardware used by the system
- Other systems that need to interact with the system

The needs of the actor are used to develop use cases. This insures that the system will be what the user expected.

Graphical Depiction:

An actor is a stereotype of a class and is depicted as a “stickman” on a use-case diagram.

For example:



USER

Naming:

The name of the actor is displayed below the icon.

Questions to help to identity actors

1. Who is interested in a certain requirements?
2. Where is the system used within the organization?
3. Who will benefit from the use of the system?
4. Who will supply the system with information, use this information, and remove this information?
5. Who will support and maintain the system?
6. Does the system use an external resource?
7. Does one person play several different roles?
8. Do several people play the same role?
9. Does the system interact with a legacy system?

From above mentioned information the actors mainly involved in the Leave Management System for School are:

1. Administrator
2. Teacher
3. Student

Brief Description of Actors

a. Teacher:

Actor	Brief Description
Teacher	The main objective of this teacher is to participate in the leave management process. The teacher will place the leave request stating the type of leave and also the days for which the leave is being applied. Then Administrator will verify the leave request placed by the teacher and he/she will approve or reject it. After verification of the leave request by the Administrator, status of the leave request is sent to the teacher. The teacher will verify the leave request placed by student and he/she will approve or reject it. After verification of leave request by the teacher, status of leave request is sent to the student.

b. Student:

Actor	Brief Description
Student	The main objective of this student is to participate in the leave management process. The student will place the leave request stating the type of leave and also the days for which the leave is being applied. Then teacher will verify the leave request placed by the teacher and he/she will approve or reject it. After verification of the leave request by the Teacher, status of the leave request is sent to the student.

c. Administrator:

Actor	Brief Description
Administrator	He/ she register the users. He looks over the leave management process, verifies the leave requests of Teachers. He maintains the database of leave requests and as well as the details of the users.

Table 2 : Description of Actors

2.2.2 Identification of Use-Cases and Sub Use-Case

Use-cases diagrams graphically represent system behavior (use cases). These diagrams present a high level view of how the system is used as viewed from an outsider's (actor's) perspective. A use-case diagram may contain all or some of the use cases of a system.

A use-case diagram can contain:

- Actors ("things" outside the system)
- Use cases (system boundaries identifying what the system should do)
- Interactions or relationship between actors and use cases in the system including the associations, dependencies, and generalizations

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implemented.

In its simple form, a use case can be described as a specific way of using the system from a user's (actor's) perspective. A more detailed description might characterize a usecase as:

- A pattern of behavior the system exhibits.
- A sequence of related transactions performed by an actor and the system.
- Delivering something of value to the actor

Use case provides a means to:

- Capture system requirements
- Communicate with the end-users and domain experts
- Test the system

Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system. Since all the needs

of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together this use case collection specifies all the ways the system.

Use case is a sequence of transactions performed by a system that yields a measurable result of values for a particular actor. The use cases are all the ways the system may be used.

The UML notation for use case is: **ellipse**

Use-case

The basic shape of a use case is an ellipse.

Naming:

- A usecase may have a name, although it is typically not a simple name. It is often written as an informal text description of the actors and the sequences of events between objects. Usecase names often start with a verb. For example, names of possible use cases for an ATM machine might include Dispense Cash or Transfer Funds.
- The name of the use case is displayed below the icon.

The set of questions used to identify the use-cases are:

1. What are the tasks of each actor?
2. Will any actor create, store, change, remove or read information in the system?
3. What use cases will create, store, change, remove, or read this information?
4. Will any actor need to inform the system about sudden, external changes?
5. Does any actor need to be informed about certain occurrences in the system?
6. What use cases will support or maintain the system?
7. Can all functional requirements be performed by the use cases?

Purpose of use cases:

- Well-structured use cases denote essential system or subsystem behaviors only, and are neither overly general nor too specific.
- A use case describes a set of sequences, in which each sequence represents the interaction of the things outside the system (its actors) with the system itself (and its key abstraction).
- Use cases specify desired behavior: they do not dedicate how the behavior will be carried out. It helps you to communicate with your developers (who build system that satisfies your requirements) without getting hung up o details.
- A use cases represents a functional requirement of the system a whole.
- A use case carries out some tangible amount of work. From the perspective of a given actor, a use case does something that's of values to an actor. Such as calculate a result, generate a new object or change the state of another object.
- Use cases represent an external view of the system.

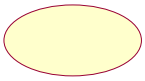
Use-cases identified for our system are:

- Login
- Change password
- Apply leave
- Check Leave balance
- Check Leave Status
- Manage leaves
- Cancel Leave Request
- Leave History
- Register Employees
- Logout

1. Use-case name: **Login**

Use case name	Brief Description
Login	The admin / teacher / student must login into to the leave management system .only when they can apply or manage leaves.

UML Notation:



Login

2. Use-case name: **Change Password**

Use case name	Brief Description
Change Password	The teacher / student can change their password by verifying the old password

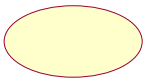
UML Notation:

Change Password

3. Use-case name: **Apply Leave**

Use case name	Brief Description
Apply Leave	The teacher / student can apply leave based on their leave balance

UML Notation:



Apply Leave

4. Use-case name: **Check Leave balance**

Use case name	Brief Description
Check Leave balance	The teacher / student can check their leave balance

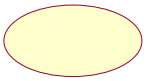
UML Notation:

Check Leave balance

5. Use-case name: **Check Leave Status**

Use case name	Brief Description
Check Leave Status	The teacher/student can check their leave status after applying leave requests.

UML Notation:



Check Leave Status

6. Use-case name: **Manage Leaves**

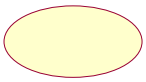
Use case name	Brief Description
Manage Leaves	The Teacher/Administrator can manage the leave requests , he can either approve or reject the request.

UML Notation:

7. Use-case name: **Cancel Leave Request**

Use case name	Brief Description
Cancel Leave request	The teacher/student can cancel their leave requests before they are verified by the approver .

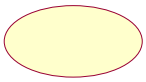
UML Notation:



8. Use-case name**: Leave History**

Use case name	Brief Description
Leave History	The teacher/student can know the details of all the leave requests applies till date .

UML Notation:

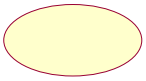


Leave History

9.Use-case name**: Register Employees**

Use case name	Brief Description
Register Employees	The Administrator has to register the students /teachers of the school and allot them a username and password.

UML Notation:



Register Employees

10. Use-case name: **Logout**

Use case name	Brief Description
Logout	The admin / teacher / student can logout of the leave management system performing their required functionality.

Table 3 : Description of Use cases

UML Notation:

Logout

2.2.3 Building Requirements Model through Use Case Diagram

Use Case Diagram

Definition: Use-case diagrams graphically represent system behavior (use cases). These diagrams present a high level view of how the system is used as viewed from an outsider's (actor's) perspective. A use-case diagram may contain all or some of the use cases of a system.

A use-case diagram can contain:

- Actors ("things" outside the system)
- Use cases (system boundaries identifying what the system should do)
- Interactions or relationships between actors and use cases in the system including the associations, dependencies, and generalizations.

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implemented.

Relations:

Association Relationship:

An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. Associations are the most general of all relationships and consequentially the most semantically weak. If two objects are usually considered independently, the relationship is an association

By default, the association tool on the toolbox is unidirectional and drawn on a diagram with a single arrow at one end of the association. The end with the arrow indicates who or what is receiving the communication.

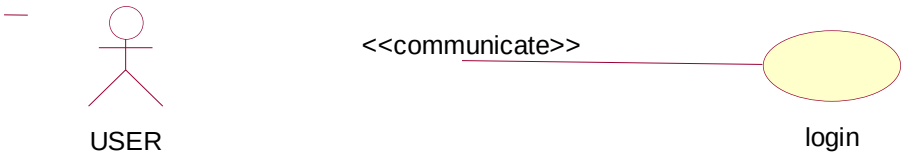
Bi-directional association:

If you prefer, you can also customize the toolbox to include the bi-directional tool to the use-case toolbox.

Graphical Depiction:

An association relationship is an orthogonal or straight solid line with an arrow at one end:

In An Association Relationship, we can provide Stereotype Communicate also as shown below:



Dependency Relationship:

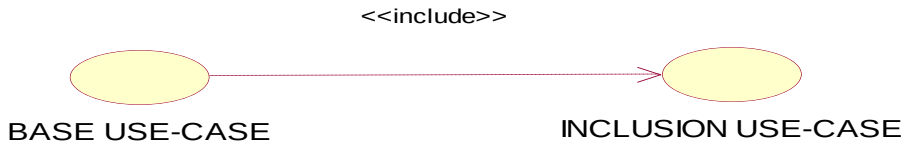
A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning. Typically, on class diagrams, a dependency relationship indicates that the operations of the client invoke operations of the supplier.

We can provide here

1. Include Relationship.
2. Extend Relationship

- There are two types of dependency relationships that may exist between use cases: *include relationship* and *extend relationship.*
- Multiple use cases may share pieces of the same functionality. This functionality is placed in a separate use case rather than documenting it in every use case that needs it.
- *Include* relationships are created between the new use case and any other use case that "uses" its functionality.

An include relationship is a stereotyped relationship that connects a base use case to an inclusion use case. An include relationship specifies how behavior in the inclusion use case is used by the base use case.



Extend Relationship:

An extend relationship is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case. Extend relationships between use cases are modelled as dependencies by using the Extend stereotype.

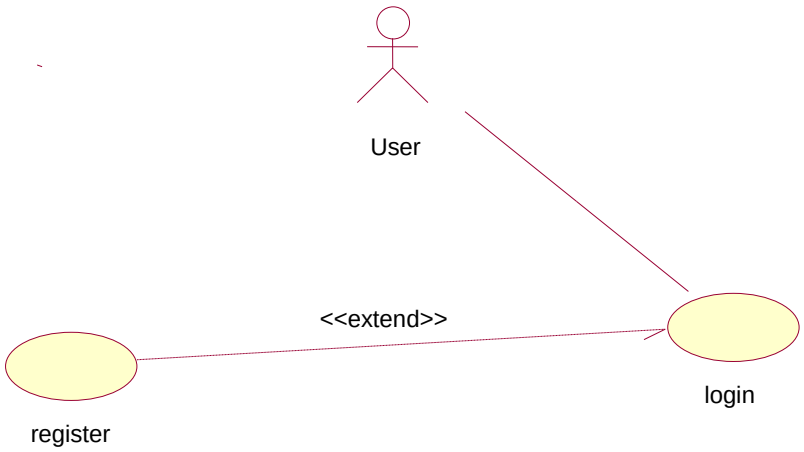
An *extend* relationship is used to show

- Optional behaviour.
- Behavior that is run only under certain conditions such as triggering an alarm
- Several different flows that may be run based on actor selection
- An *extend* relationship is drawn as a dependency relationship that points from the extension to the base use case

The extend relationship sample demonstrates how you can use an extend relationship to connect use cases. The sample illustrates two important aspects of extend relationships:

- An extend relationship shows optional functionality or system behavior.
- A base use case does not need to acknowledge any specific extended use cases

The sample below shows a diagram containing an actor who logins. The Customer has the option of registering as shown through the extend relationship.



Finally we can conclude,

- **«extend»** is used when you wish to show that a use case provides additional functionality that may be required in another use case.
- **«include»** applies when there is a sequence of behavior that is used frequently in a number of use cases, and you want to avoid copying the same description of it into each use case in which it is used.

Basic Use Case Diagram for leave Management System

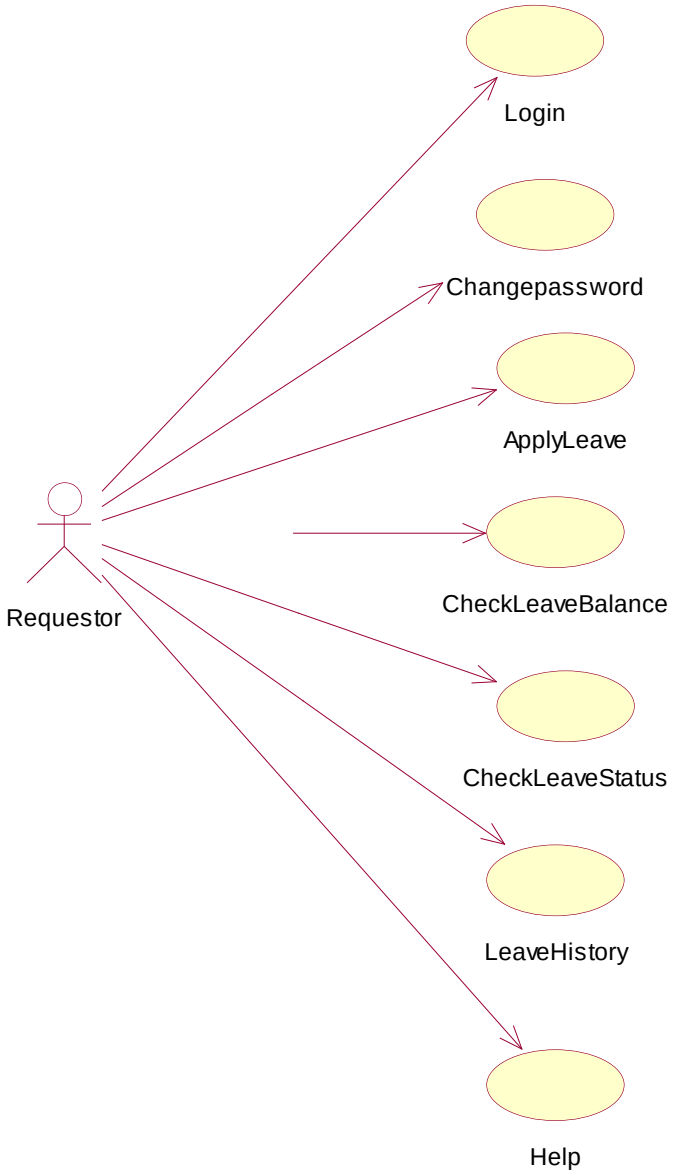


Figure 1: Basic Use Case Diagram

Use Case Diagram of Leave Management for School

Figure 2: Use Case Diagram of Leave Management System for School

Use Case Diagram with Stereotypes

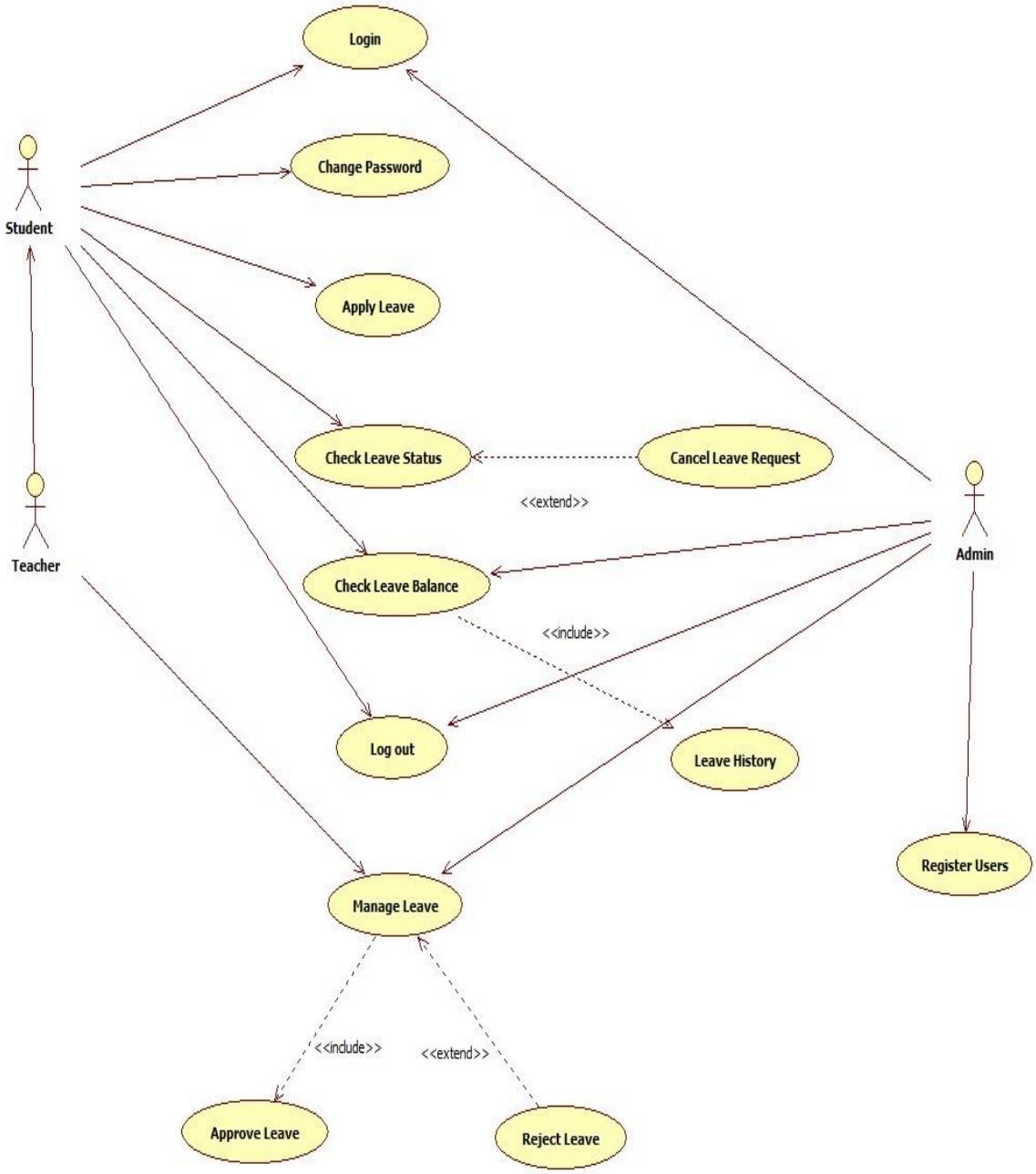


Figure 3: Use Case Diagram with Stereotypes

2.2.4 Flow of Events

A flow of events is a sequence of transactions (or events) performed by the system. They typically contain very detailed information. Flow of events document is typically created in the elaboration phase.

Each use case is documented with flow of events

- A description of evens needed to accomplish required behavior
- Written in terms of what the system should do, NOT how it should do it
- Written in the domain language, not in terms of the implementation

A flow of events should include:

- When and how the use case stars and ends
- What interaction the use case has with the actors
- What data is needed by the use case
- The normal sequence of events for the use case
- The description of any alternate or exceptional flows

The flow of events for a use case is contained in a document called the Use Case Specification. Each project should use a standard template for the creation of the Use Case Specification. Including the following

1. Use Case Name, brief description
2. Flow of Events:
    - Basic flow
    - Alternate flow
    - Pre-Conditions
    - Extension Points

Use case specification for Login:

1. Use case Number: 1

2. Use case Name: login

3. Brief description: The users must login into the leave management system in order to apply and manage leaves.

4. Precondition: Registration of user(Teacher/Student).

5. Post condition: User will be able to apply leave.

6. Flow of events:

6.1. Basic flow:

- The user will go to user tab. Then the user (teacher/student) will fill the details like username and password.
- If the username or password is incorrect then the system will execute the alternative flow.

6.2. Alternative flow:

Informs the user to enter correct username or password.

7. Extension Point: nothing

Use case specification for Change Password:

1. Use case Number: 2

2. Use case Name: change password

3. Brief description: Users can change their password by entering old password.

4. Precondition: Registration of user.

5. Post condition: User can login with the new password.

6. Flow of events:

6.1. Basic flow:

- The user will go to change password tab. The user will fill the details like old password, new password and re-enter of new password.
- If the old password is incorrect or both the new passwords do not match then the system will execute the alternative flow. Otherwise a mail is sent and password is changed.

6.2. Alternative flow:

Informs user to enter the correct old password or to enter the both new passwords as same.

7. Extension Point: Nothing

Use case specification for Apply Leave:

1. Use case Number: 3

2. Use case Name: Apply leave

3. Brief description: The user can apply leave based on their leave balance.

4. Precondition: Registration of user.

5. Post condition: Leave is managed and leaves balance and history is updated.

6. Flow of events:

6.1. Basic flow:

- The user will login and go to apply leave tab. If the login is incorrect then the system will execute alternative flow (6.2.1).
- The user will apply the leave by filling the details like from date, to date and type of leave. If the details are correct

according to the conditions then the leave is applied and a mail is sent. Otherwise the alternative flow (6.2.2) will be executed.

- Then this leave application goes to the teacher if student logged in and goes to admin if teacher logged in.

6.2. Alternative flow:

6.2.1. Invalid username or password: Identifies the user to enter correct username or password.

6.2.2. Invalid leave details: Indicate the user to enter correct leave details according to the conditions mentioned.

7. Extension Point: Requestor can cancel the leave request before approval.

Use case specification for Check leave balance:

1. Use case Number: 4

2. Use case Name: Check leave balance

3. Brief description: The user can check their leave balance.

4. Precondition: Registration of users.

5. Post condition: Leave balance is displayed (i.e. the details like type of leave and number of leaves remaining are displayed).

6. Flow of events:

6.1. Basic flow: The user will login and go to leave balance tab. If the login is incorrect then the system will execute alternative flow.

6.2. Alternative flow: Invalid username or password Informs the user to enter correct username or password.

7. Extension Point: nothing.

Use case specification for Check Leave Status:

1. **Use case Number:** 5
2. **Use case Name:** Check leave status
3. **Brief description:** The users can check their leave status after applying leave requests
4. **Precondition:** The user should apply the leave.
5. **Post condition:** Leave status table is displayed.
6. **Flow of events:**
   - **6.1. Basic flow:** The user will login and go to leave status tab. If the Login is correct then leave status is displayed. Otherwise the system will execute the alternative flow.
   - **6.2. Alternative flow:** Invalid username or password **:** Indicates the user to enter correct username or password.
7. **Extension Point:** User can cancel the leave request before the approval.

Use case specification for Manage Leave:

1. **Use case Number:** 6
2. **Use case Name:** Manage leave
3. **Brief description:** The teacher can manage the leave requests (if student applied for leave) and admin can manage the leave requests (if teacher applied for leave), he can either approve or reject the request.
4. **Precondition:** Leave must be applied by student or teacher.
5. **Post condition:** Leave balance and status will be updated.
6. **Flow of events:**
   - **6.1. Basic flow:** The approver (admin / teacher) has to login and go to manage leave tab. If the login is unsuccessful then the alternative flow is executed otherwise approver profile will be opened. Then

the approver will manage leave i.e. we can either approve or reject.

6.2. Alternative flow: Invalid username or password:

Indicates user to enter username or password.

7. Extension Point: nothing

Use case specification for Cancel Leave Request:

1. Use case Number: 7

2. Use case Name: cancel leave request

3. Brief description: The user (teacher / student) can cancel their leave requests before they are verified by the approver.

4. Precondition: User has to apply for the leave.

5. Post condition: Leave status and leave history is updated.

6. Flow of events:

6.1. Basic flow:

- User will login. If it is successful then go to leave status tab. Otherwise the system will execute the alternative flow (6.2.1).
- The user may cancel the leave request before it is not approved by the approver or admin.

6.2. Alternative flow:

6.2.1. Invalid username or password:

Indicates the username or password is incorrect.

7. Extension Point: nothing

Use case specification for Check Leave History:

1. **Use case Number:** 8
2. **Use case Name:** check leave history
3. **Brief description:** The users can know the details of all the leave requests applied till date.
4. **Precondition:** Registration of user.
5. **Post condition:** Leave history (i.e., number of leaves taken, type of leave and leave status is displayed).
6. **Flow of events:**

   6.1. Basic flow: The user will login and got of leave history tab.

   If the login is incorrect then system will execute alternative flow.

   6.2. Alternative flow: Invalid username or password:

   Informs the user to enter correct username or password.
7. **Extension Point:** nothing

Use case specification for Register Employees:

1. **Use case Number:** 9
2. **Use case Name:** register users
3. **Brief description:** The administrator has to register the users of the organization and allot then a username and password uniquely.
4. **Precondition:** User not registered in prior.
5. **Post condition:** User is able to register with the new username and password.
6. **Flow of events:**

   6.1. Basic flow:

   - The administrator has to go to admin tab. Then the user has to fill the username and password. If the details are

correct then the leave policy is displayed. Otherwise the alternative flow (6.2.1) is executed.

- If the admin logins successfully the he/she will register the users. If the user is already registered then the system will execute the alternative flow (6.2.2).

6.2. Alternative flow:

6.2.1. Invalid username or password:

Indicates the user to correct username or password.

6.2.2. A message as "user has been registered already".

7. Extension Point: nothing

Use case specification for Logout:

1. Use case Number: 10

2. Use case Name: logout

3. Brief description: Users (Admin/Teacher/Student) can logout of the leave management system after performing their required functionality.

4. Precondition: The user should login.

5. Post condition: The user will move out from the leave management system.

6. Flow of events:

6.1. Basic flow: The user should login. If the login is successful then go to logout tab . Otherwise the system will executes the alternative flow.

6.2. Alternative flow: Invalid username or password:

Indicates the user to enter correct username or password.

7. Extension Point: nothing

2.2.5 Sample Prototypes for Application

1) In software development, a prototype is a rudimentary working model of a product or information system, usually built for demonstration purposes or as part of the development process. In the systems development life cycle (SDLC) Prototyping Model, a basic version of the system is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed.

2) In prototype-based programming, a prototype is an original object; new objects are created by copying the prototype.

3) In hardware design, a prototype is a "hand-built" model that represents a manufactured (easily replicable) product sufficiently for designers to visualize and test the design.

The word *prototype* comes from the Latin words *proto*, meaning *original*, and *typus*, meaning *form or model*. In a non-technical context, a prototype is an especially representative example of a given category.

Login Prototype:

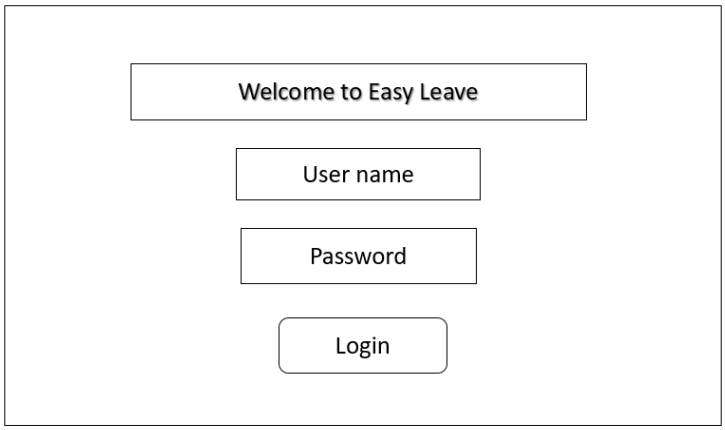


Figure 4: Login Prototype

Registration Form:

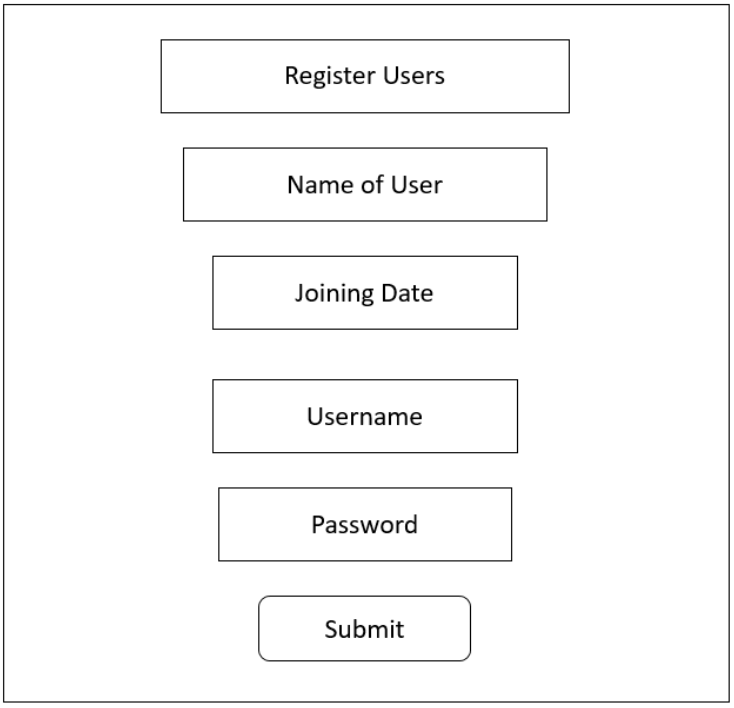


Figure 5: Registration Prototype

2.2.6 Activity Diagram

An Activity diagram is a variation of a special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations.

The purpose of Activity diagram is to provide a view of flows and what is going on inside a use case or among several classes.

You can also use activity diagrams to model code-specific information such as a class operation.

Activity diagrams are very similar to a flowchart because you can model a workflow from activity to activity.

An activity diagram is basically a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities.

- Activity diagrams also may be created at this stage in the life cycle. These diagrams represent the dynamics of the system.
- They are flow charts that are used to show the workflow of a system; that is, they show the flow of control from activity to activity in the system, what activities can be done in parallel, and any alternate paths through the flow.
- At this point in the life cycle, activity diagrams may be created to represent the flow across use cases or they may be created to represent the flow within a particular use case.
- Later in the life cycle, activity diagrams may be created to show the workflow for an operation.

Activities:

Activity diagrams contain activities, transitions between the activities, decision points, and synchronization bars. An activity represents the performance of some behavior in the workflow. In the UML, activities are represented as rectangles with rounded edges, transitions are drawn as directed arrows, decision points are shown as diamonds, and synchronization bars are drawn as thick horizontal or vertical bars as shown in the following.

The activity icon appears as a rectangle with rounded ends with a name and a component for actions

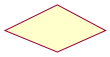
.

Transitions:

Transitions are used to show the passing of the flow of control from activity to activity. They are typically triggered by the completion of the behavior in the originating activity. Transition connects activities with other model elements and object flows connect activities with objects. They are typically triggered by the completion of the behavior in the originating activity.

Decisions:

When modeling the workflow of a system it is often necessary to show where the flow of control branches based on a decision point. The transitions from a decision point contain a guard condition, which is used to determine which path from the decision point is taken. Decisions along with their guard conditions allow you to show alternate paths through a work flow.



Decision Point

End State:

An end state represents a final or terminal state on an activity diagram or state chart diagram. Place an end state when you want t explicitly show the end of a workflow on an activity diagram or the end of a state chart diagram. Transitions can only occur into an end state; however, there can be any number of end states per context.

End state

Start State:

A start state (also called an “initial state”) explicitly show the beginning of a workflow on an activity diagram or the beginning of the execution of a state machine on a state chart diagram.

Start state

Swim lanes:

Swim lanes may be used to partition an activity diagram. This typically is done to show what person or organization is responsible for the activities contained in the swim lane.

Swim lanes are helpful when modeling a business workflow because they can represent organizational units or roles within a business model. Swim lanes are very similar to an object because they provide a way to tell who is performing a certain role. Swim lanes only appear on activity diagrams.

When a swim lane is dragged onto an activity diagram, it becomes a swim lane view. Swim lanes appear as small icons in the browse while a swim lane views appear between the thin, vertical lines with a header that can be renamed and relocated.

Synchronization Bars:

In a workflow there are typically some activities that may be done in parallel. A synchronization bar allows *you* to specify what activities may be done concurrently.

Synchronization bars are also used to show joins in the workflow; that is, what activities must complete before processing may continue.

Means, a synchronization bar may have many incoming transitions and one outgoing transition, or one incoming transition and many outgoing transitions.

Horizontal synchronization

Vertical synchronization

The activity diagram for overall system is:

- At first user with his credentials to see the details of leave in his page.
- To apply a leave he chooses type of leave and from and to dates.
- If the particular leave or no. of days does not match criteria he logs out.
- He applies the leave and request is send to respective higher authority.
- If leave is approved he logs out.

Modeling a workflow in an activity diagram can be done several ways; however, the following steps present just one logical process:

- Identify a workflow objective. Ask, "What needs to take place or happen by the end of the workflow? What needs to be accomplished?" For example, if your activity diagram models the workflow of ordering a book from an online bookstore, the goal of the entire workflow could be getting the book to the customer.
- Decide the pre and post-conditions of the workflow through a start state and an end state. In most cases, activity diagrams have a flowchart structure so start and end states are used to designate the beginning and ending of the workflow. State and end states clarify the perimeter of the workflow.

- Define and recognize all activities and states that must take place to meet your objective. Place and name them on the activity diagram in a logical order.
- Define and diagram any objects that are created or modified within your activity diagram. Connect the objects and activities with object flows.
- Decide who or what is responsible for performing the activities and states through swim lanes. Name each swim lane and place the appropriate activities and states within each swim lane.
- Connect all elements on the diagram with transitions. Begin with the "main" workflow.

- Place decisions on the diagram where the workflow may split into an alternate flow. For example, based on a Boolean expression, the workflow could branch to a different workflow.
- Evaluate your diagram and see if you have any concurrent workflows. If so, use synchronizations to represent forking and joining.
- Set all actions, triggers and guard conditions in the specifications of each model element.

Activity Diagram for Leave Management System:

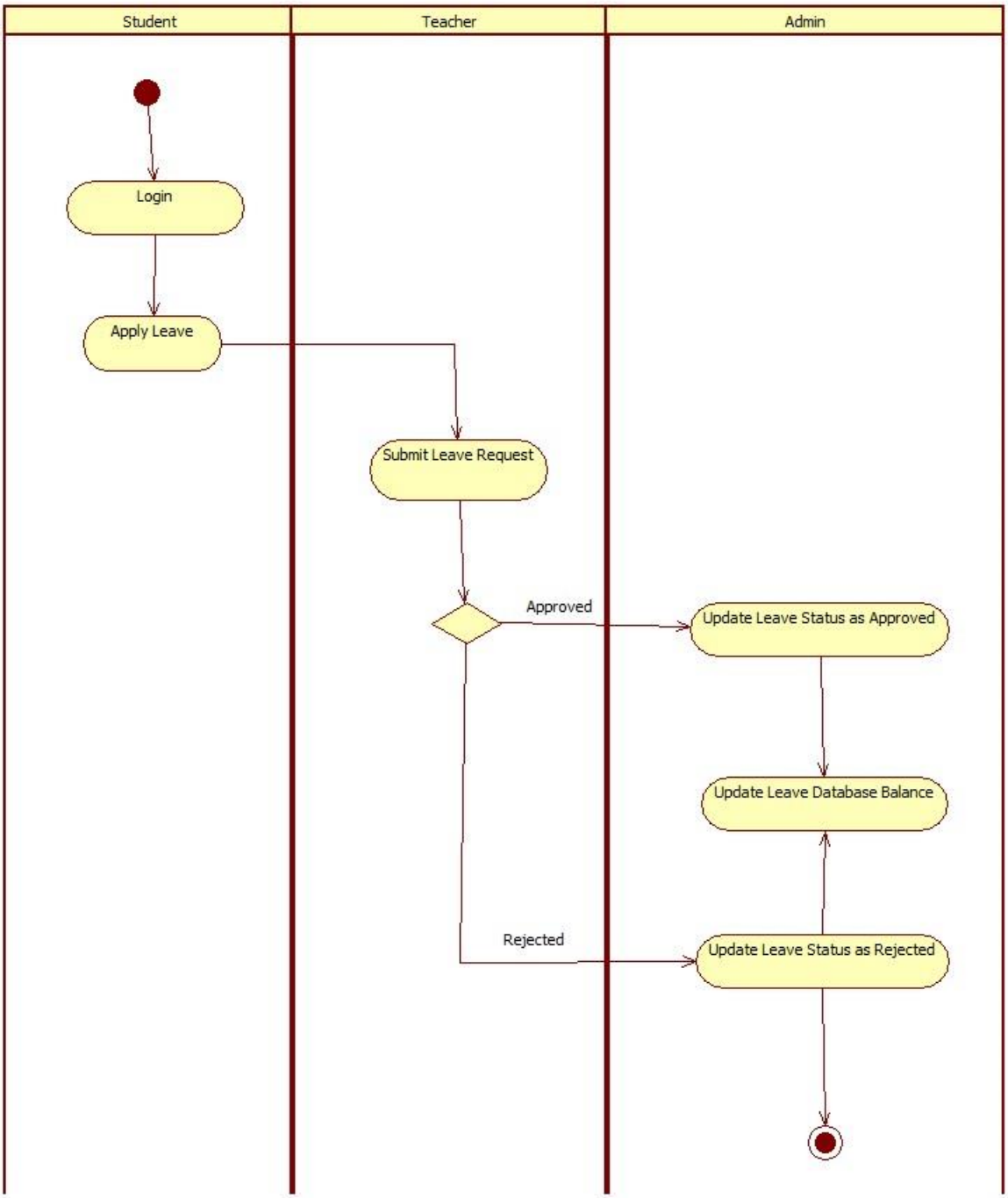


Figure 6: Activity Diagram

Activity Diagram for Change Password:

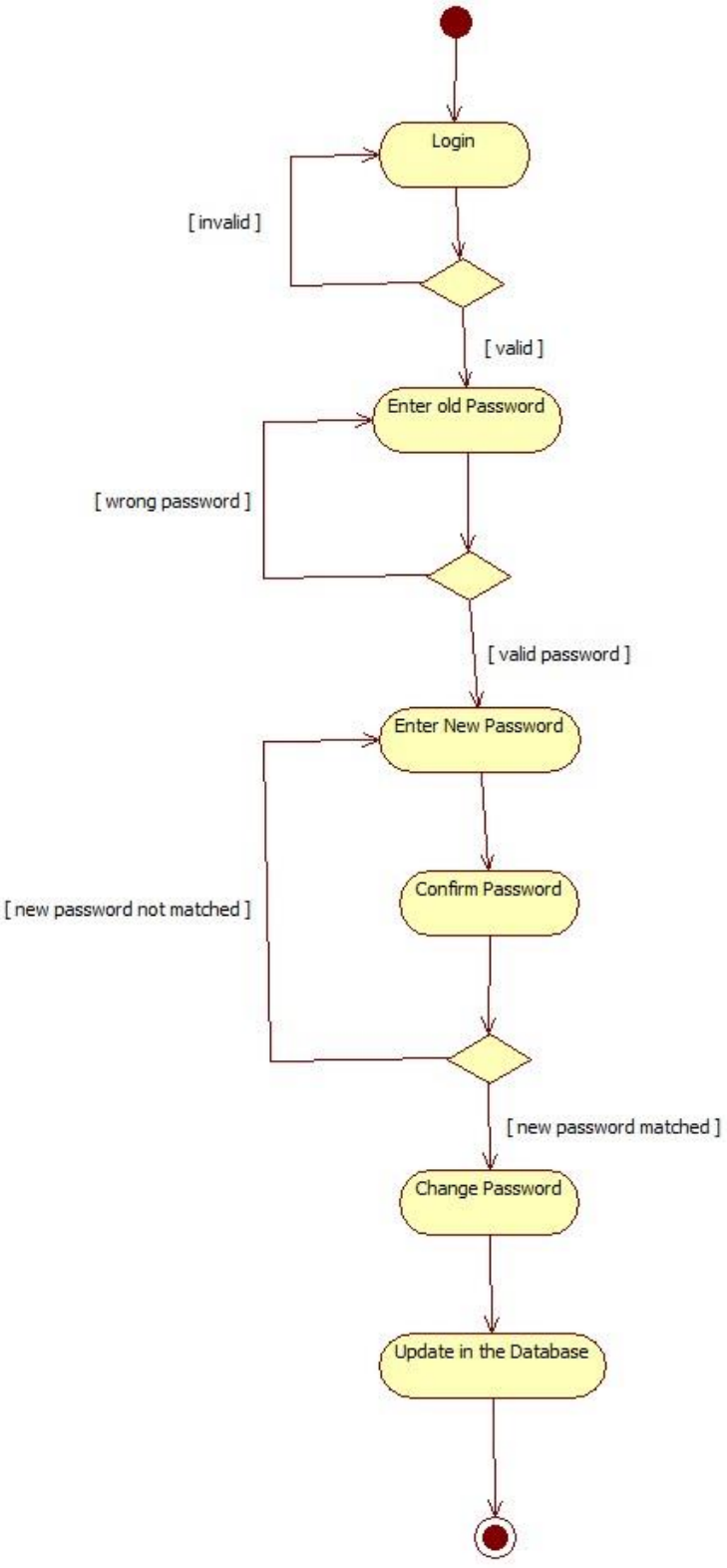


Figure 7: Activity Diagram for Change Password

Activity Diagram for Apply Leave

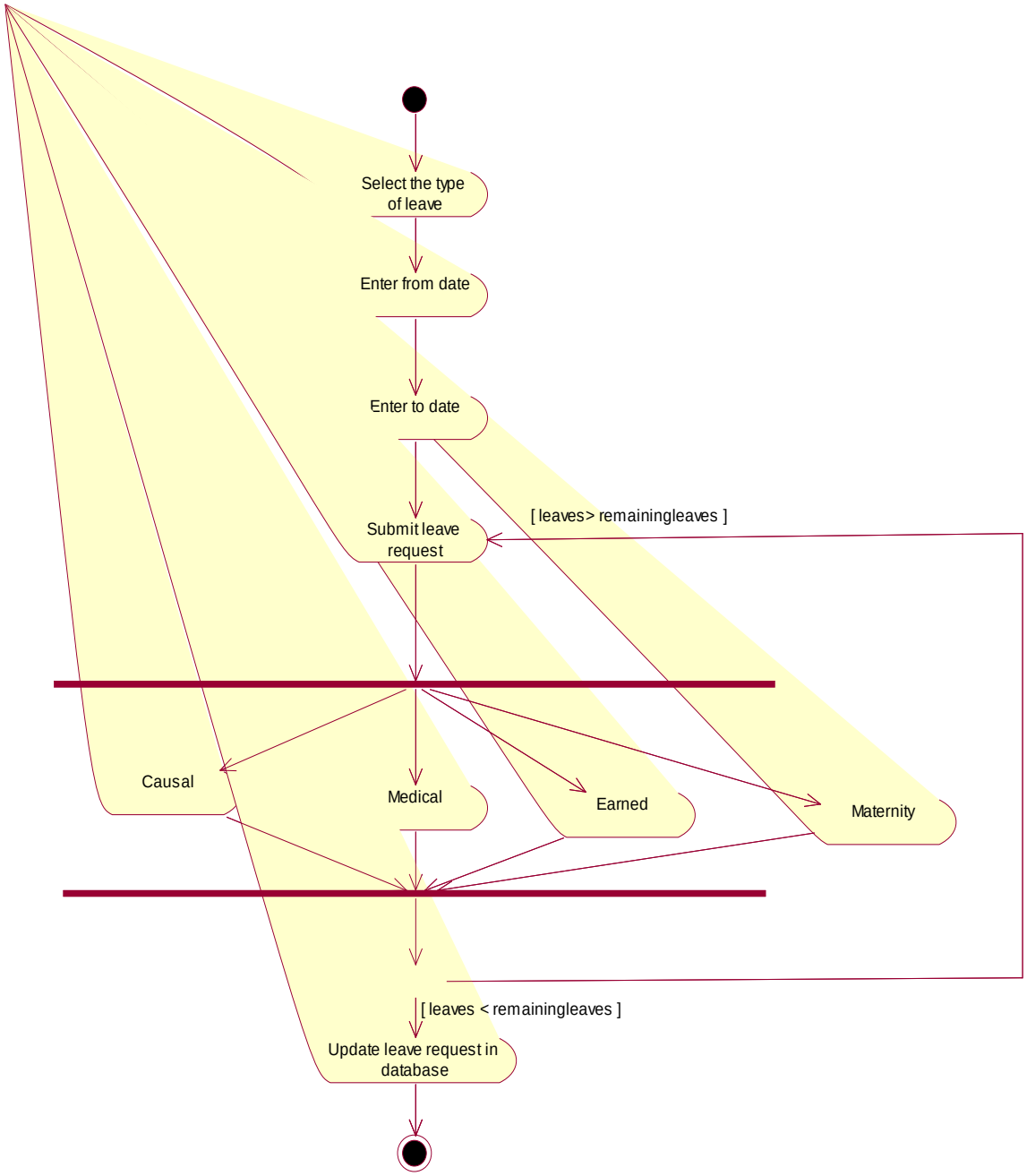


Figure 8:Activity Diagram for Apply Leave

2.3 Logical View

2.3.1 Identification of Analysis Classes

For identification to analysis classes we have three approaches, they area:

1. The noun phrase approach
2. The common class patterns approach
3. The use-case driven approach

In our application we used Use case driven approach for identifying analysis classes

1. Noun Phrase Approach:

In this method, analysts read through the requirements or use cases looking for noun phrases. Nouns in the textual description are considered to be classes and verbs to be methods of the classes all plurals are changed to singular, the nouns are listed, and the list divided into three categories relevant classes , fuzzy classes (the "fuzzy area," classes we are not sure about), and irrelevant classes.

i. Identifying Tentative classes :

The following are guidelines for selecting classes in an application:

- Look for nouns and noun phrases in the use cases.
- Some classes are implicit or taken from general knowledge.
- Carefully choose and define class names.

ii. Selecting classes from the Relevant and Fuzzy Categories:

The following guide lines help in selecting candidate classes from the relevant and fuzzy categories of classes in the problem domain.

a. Redundant Classes
b. Adjectives classes
c. Attribute classes
d. Irrelevant classes

2. Common Class Pattern Approach:

The second method for identifying classes is using common class patterns, which is based on a knowledge base of the common classes. The following pattern s are used for finding the candidate class and object.

- Concept class
- Event class
- Organization class
- People class (also known as person, roles, and roles played class)
- Places class

3. Use-Case Driven Approach:

One of the first steps in creating a class diagram is to derive from a use case, via a collaboration, those classes that participate in realizing the use case. Through further analysis, a class diagram are then usually assembled into a larger analysis class be drawn at any scale that is appropriate, from a single use-case instance to a large, complex system.

2.3.2 Identification of Responsibility of Classes

Class Responsibility Collaboration Cards (CRC Cards)

At the starting, for the identification of classes we need to concentrate completely on uses cases. A further examination of the use cases also helps in identifying operations and the messages that classes need to exchange. However, it is easy to think first in terms of the overall responsibilities of a class rather than its individual operations.

A responsibility is a high level description of something a class can do. It reflects the knowledge or information that is available to that class, either stored within its own attributes or requested via collaboration with other classes, and also the services that it can offer to other objects. A responsibility may correspond to one or more operations. It is difficult t determine the appropriate responsibilities for each class as there may be many alternatives that all appear to be equally justified.

Class Responsibility Collaboration (CRC) cards provide an effective technique for exploring the possible ways of allocating responsibilities to classes and the collaborations that are necessary to fulfill the responsibilities.

CRC cards can be used at several different stages of a project for different purposes.

1. They can be used early in a project to help the production of an initial class diagram.
2. To develop a shared understanding of user requirements among the members of the team.
3. CRCs are helpful in modeling object interaction.

The format of a typical CRC card is shown below:

Class Name:	
Responsibilities	Collaborations
Responsibilities of a class are listed in this section	Collaborations with other classes are listed here, together with a brief description of the purpose of the collaboration

CRC cards are an aid to a group role-playing activity. Index cards are used in preference to pieces of paper due to their robustness and to the limitations that their size imposes on the number of responsibilities and collaborations that can be effectively allocated to each class. A class name is entered at the top of each card and responsibilities and collaborations are listed underneath they become apparent.

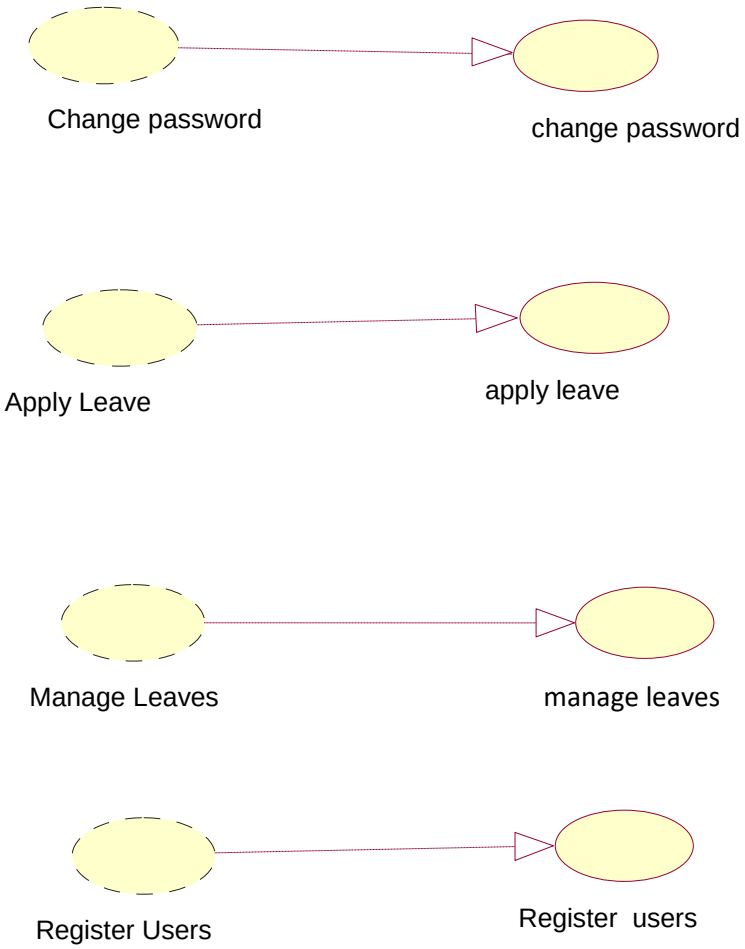
From a UML perspective, use of CRC cards is in analyzing the object interaction that triggered by a particular use case scenario. The process of using CRC cards is usually structured as follows.

1. Conduct a session to identify which objects are involved in the use case.
2. Allocate each object to a team member who will play the role of that object.
3. Act out the use case: This involves a series of negotiations among the objects to explore how responsibility can be allocated and to identify how the objects can collaborate with each other.
4. Identify and record any missing or redundant objects.

2.3.3 Use Case Realizations

A use case realization is a graphic sequence of events, also referred as an instance of a use case. These realizations are represented using either a sequence or collaboration diagrams.

Use case Realization Diagrams:



2.3.4 Sequence Diagram

A sequence diagram is a graphical view of a scenario that shows object interaction in a time based sequence-what happens first, what happens next….

Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

A sequence diagram has two dimensions: the vertical dimension represents time; the horizontal dimension represents different objects. The vertical line is called the object's lifeline. The lifeline represents the object's existence during the interaction.

Steps:

1. An object is shown as a box at the top of a dashed vertical line. Object names can be specific (e.g., Algebra 101, Section 1) or they can be general (e.g., a course offering). Often, an anonymous object (class name may be used to represent any object in the class.)

2. Each message is represented by an Arrow between the lifelines of two objects. The order in which these messages occur is shown top to bottom on the page. Each message is labeled with the message name.

The sequence diagram is very simple and has immediate visual appeal—this is its great strength. A sequence diagram is an alternative way to understand the overall flow of control of a program. Instead of looking at the code and trying to find out the overall sequence of behaviour.

The following tools located on the sequence diagram toolbox which enable to model sequence diagrams:

Object: An object has state, behavior, and identity. The structure and behavior of similar 1objects are defined in their common class. Each object in a diagram indicates some instance of a class. An object that is not named is referred to as a class instance.

Message Icons: A message icon represents the communication between objects indicating that an action will follow. The message icon is a horizontal, solid arrow connecting two lifelines together.

Focus of Control: Focus of Control (FOC) is an advanced notational technique that enhances sequence diagrams. It shows the period of time during which an object is performing an action either directly or through an underlying procedure.

Message to self: A message to self is a tool that sends a message from one object back to the same object. It does not involve other objects because the message returns to the same object. The sender of a message is the same as the receiver.

Note: A note captures the assumptions and decisions applied during analysis and design. Notes may contain any information, including plan text, fragments of code, or references to other documents.

Note Anchor: A note anchor connects a note to the element that it affects.

Sequence Diagram for Login:

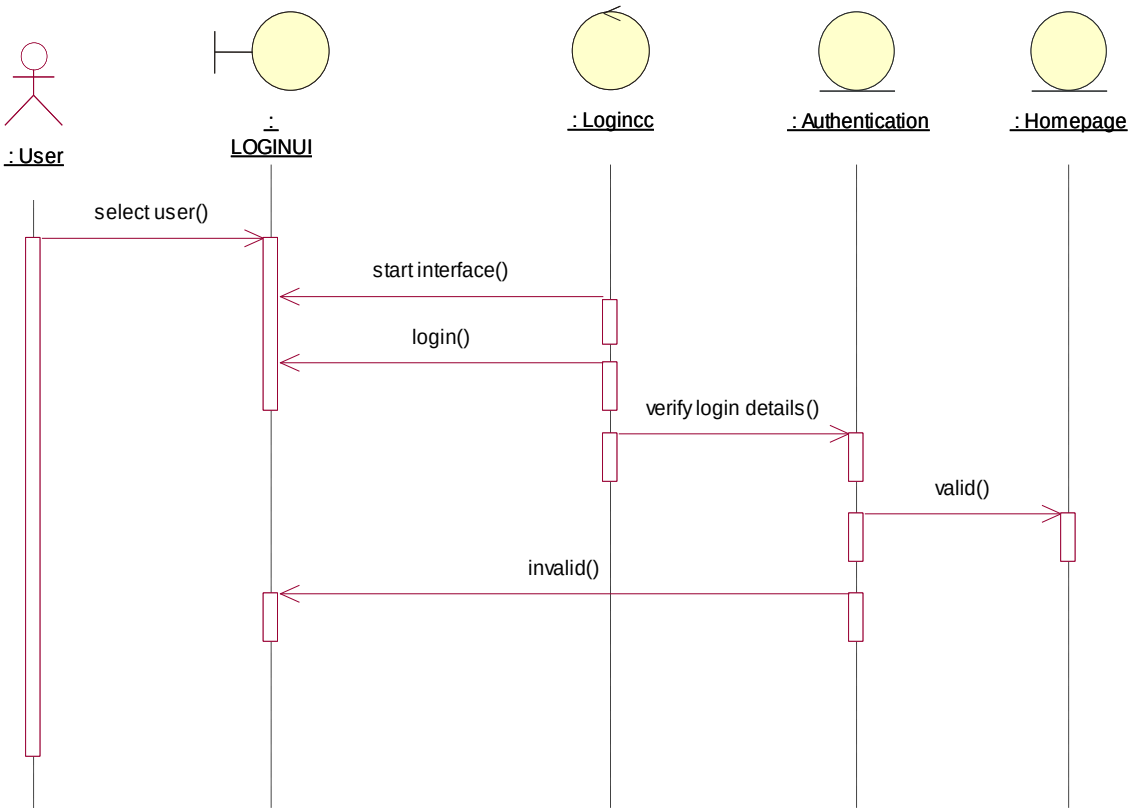


Figure 9: Sequence Diagram for Login

Sequence Diagram for Apply Leave

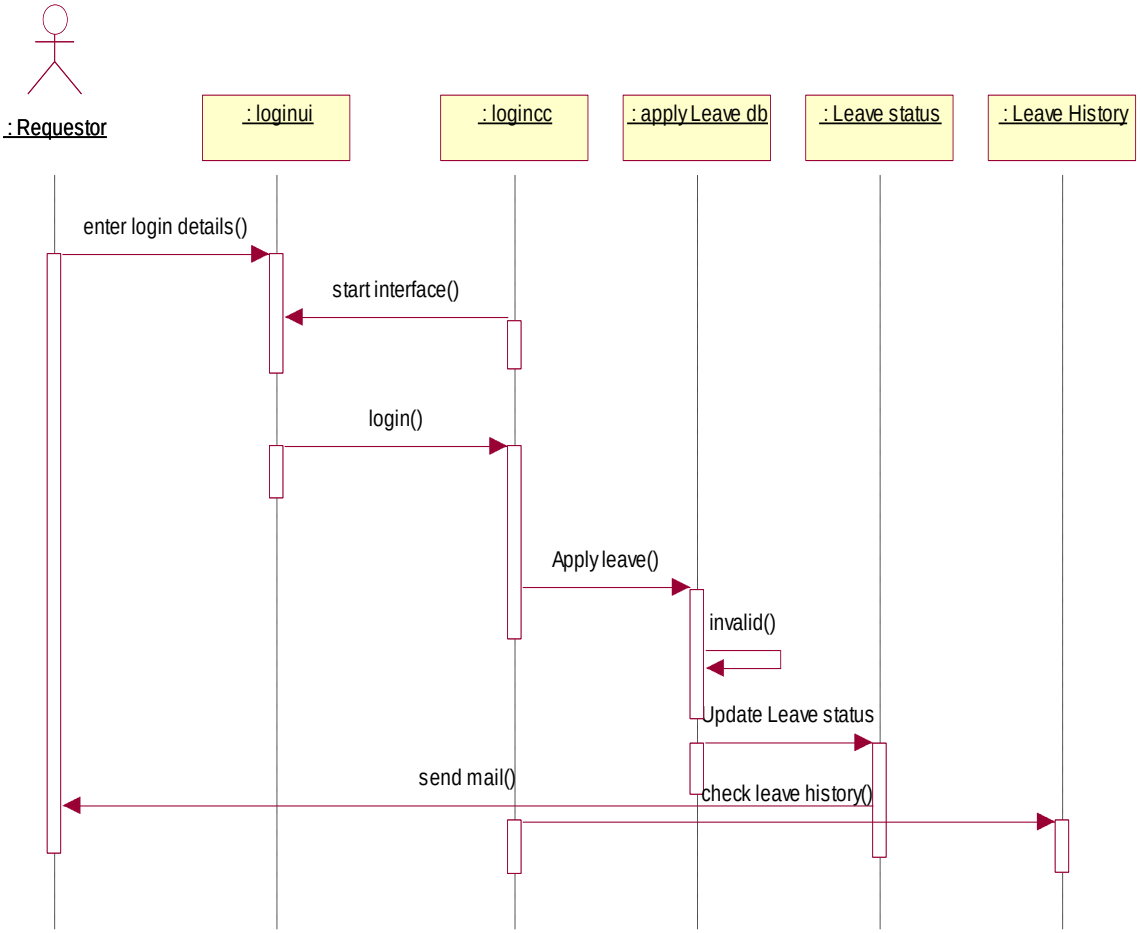


Figure 10: Sequence Diagram for Apply Leave

Sequence Diagram for Change Password

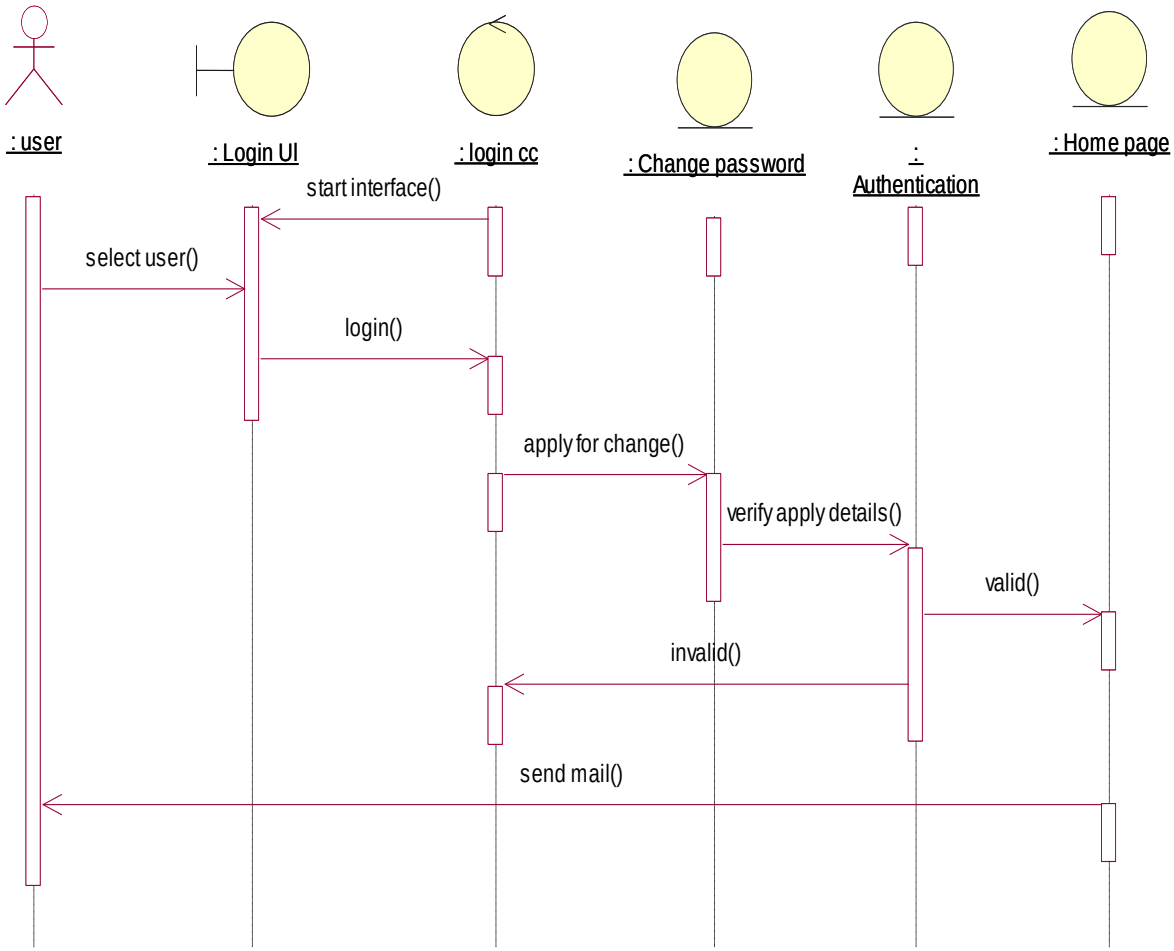


Figure 11: Sequence Diagram for Change Password

2.3.5 Collaboration Diagrams

Collaborations can also be represented in various ways that reveal their internal details. The *collaboration diagram* is probably the most useful among all UML diagrams for use case realization.

Collaboration diagrams and sequence diagrams are called interaction diagrams. A collaboration diagram shows that the order of messages that implement an operation or a transaction. Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object like entities in the current model.

A Collaboration Diagram is an alternate way to show a scenario. This type of diagram shows object interactions organized around the objects and their links to each other. A collaboration diagram contains

- Objects drawn as rectangles
- Links between objects shown as lines connecting the linked objects
- Messages shown as text and an arrow that points from the client to the supplier

The disadvantages of interaction diagrams are that they are great only for representing a single sequential process; they begin to break down when you want to represent conditional looping behavior.

Collaboration Diagram for Login:

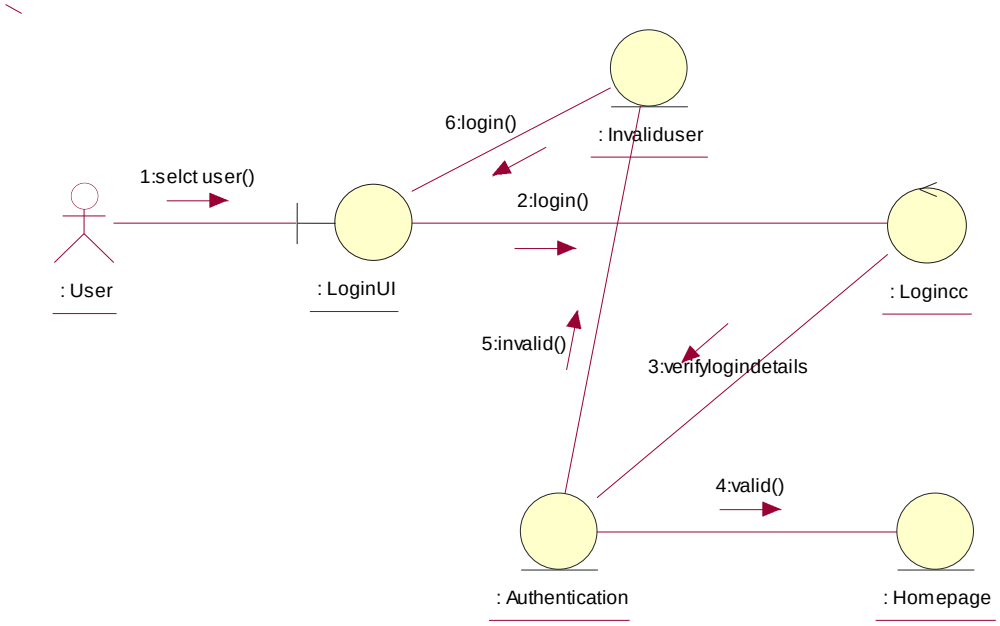


Figure 12: Collaboration Diagram for Login

Collaboration Diagram for Apply Leave:

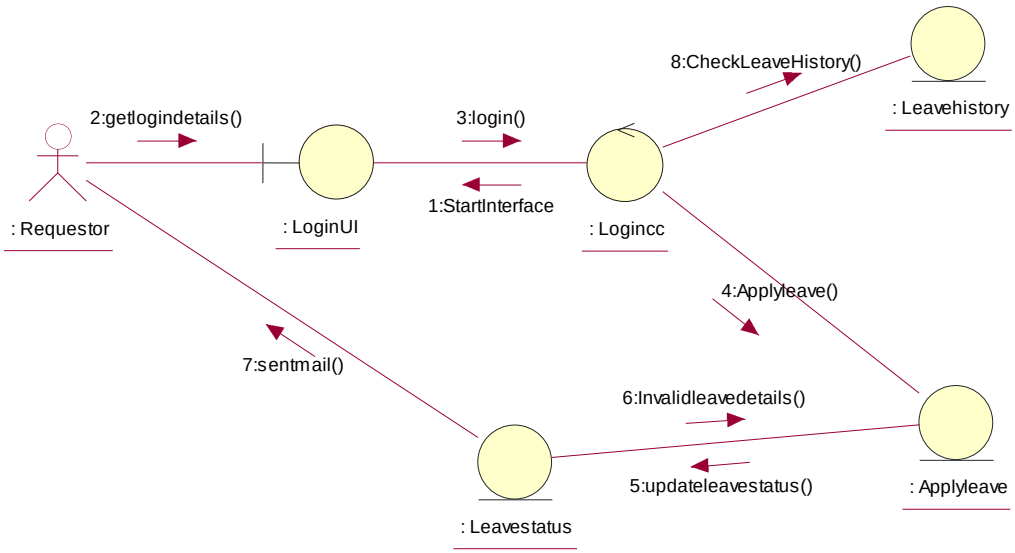


Figure 13: Collaboration Diagram for Apply Leave

Collaboration Diagram for Leave Management System

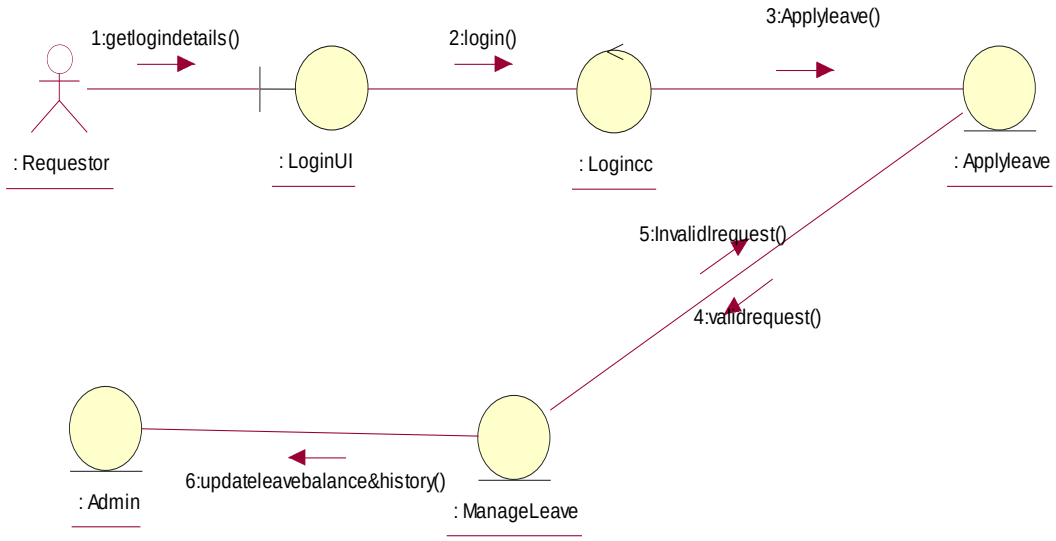


Figure 14: Collaboration Diagram for Leave Management System

Difference between sequence and collaboration diagrams:

- Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction.
- Sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other.
- A sequence diagram is a graphical view of a scenario that show s object interaction in a time based sequence.
- A collaboration diagram shows object interactions organized around the objects and their links to each other.

2.3.6 Identification of Attributes and Methods of Class

Attributes:

Attributes are part of the essential description of a class. They belong to the class, unlike objects, which instantiate the class. Attributes are the common structure of what a member of the class can 'know'. Each object will have its own, possibly unique, value for each attribute.

Guidelines for identifying attributes of classes are as follows:

- Attributes usually correspond to nouns followed by prepositional phrases. Attributes also may correspond to adjectives or adverbs.
- Keep the class simple; state only enough attributes to define the object state.
- Attributes are less likely to be fully described in the problem statement.
- Omit derived attributes.
- Do not carry discovery attributes to excess.

Some questions are there which help in identifying the responsibilities of classes and deciding what data elements to keep track of:

- What information about an object should we keep track of?
- What services must a class provide?

Answering the first question helps us to identify the attributes of a class. Answering the second question helps us to identify class methods.

The attributes identified in our system are

- Attributes for Login: username, password.
- Attributes for User: Name, Username, Password, Department, Gender, Joining Date, No of leaves taken.
- Attributes for Administrator: Name of employee, Leave details, Leave id.
- Attributes for Apply Leave: Leave Type, From Date, To Date, Leave id, Department, Name.
- Attributes for Manage Leave: Leave id, Name, Leave Type, From Date, To Date, Leave id, Department, Name.
- Attributes for Change Password: Old password, New Password.
- Attributes for Leave Status and history: Username.
- Attributes for Leave Balance: Username, Leave Status.

The responsibilities identified in our system are:

- Methods for Login: login ().
- Methods for Apply Leave: apply leave ().
- Methods for Manage Leave: manage leaves (), approve (), reject ().
- Methods for Leave status and history: checkLeaveHistory (), LeaveStatus ().
- Methods for Change Password: change ().
- Methods for Leave Balance:checkLeaveBalance()

2.3.7 Identification of Relationships among Classes

Need for Relationships among the class: All systems are made up of many classes and objects. System behavior is achieved through the collaborations of the objects in the system.

Two types of relationships in CLASS diagram are:

1. Associations Relationship
2. Aggregations Relationship

1. Association Relationships:

An Association is a bidirectional semantic connection between classes. It is not a data flow as defined in structured analysis and design data may flow in either direction across the association. An association between classes means that there is a link between objects in the associated classes.

For example, an association between the E-Pay class and the Administrator class means that objects in the E-Pay class are connected to objects in the Administrator class. Association Relationship without Multiplicity

2. Aggregation Relationships:

An Aggregation Relationship is a specialized form of association in which a whole is related to its part(s).

Aggregation is known as a "part-of" or containment relationship. The UML notation for an aggregation relationship is an association with a diamond next to the class denoting the aggregate (whole).

Naming Relationships:

An association may be named. Usually the name is an active verb or verb phrase that communicates the meaning of the relationship. Since the verb phrase typically implies a reading direction, it is desirable to name the association so it reads correctly from left to right or top to bottom. The words may have to be changed to read the association in the other direction. It is important to note that the name of the association is optional.

Role Names:

The end of an association where it connects to a class is called an association role. Role names can be used instead of association names.

A role name is a noun that denotes how one class associates with another. The role name is placed on the association near the class that it modifies, and may be placed on one or both ends of an association line.

- It is not necessary to have both a role name and an association name.
- Associations are named or role names are used only when the names are needed for clarity.

Multiplicity Indicator:

Although multiplicity is specified for classes, it defines the number of objects that participate in a relationship. Multiplicity defines the number of objects that are linked to one another. There are two multiplicity indicators for each association or aggregation one at each end of the line.

Some common multiplicity indicators are

1	Exactly one
0.. *	Zero or more
1.. *	One or more
0.. 1	Zero or one
5 .. 8	Specific range (5, 6, 7, or 8)
4.. 7,9	Combination (4, 5, 6, 7, or 9)

Reflexive Relationship:

Multiple objects belonging to the same class may have to communicate with one another. This is shown on the class diagram as a reflexive association or aggregation. Role names rather than association names typically are used for reflexive relationships.

2.3.8 UML Class Diagram

Class diagrams contain icons representing classes, interfaces, and their relationships. You can create one or more class diagrams to represent the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model. You can also create one or more class diagrams to represent classes contained by each package in your model; such class diagrams are themselves contained by the package enclosing the classes they represent; the icons representing logical packages and classes in class diagrams.

- Class diagrams are created to provide a picture or view of some or all of the classes in the model.
- The main class diagram in the logical view of the model is typically a picture of the packages in the' system. Each package also has its own main class diagram, which typically displays the "public" classes of the package.

A class diagram is a picture for describing generic descriptions of possible systems. Class diagrams and collaboration diagrams are alternate representations of object models.

A Class is a description of a group of objects with common properties (attributes) common behavior (operations), common relationships to other objects, and common semantics. Thus, a class is a template to create objects. Each object is an instance of some class and objects cannot be instances of more than one class.

In the UML, classes are represented as compartmentalized rectangles.

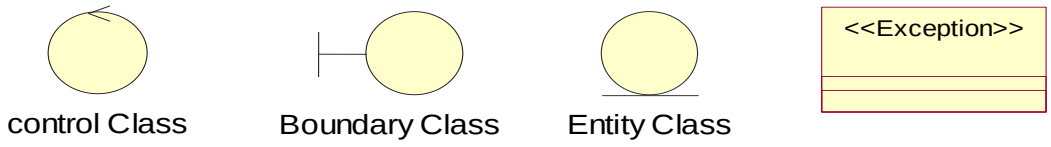
- The top compartment contains the name of the class.
- The middle compartment contains the structure of the class (attributes).

- The bottom compartment contains the behavior of the class (operations).

Stereotypes and Classes:

As like stereotypes for relationships in use case diagrams. Classes can also have stereotypes. Here a stereotype provides the capability to create a new kind of modeling element. Here, we can create new kinds of classes. Some common stereotypes for a class are entity Class, boundary Class, control class, and exception.

Notations of these stereotypes:



Entity Classes

- An **entity class** models information and associated behavior that is generally long lived.
- This type of class may reflect a real-world entity or it may be needed to perform tasks internal to the system.
- They are typically independent of their surroundings; that is, they are not sensitive to how the surroundings communicate with the system.

Control Classes

- Control classes model sequencing behavior specific to one or more use cases.
- Control classes coordinate the events needed to realize the behavior specified in the use case.

- Control classes typically are application-dependent classes.

In the early stages of the Elaboration Phase, a control class is added for each actor/use case pair. The control class is responsible for the flow of events in the use case.

In the early stages of the Elaboration Phase, a control class is added for each actor/use case pair. The control class is responsible for the flow of events in the use cases.

Boundary Classes:

Boundary classes handle the communication between the system surroundings and the inside of the system. They can provide the interface to a user or another system (i.e., the interface to an actor). They constitute the surroundings dependent part of the system.

Boundary classes are used to model the system interfaces

Boundary classes are also added to facilitate communication with other systems. During design phase, these classes are refined to take into consideration the chosen communication protocols.

Documenting classes:

As classes are created, they should also be documented. The documentation should state the purpose of the class and not the structure of the class.

For example, a Student class could be documented as follows: A student is someone currently registered to take classes at the university and this information will be provided for billing purpose.

Packages:

If a system contained only a few classes, you could manage them easily. Most systems are composed of many classes, and thus you need a mechanism to group them together for ease of use, maintainability, and reusability.

A package in the logical view of the model is a collection of related packages and/or classes. By grouping classes into packages, we can look at the "higher" level view of the model (i.e., the packages).

Class Diagram for Leave Management System

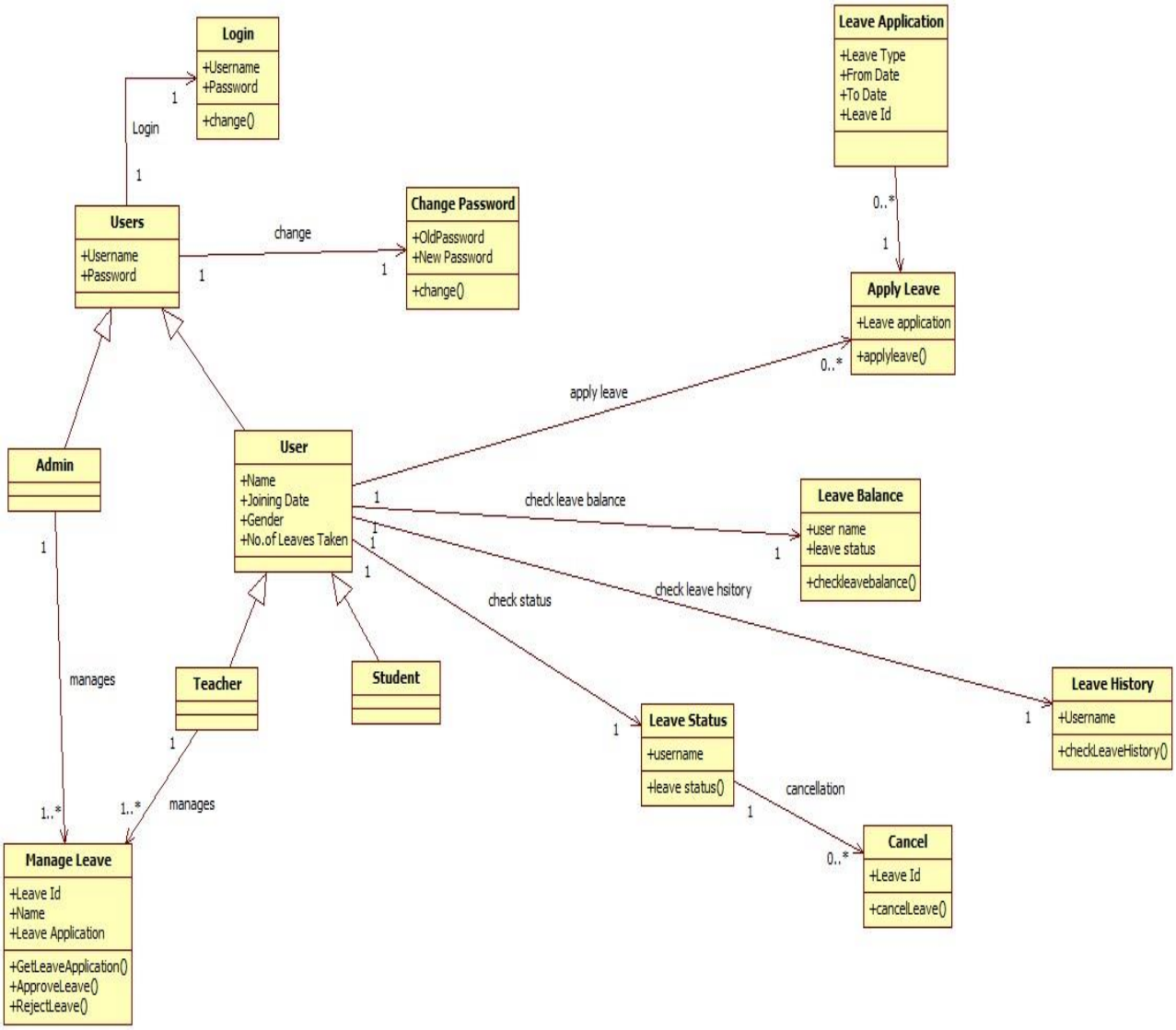


Figure 15: Class Diagram for Leave Management System

Class Diagram for Manage Leave

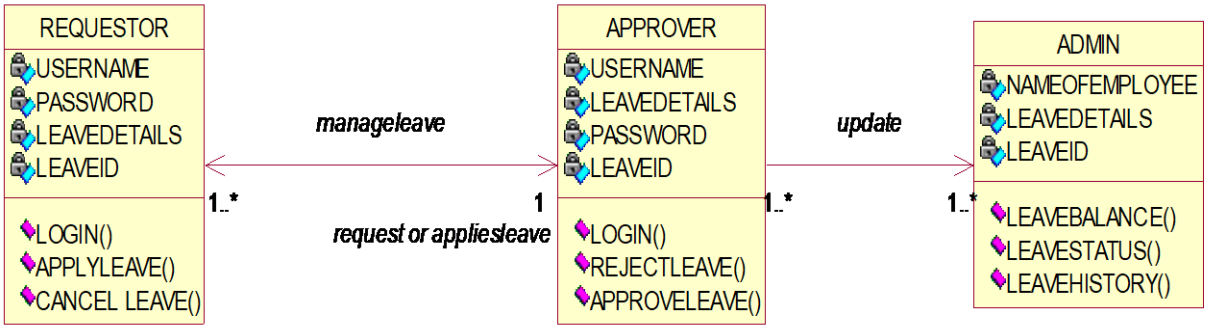


Figure 16: Class Diagram for Manage Leave

Class Diagram for Apply Leave

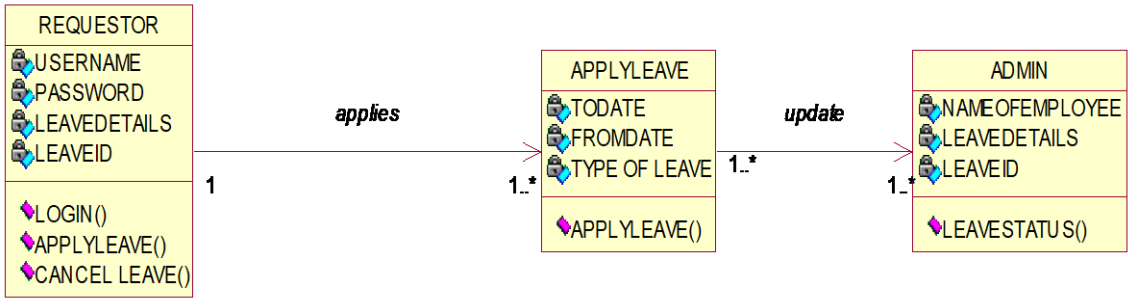


Figure 17: Class Diagram for Apply Leave

2.3.9 UML State Chart Diagram

Use cases and scenarios provide a way to describe system behavior; in the form of interaction between objects in the system. Sometime it is necessary to consider inside behavior of an object.

A state chart diagram shows the states of a single objects, the events or messages that cause a transition from one state to another and the actions that result from a state change. As I activity diagram, state chart diagram also contains special symbols for start state and stop state.

State chart diagram cannot be created for every class in the system, it is only for those lass objects with significant behavior.

State:

A state represents a condition or situation during the life of an object during which it satisfies some condition, performs some action or waits for some event.

UML notation for STATE is

To identify the states for an object its better to concentrate on sequence diagram. In our application the object for Account may have in the following states, initialization, open and closed state. These states are obtained from the attribute and links defined for the object. Each state also contains a compartment for actions.

Action:

Actions on states can occur at one of four times:

- on entry
- on exit
- do
- on event.

- **on entry :**What type of action that object has to perform after entering into the state.
- **on exit :** What type of action that object has to perform after exiting from the state.
- **Do :** The task to be performed when object is in this state, and must to continue until it leaves the state.
- **on event :** An on event action is similar to a state transition label with the following
- syntax: event(args)[condition] : the Action

State transition:

A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two states, two activities, or between an activity and a state.

We can show one or more state transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event, unless there are conditions on the event.

Provide a label for each state transition with the name of at least one event that causes the state transition. You do not have to use unique

labels for state transitions because the same event can cause a transition to many different states or activities.

Transitions are labeled with the following syntax:

event (arguments) [condition] / action ^ target.sendEvent (arguments)

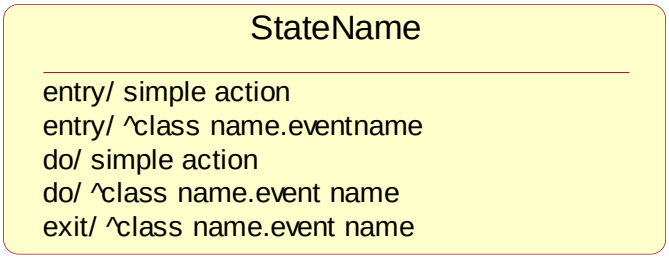
Only one event is allowed per transition, and one action per event.

State Details:

Actions that accompany all state transitions into a state may be placed as an entry action within the state. Likewise that accompany all state transitions out of a state may be placed as exit actions within the state. Behavior that occurs within the state is called an activity.

An activity starts when the state is entered and either completes or is interrupted by an outgoing state transition. The behavior may be a simple action or it may be an event sent to another object.

UML notation for State Details;



Purpose of state chart diagrams:

- We use state chart diagram when working on real-time process control applications or systems that involve concurrent processing. It will also be used when showing the behavior of a class over several use cases.

- State chart diagrams are used to model dynamic view of a system.
- State chart diagrams are used to emphasizing the potential states of the objects and the transitions among those states.
- State chart diagrams are used to modeling the lifetime of an object.
- State chart diagrams are used to model the behavior of an interface. Although an interface may not have any direct instances, a class that realizes such an interface may. Those classes conform to behavior specified by the state machine of this interface.
- State chart diagrams are used to focus on the changing state of a system driven by events.
- This diagram is also for constructing executable systems through forward and reverse engineering.
- To model reactive objects, especially instances of a class, use cases, and the system as a whole.

Elements of state chart diagrams:

- State
- Event
- Transition
- Action state
- Sequential sub state
- Concurrent sub state
- Initial state
- Final state
- History state
- Vertical Synchronization
- Horizontal Synchronization
- Guard conditions
- Forks and joins

State Chart Diagram for Apply Leave:

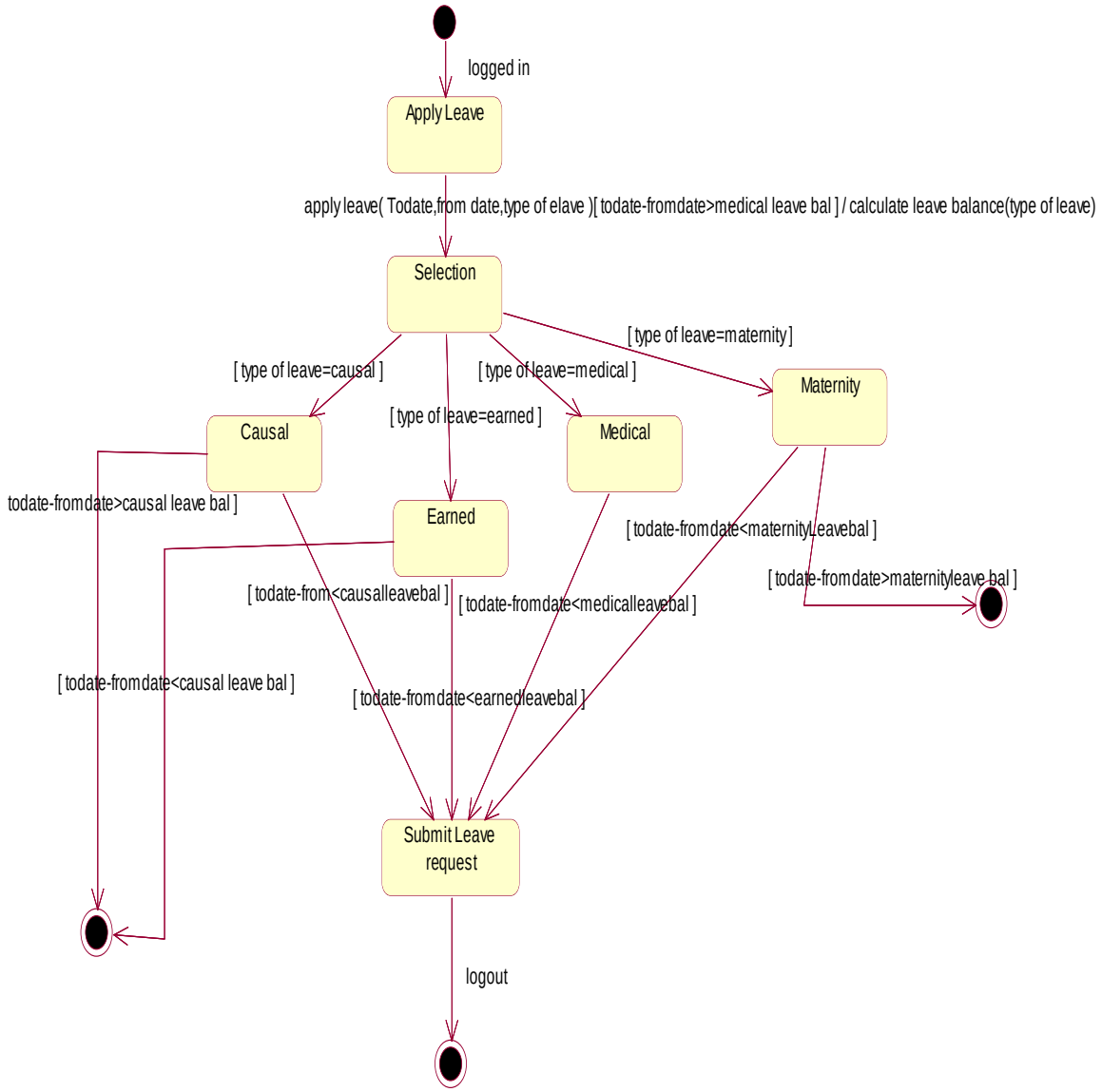


Figure 18: State Chart Diagram for Apply Leave

State Chart Diagram for Manage Leave

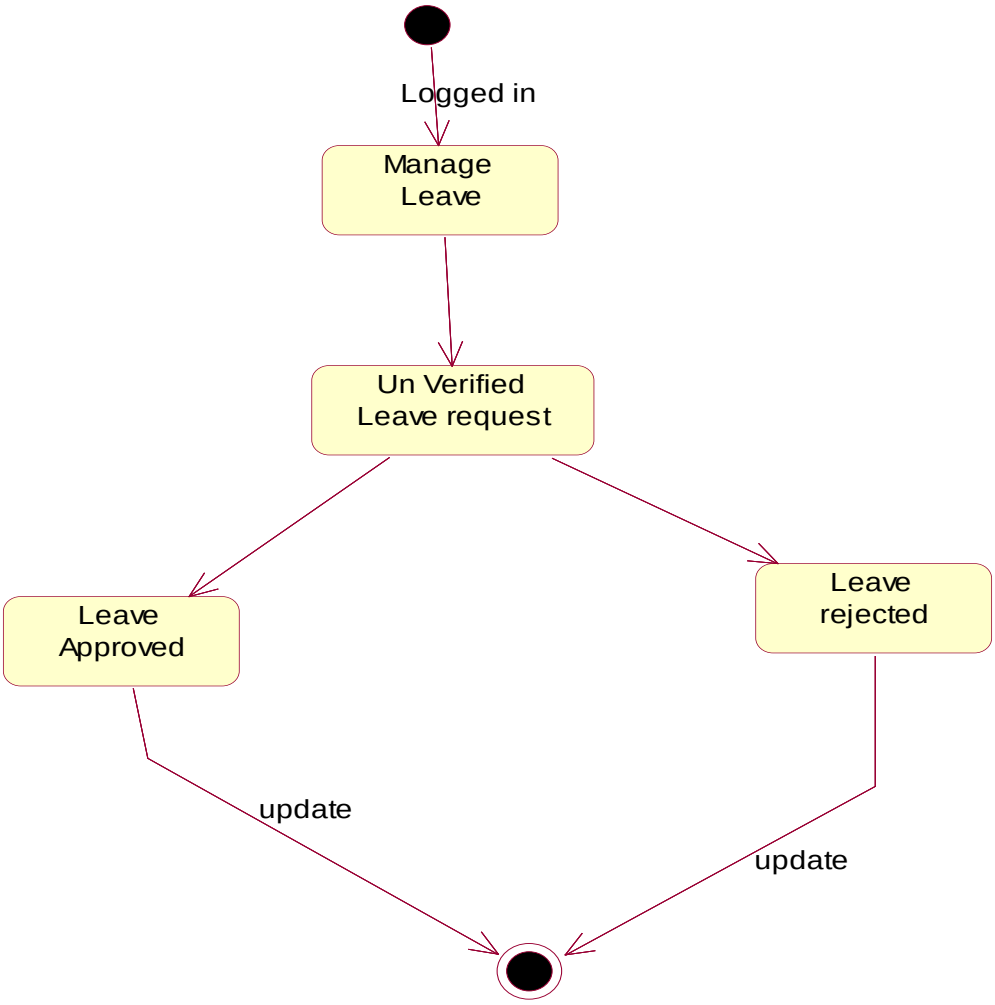


Figure 19: State Chart Diagram for Manage Leave

State Chart Diagram for Change Password

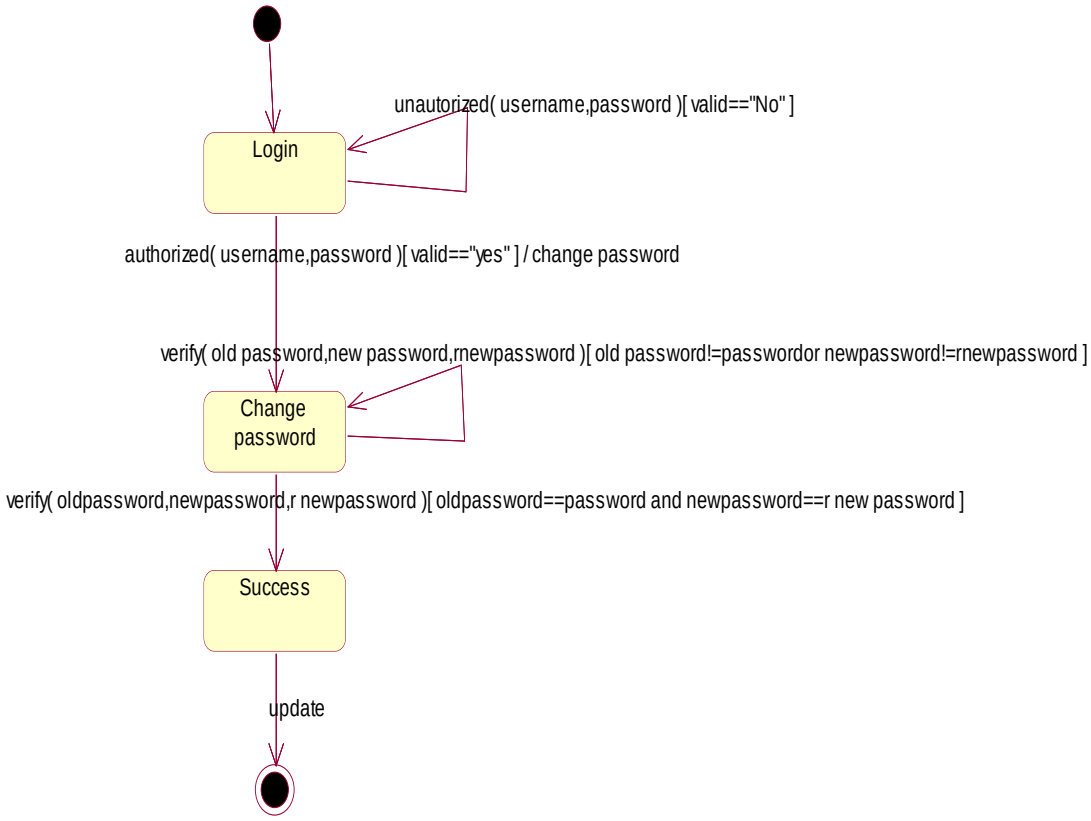


Figure 20: State Chart Diagram for Change Password

3

SYSTEM DESIGN

3.1 Refining Attributes methods and Relationships

Attributes:

During analysis Stage we need to consider in detail the data types of the attributes also. Common primitive data types include Boolean (true or false), Character (any alphanumeric or special character), Integer (whole numbers) and Floating-Point (decimal numbers). In most object-oriented languages more complex data types, such as Money, String, Date, or Name can be constructed from the primitive data types or may be available in standard libraries. An attribute's data type is declared in UML using the following syntax:

name ':' type-expression '=' initial-value '{'property-string'}'

The name is the attribute name, the type-expression is its data type, the initial value is the value the attribute is set to when the object is first created and the property-string describes a property of the attribute, such as constant or fixed. The characters in single quotes are literals.

Attribute declarations can also include arrays also. For example, an Employee class might include an attribute to hold a list of qualifications that would be declared using the syntax: Qualification [O ... 10]: String

Operations :

Each operation also has to be specified in terms of the parameters that it passes and returns. The syntax used for an operation is:

Operation name' ('parameter-list ') ": "return-type-expression

An operation's *signature* is determined by the operation's name, the number and type of its parameters and the type of the return value if any.

Object visibility:

The concept of encapsulation is one of the fundamental principles of object-orientation. During analysis various assumptions have been made regarding the encapsulation boundary for an object and the way that objects interact with each other.

For example, it is assumed that the attributes of an object cannot be accessed directly by other objects but only via 'get' and 'set' operations (primary operations) that are assumed to be available for each attribute. Moving to design involves making decisions regarding which operations (and possibly attributes) are publicly accessible. In other words we must define the encapsulation boundary.

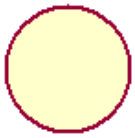
Visibility symbol	Visibility	Meaning
+	Public	The feature (an operation or an attribute) is directly accessible by an instance of any class.
-	Private	The feature may only be used by an instance of the class that includes it.
#	Protected	The feature may be used either by instances of the class that includes it or of a subclass or descendant of that class.
~	Package	The feature is directly accessible only by instances of a class in the same package.

Table 4: Object Visibility

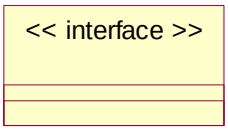
Interfaces :

Generally a class may present more than one external interface to other classes or the same interface may be required from more than one class. An interface in UML is a group of externally visible (i.e. public) operations. The interface contains no internal structure, it has no attributes, no associations and the implementation of the operations is not defined. Formally, an interface is equivalent to an abstract class that has no attributes, no associations and only abstract operations.

The following figure shows two alternative notations for an interface. The simpler of the two UML interface notations is a circle. This is attached by a solid line to the classes that support the interface.



The alternative notation uses a stereotyped class icon. As an interface only specifies the operations and has no internal structure, the attributes compartment is omitted. This notation lists the operations on the diagram. The *realize* relationship, represented by the dashed line with a triangular arrowhead, indicates that the class supports at least the operations listed in the interface



Refining Attributes:

In the analysis phase, the name of the attribute was sufficient. However, in the design phase, detailed information must be added to the model. There are three basic types of attributes. They are:

1) Single-value attributes.

2) Multiplicity or multi-value attributes.

3) Reference to another object, or instance connection.

Attributes represent the state of an object. The following is the attribute presentation:

Visibility name: type-expression=initial-value

Where visibility is one of the following:

+ public visibility

protected visibility

- private visibility

During analysis, we identified the following attributes for classes:

Refining attributes for Librarian class:

- name
- number

At this stage we need to add more information to these attributes, such as

visibility and implementation type.

+ name : String

+ number: String

Refining attributes for member class:

name

id

After refining

+ id : String

+ name: String

Refining methods:

A class can provide several types of methods.

- Constructor : Method that creates instances of the class.
- Destructor : The method that destroys instances.
- Conversion method: The method that converts a value from one unit of measure to another.
- Copy method : The method that copies the contents of one instance to another Instance.
- Attribute set : The method that sets the values of one or more attributes.
- I/O methods : The methods that provide or receive data to or from a device.
- Domain specific : The method specific to the application.

The operation syntax is:

Visibility name : (parameter-list) : return-type-expression

3.2 Implementation Diagrams

3.2.1 Component Diagrams

- Component Diagrams show the dependencies between software components in the system. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time or at runtime.
- In a large project there will be many files that make up the system. These files will have dependencies on one another. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time, at link-time or at run-time. There are also dependencies between source code files and the executable files or byte code files that are derived from them by compilation. Component diagrams are one of the two types of implementation diagram in UML. Component diagrams show these dependencies between software components in the system. Stereotypes can be used to show dependencies that are specific to particular languages also.
- A component diagram shows the allocation of classes and objects to components in the physical design of a system. A component diagram may represent all or part of the component architecture of a system along with dependency relationships.
- The dependency relationship indicates that one entity in a component diagram uses the services or facilities of another.
    - Dependencies in the component diagram represent compilation dependencies.
    - The dependency relationship may also be used to show calling dependencies among components, using dependency arrows from components to interfaces on other components.

Types of components:

- Deployment component
- Work product component
- Executable component

Uses:

- To model source code.
- To model executable releases.
- To model physical databases.
- To model adaptable systems.

Component Diagram for Leave Management System

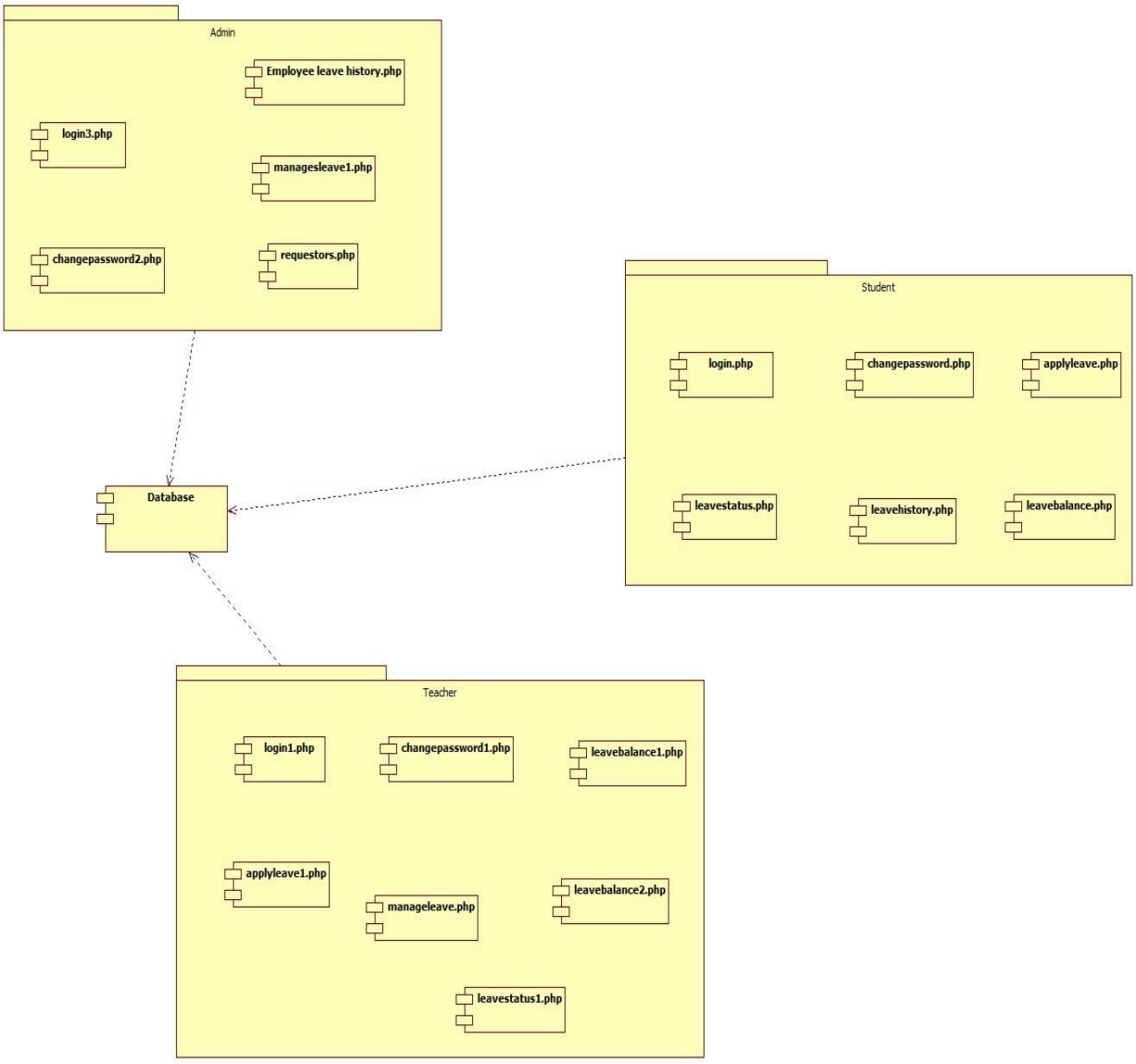


Figure 21: Component Diagram for Leave Management System

Component Diagram for Manage Leaves

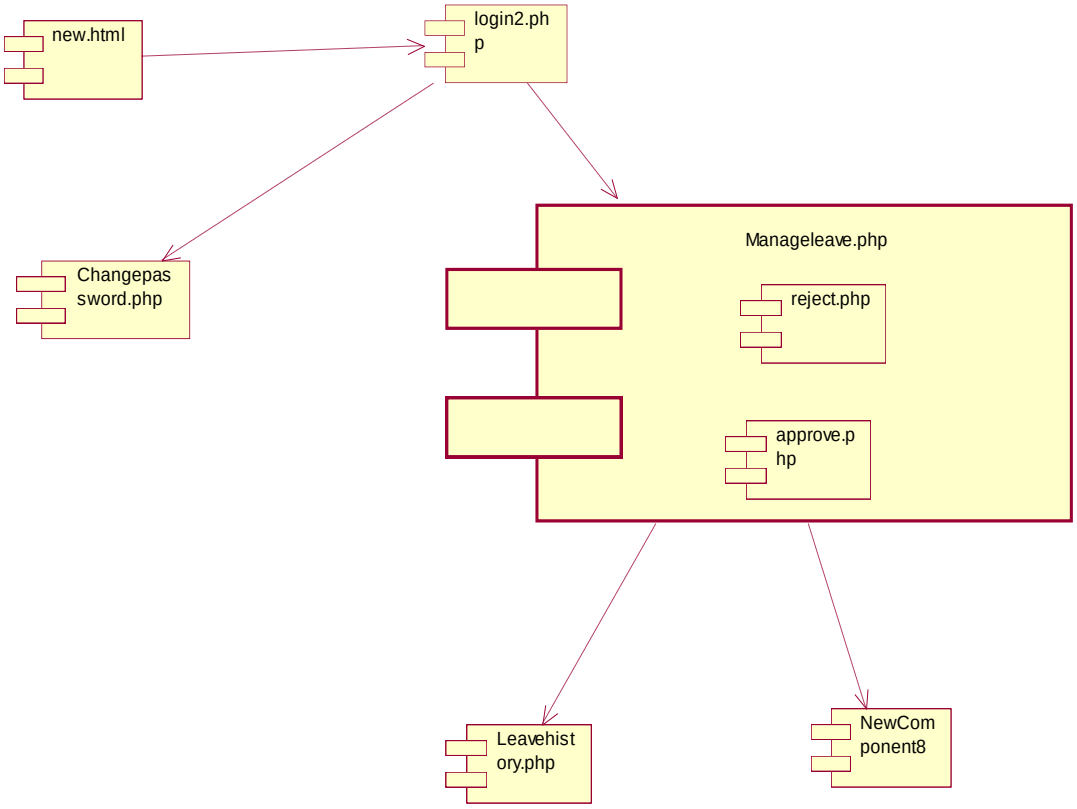


Figure 22: Component Diagram for Manage Leaves

3.2.2 Deployment Diagram

- The second type of implementation diagram provided by UML is the deployment diagram. Deployment diagrams are used to show the configuration of run-time processing elements and the software components and processes that are located on them.

- Deployment diagrams are made up of nodes and communication associations. Nodes are typically used to show computers and the communication associations show the network and protocols that are used to communicate between nodes. Nodes can be used to show other processing resources such as people or mechanical resources.

- Nodes are drawn as 3D views of cubes or rectangular prisms, and the following figure shows a simplest deployment diagram where the nodes connected by communication associations .

Uses:

- To model embedded system.
- To model client-server system.
- To model distributed systems.

Deployment Diagram for Leave Management System

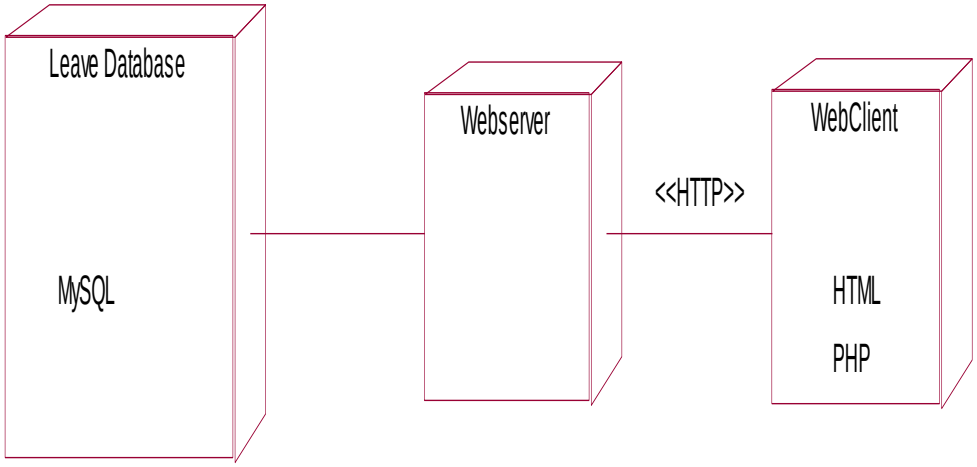


Figure 23: Deployment Diagram for Leave Management System

3.3 Architecture of the System

Leave Management System for School is designed in three tires separately using three layers called presentation layer, business logic layer and data link layer. The project was developed using 3-tier Architecture.

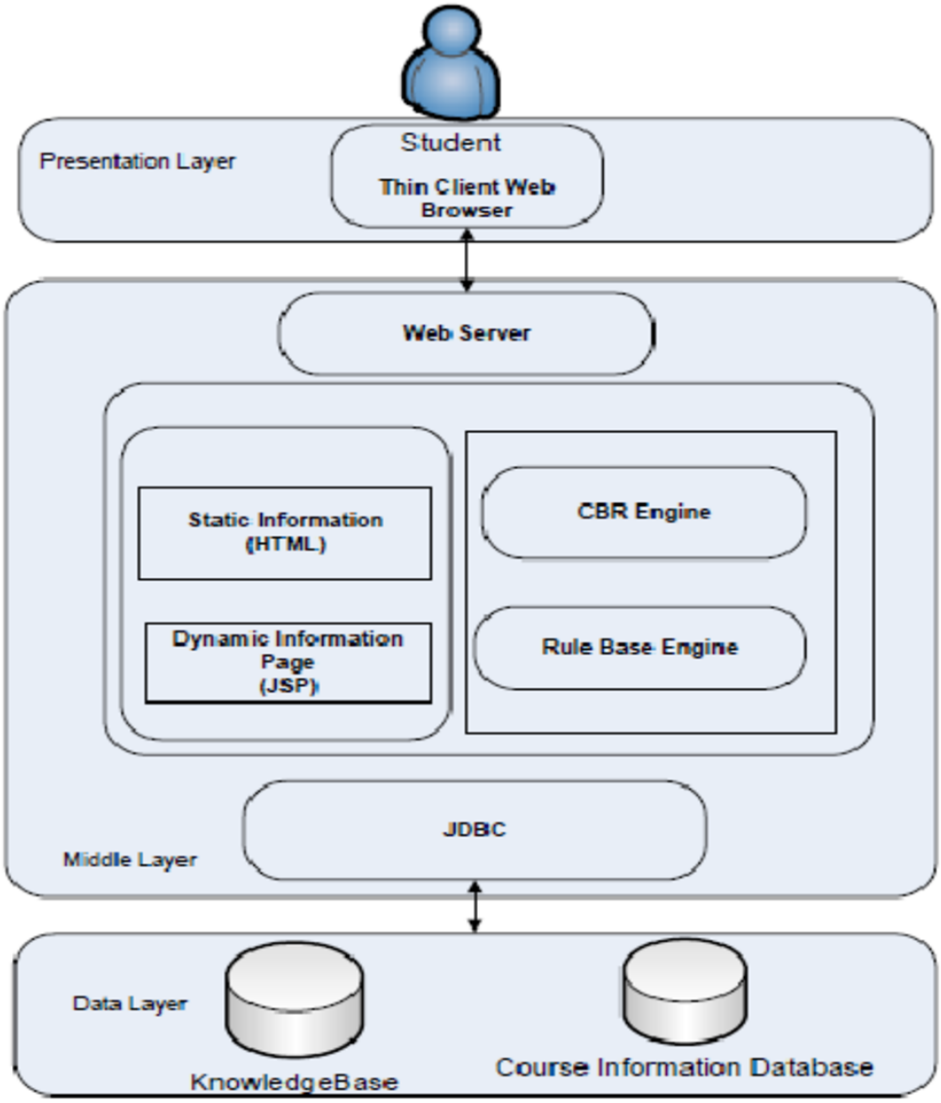


Figure 24: 3-tier Architecture of System

First tier is **Presentation** tier, in which user interacts with leave management system through any web browser (Google chrome, Internet Explorer etc…).It is interface between user and application.

Second tier is **Business** Tier, in which all system implementation is written. Having dynamic pages written in PHP. This will run whole system in server. It is interface between application and database.

Third tier is **Data Base** tier, in which all the information of users and application is stored. All activities, modifications and whole data is maintained by database. With help of queries user retrieve data from it.

3.4 Workflow

A person should be able to login to the system through the login page of the application, change the password, apply for leave specifying the from and to dates, and approve/ reject the leave applications that are submitted to him / her. As soon as a leave application / cancellation request /approval / rejection is made by the person he / she will be notified.

The number of days of leave (as per the assumed leave policy) should be automatically credited to everybody. The processing leave should be approved by the system automatically after some days.

Project have total 3 modules

- Student Module
- Teacher Module
- Admin Module

3.4.1 Admin Module

First for the initial application page by clicking on Admin, it will redirect to admin login page. For accessing this web application, Admin have to login with his unique username and password. After logged in as Admin he will be redirected to his Admin Module.

Admin module have several functions like:

- **Change Password**: Admin can change his password by giving old password and new password he wants to update.
- **Manage Leave** : In this section of web page admin get requests from teachers for leave. Admin may approve or reject the leaves applied by staff.
- **Register Users** : Main duty of Admin is to register users. Admin can register teacher or student in this web application system by giving their details such as name,student/teacher, password,username and so on.
- **Employee Leave History**: Records of all teachers leaves are maintained. These records contain information like Teacher name, type of leave applied, no.of days applied , status of leave(approved/pending/rejected).
- **Help**: If a new user doesn't know how to access this application he can access this page and it gives detailed information of this application and help them to access application.

3.4.2 Teacher Module

Teacher Module functionalities are:

- **Change Password:** A teacher can change password by giving old password as a proof of identity and a new password to change.
- **Apply Leave:** A teacher can apply leave by specifying type of leave(casual/medical/earned/meternity) along with from date and to date. A request of this leave send to admin.
- **Manage Leave:** Students' leave requests are send to teacher. Teacher can approve or reject student leave.
- **Leave Balance :** A table of no.of days of leave left for each of type is left is displayed.
- **Leave Status:** The status(approved/pending/rejected) of applied is displayed.
- **Student Leave History:** A record of all students leave details is maintained. It contains name of student, type of leave applied and no.of days applied and so on…

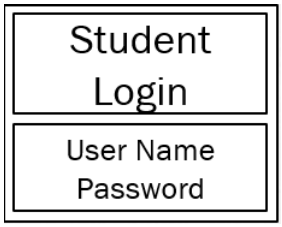
3.4.3 Student Module

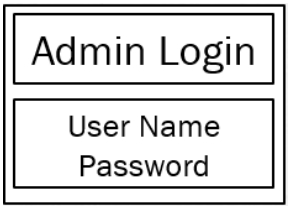
Student Module Functionalities are:

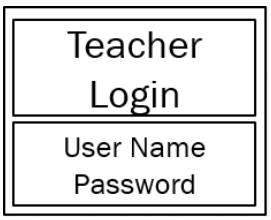
- **Change Password** : A student can change password by giving old password as a proof and a new changed password.
- **Apply Leave** : A student can apply leave by specifying type of leave,from date and to date.
- **Leave Balance**: A table of type of leave along with no.of days left is displayed.
- **Leave Status**: The status of applied leave is displayed.
- **Leave History** : A record of all applied leave details is maintained. It has type of leave, no.of days applied and status of each leave.

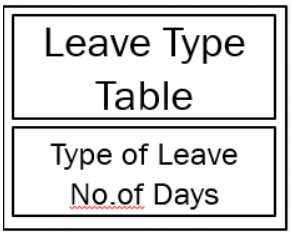
3.5 Data Base Design

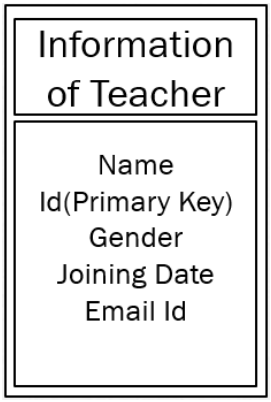
To Implement Leave Management System for School a database with 9 tables are created.

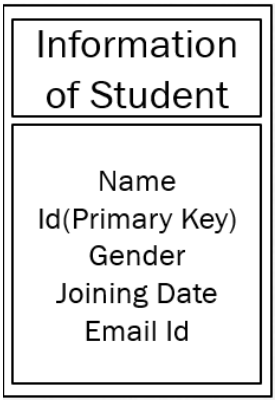


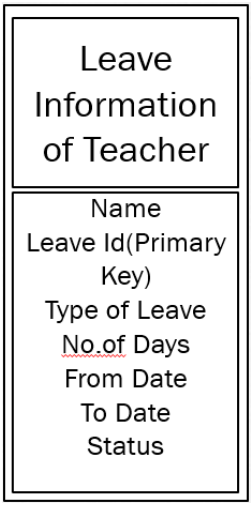


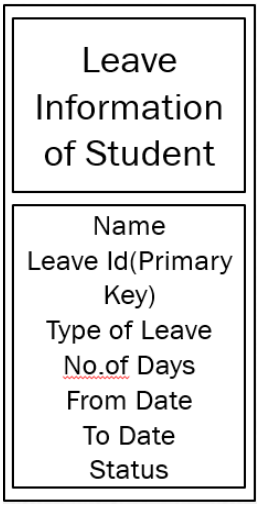


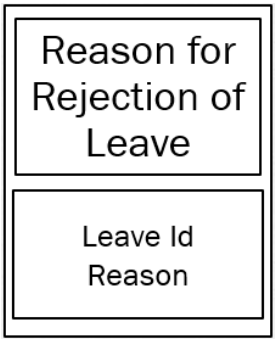












4

IMPLEMENTATION

4.1 Technologies Used

- Front End : HTML , CSS
- Back End : PHP
- Data Base: MySQL

HTML

- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

CSS

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files
- CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

PHP

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language. PHP scripts are executed on the server
- It is powerful enough to be at the core of the biggest blogging system on the web. It is deep enough to run the largest social network
- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP can generate dynamic page content and collect data from user.

MySQL

- MySQL is the most popular Open Source Relational SQL Database Management System.
- MySQL is one of the best RDBMS being used for developing various web-based software applications.
- MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company.

SQL

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert , delete ,update records of Database.
- SQL can create new databases
- SQL can create new tables in a database

4.2 Implementation of Domain Objects Layer and Technical Service Layer

Code for Index Page:

```
<html>
<head>
<title>Login</title>
        <link href="sty.css" rel="stylesheet" type="text/css" />
</head>
<body  style="background-image:url(back1.jpg);background-size:cover";>
 
 
 
<h1 style="color:green ;font-size:45px"><marquee>Online Leave
Management</marquee></h1>
<div>
<div class="header">
</div>
 <div id="menu">
          <ul>
           <li>
<a href="adlog.php"
style="float:left;fontsize:1.6vw;color:#000300">Admin</a></li>
         <li><a href="new1.html" style="float:left;font-size:1.6vw
;color:#000000">Leave</a></li>
      </ul>
   </div>
<center style="font-size:10vw ;font-style: italic ;padding:10px;
color:#fff708";>Easy Leave</center>
<center ><img src="leavepic.png" height="250"/></center>
```

```
</body>
</html>
```

Code for Login Page

```
<html>
<head>
<title>Login</title>
        <link href="styles.css" rel="stylesheet" type="text/css" />
        <style>
table, th, td {border: 1px solid black;}
        </style>
</head>
<body  style="background-image:url(1.jpg);">
 
 
 
<center><br><h1>WELCOME TO EASY LEAVE</h1></br>
<div class="login-page">
        <div class="form">
        <form action="#" method="post">
                <h3><font color="white">
                <table bgcolor="#FFFF00"  style="color:coral">
<tr>
<td><b><h2>User Name</h2></b></td>
<td><input type="text" name="userid" placeholder="username"
required/></td>
</tr>
<tr><td><b><h2>Password</h2></b>  </td>
```

```
<td><input type="password" name="password" placeholder="password"
required/></td></tr>
</table>
</font></h3>
                <button>submit</button>
        </form></div></div>
</center>

<?php
session_start();
if(isset($_POST['submit']))
{
 mysql_connect('localhost','root','') or die(mysql_error());
 mysql_select_db('anu') or die(mysql_error());
 $userid=$_POST['userid'];
 $password=$_POST['password'];
 if($userid!=''&&$password!='')
 {
  $query=mysql_query("select * from login where username='".$userid."' and
password='".$password."'") or die(mysql_error());
  $res=mysql_fetch_row($query);
  if($res)
  {
   $_SESSION['userid']=$userid;
        $_SESSION['name']=$userid;
        $_SESSION['password']=$password;
   header('location:lp.html');
  }
  else
  {  echo'You entered username or password is incorrect';  }
```

```
 }
 else
 {
  echo'Enter both username and password';
 }
}
?>
</body>
</html>
```

Code for Apply Leave

```
<html>
<head>
        <title>apply leave</title>
        <link href="sty.css" rel="stylesheet" type="text/css" />
        <style>
        ul { list-style-type: none; margin: 0;padding: 0;overflow: hidden;
   background-color: #333;}
li {float: left;}
li a {display: block;color: white; text-align: center; padding: 14px 16px;
   text-decoration: none;}
li a:hover {background-color:#2E8B57 ;}
        </style>
</head>
<body  style="background-image:url(im6.jpg)";>
<i><b><ul>
  <li><a href="new.html">Home</a></li>
  <li><a href="changepass.php">Change Password</a></li>
  <li><a href="applyleave.php">Apply Leave</a></li>
  <li><a href="leavebal.php">Leave Balance</a></li>
  <li><a href="leavestatus.php">Leave status</a></li>
```

```
 <li><a href="leavehistory.php">Leave History</a></li>

 <li><a href="help.php">Help</a></li>

 <li style="float:right"><a href="new.html">Logout</a></li>

  <li style="float:right"><a href="new.html"><?php session_start(); echo
"welcome  ".$_SESSION['name']."";?></a></li></ul></b></i>

<center><br><img src="l3.jpg" height="125" width="125" class="im"><font
color="#FF00FF" size="5vm"/><h1>Apply Leave</h1>

<div class="login-page">

<div class="form">

        <form action="#" method="post">

                <h3><font color="white">

                <table style="color:coral">

<tr><td><font color="#8A2BE2" size="5vm"/><b>Type of Leave</b></td>

<td><font color="#8A2BE2" size="5vm"/>

<select name="type" placeholder="Type of Leave"required></br></br>

                <option value="Type of Leave">select</option>

                <option value="casual">casual</option>

                <option value="medical">medical</option>

</select>

</td></tr>

<tr><td><font color="#8A2BE2" size="5vm"/><b>From Date</b>  </td>

<td><input type="date" name="from" placeholder="From
Date"required></td></tr>

<tr><td><font color="#8A2BE2" size="5vm"/><b>To Date</b>  </td>

<td><input type="date" name="to" placeholder="To
Date"required></td></tr>

<tr><td colspan=2><center><input type="submit"  class="button"
name="ApplyLeave" value="APPLYLEAVE"/></center></td></tr>

</table></font></h3>

</form></center>

<?php

  $con = mysqli_connect('localhost','root','') or die(mysqli_error());

 mysqli_select_db($con,'anu') or die(mysqli_error());
```

```
 if(isset($_POST['ApplyLeave']))
{
        if($_SESSION['name']!=''&&$_SESSION['password']!='')
        {
        $lid=rand();
        $type1=$_POST['type'];
        $from=$_POST['from'];
        $to=$_POST['to'];
        $start=new DateTime($from);
        $end=new DateTime($to);
        $days=$start->diff($end,true)->days;
        $sundays=intval($days/7)+(($start->format('N')+$days%7)>=7);
        $n=$days-$sundays;
        $query=mysqli_query($con,"select * from info1  where
Name='".$_SESSION['name']."'");
        $re=mysqli_fetch_array($query,MYSQLI_ASSOC);
      $c='';$e='';$m='';$mt='';
        $_SESSION['type1']=$type1;
        $_SESSION['to']=$to;
        $_SESSION['from']=$from;
        $_SESSION['lid']=$lid;
        if($type1=='casual')
        {
                $c=15-$re['Days'];
                if($days>4)
                {
                        header("location:cl.php");
                        exit();
                }
                else if($c==0)
```

```
        {
        header("location:cl2.php");
                        exit();
        }
        else if($c<$days)
        {$_SESSION['c']=$c;
                        header("location:cl3.php");
                        exit();
        }
        else
        {
        $n=$days;
        $_SESSION['n']=$n;
        header("location:cl4.php");
                        exit();
                        }
}
if($type1=='medical')
{
        $m=15-$re['Days'];
        if($m==0)
        {
                        header("location:m1.php");
                        exit();
        }
        else if($m<$n)
        {
        $_SESSION['m']=$m;
                header("location:m2.php");
                        exit();
        }
```

```
        else
        {
        $days=$m-$n;
        $_SESSION['n']=$n;
                header("location:m3.php");
                }
}
if($type1=='earned')
{
        $e=15-$re['Days'];
        if($e==0)
        {
                        header("location:e1.php");
                        exit();
        }
        else if($e<$n)
        {
                $_SESSION['e']=$e;
                header("location:e2.php");
                exit();
        }
        else
        {
        $days=$e-$n;
        $_SESSION['n']=$n;
        header("location:e3.php");
        }
}
if($type1=='maternity')
{
```

```
                if($gen=="female"||$gen=="Female")
                {
                        $mt=60-$re['Days'];
                        if($mt==0)
                        {
                        header("location:mt1.php");
                        }
                        else if($mt<$n)
                        {
                                $_SESSION['mt']=$mt;
                                header("location:mt2.php");
                        }
                        else
                        {
                                $days=$mt-$n;
                                $_SESSION['n']=$n;
                                header("location:mt3.php");
                        }
                }
                else
                {
                        header("location:mt4.php");
                }
        }
}
}
?>
</body>
</html>
```

5

TESTING

5.1 Introduction to Testing

Testing is a fault detection technique that tries to create failure and erroneous states in a planned way. This allows the developer to detect failures in the system before it is released to the customer.

Note that this definition of testing implies that a successful test is test that identifies faults. We will use this definition throughout the definition phase. Another often used definition of testing is that it demonstrates that faults are not present.

Testing can be done in two ways:

1. Top down approach

2. Bottom up approach

1. Top down approach: This type of testing starts from upper level modules. Since the detailed activities usually performed in the lower level routines are not provided stubs are written.

2. Bottom up Approach: Testing can be performed starting from smallest and lowest level modules and proceeding one at a time. For each module in bottom up testing a short program executes the module and provides the needed data so that the module is asked to perform the way it will when embedded within the larger system.

In this project, bottom up approach is used where the lower level modules are tested first and the next ones having much data in them.

Testing Methodologies

The following are the Testing Methodologies:

1. Unit Testing.

2. Integration Testing.

3. User Acceptance Testing.

4. Output Testing.

1. Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a modules control structure to ensure complete coverage and maximum error detection.

This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing. During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification.

All important processing path are tested for the expected results. All error handling paths are also tested.

2. Integration Testing

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted.

The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

i. Top Down Integration

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module.

The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner. In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

ii. Bottom-up Integration

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated.

The bottom up integration strategy may be implemented with the following steps:

1. The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.

2. A driver (i.e.) the control program for testing is written to coordinate test case input and output.

3. The cluster is tested. d. Drivers are removed and clusters are combined moving upward in the program structure. The bottom up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

User Acceptance Testing

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required.

The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

Output Testing

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format.

Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways one is on screen and another in printed format.

5.2 Test Cases

- Test application working in an accurate way or not.
- Test time taken to redirect to other page.
- Test clicking action of buttons.
- Test all direct and alternative flows

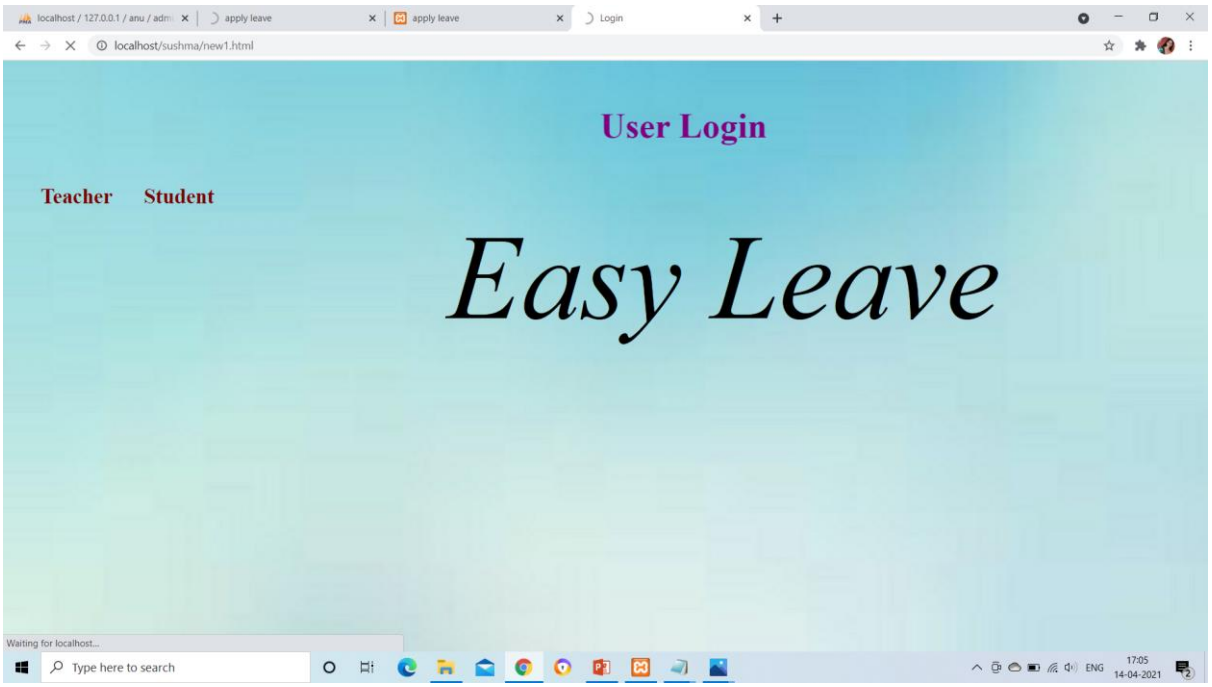
6

RESULTS

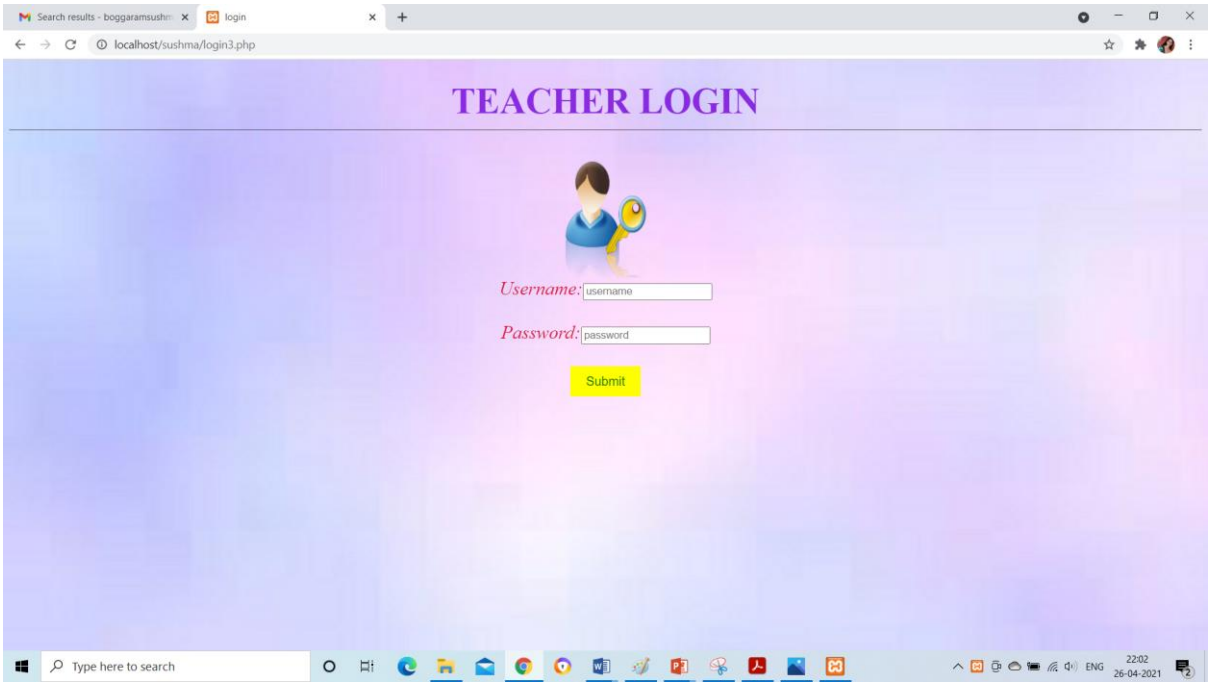
USER INTERFACE LAYER

Welcome page of Easy Leave:

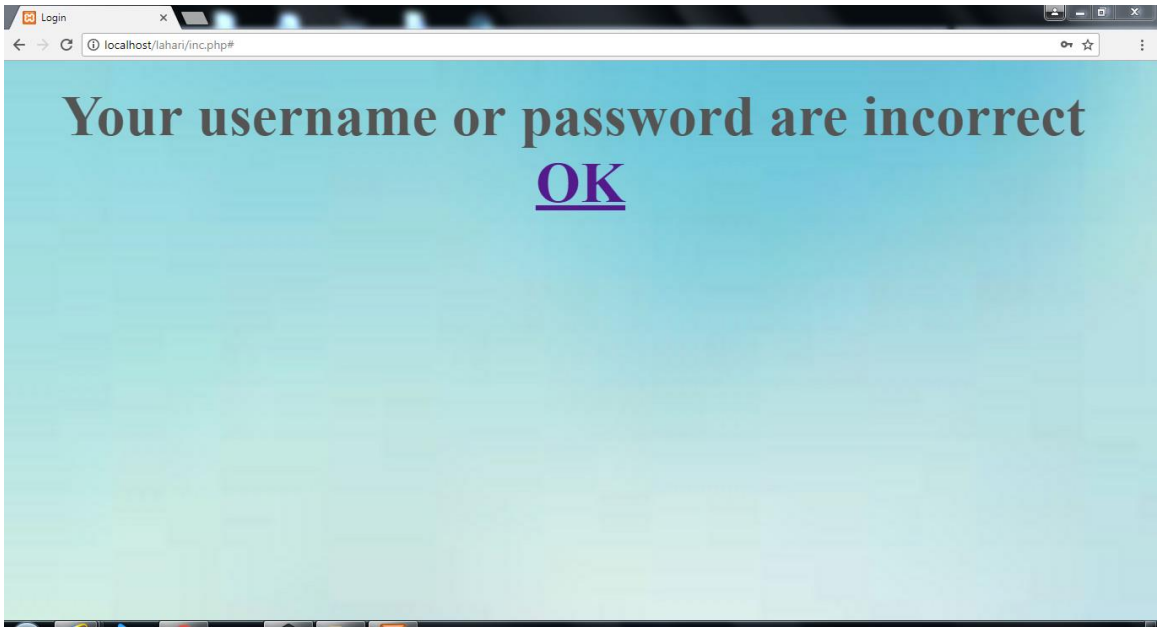
When Leave is chosen:



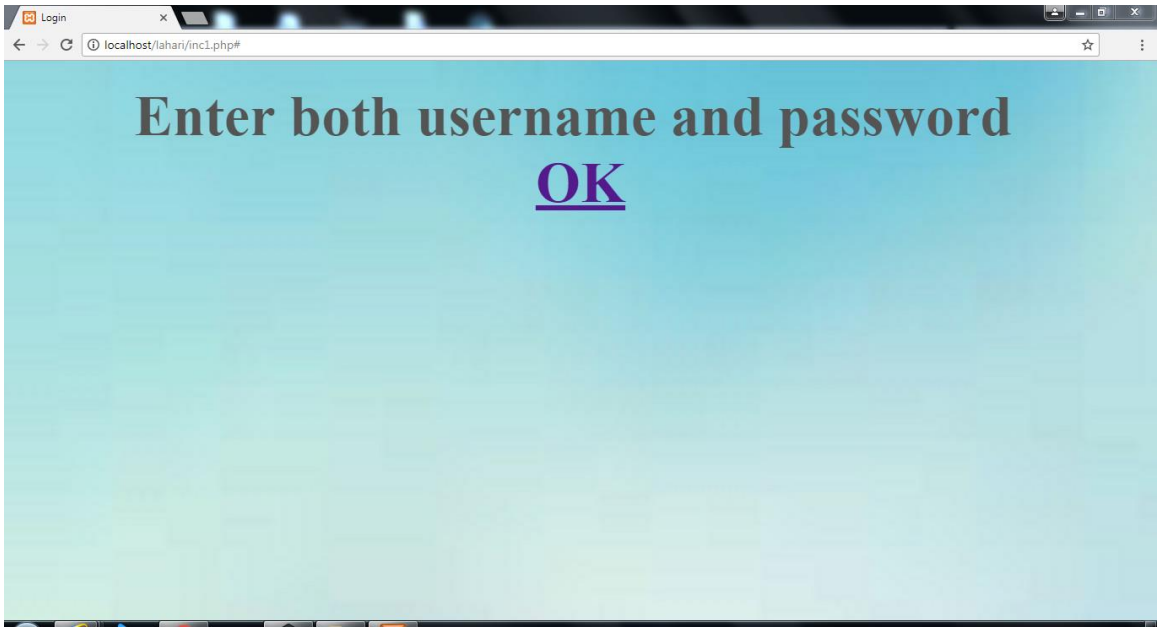
When Teacher is chosen:



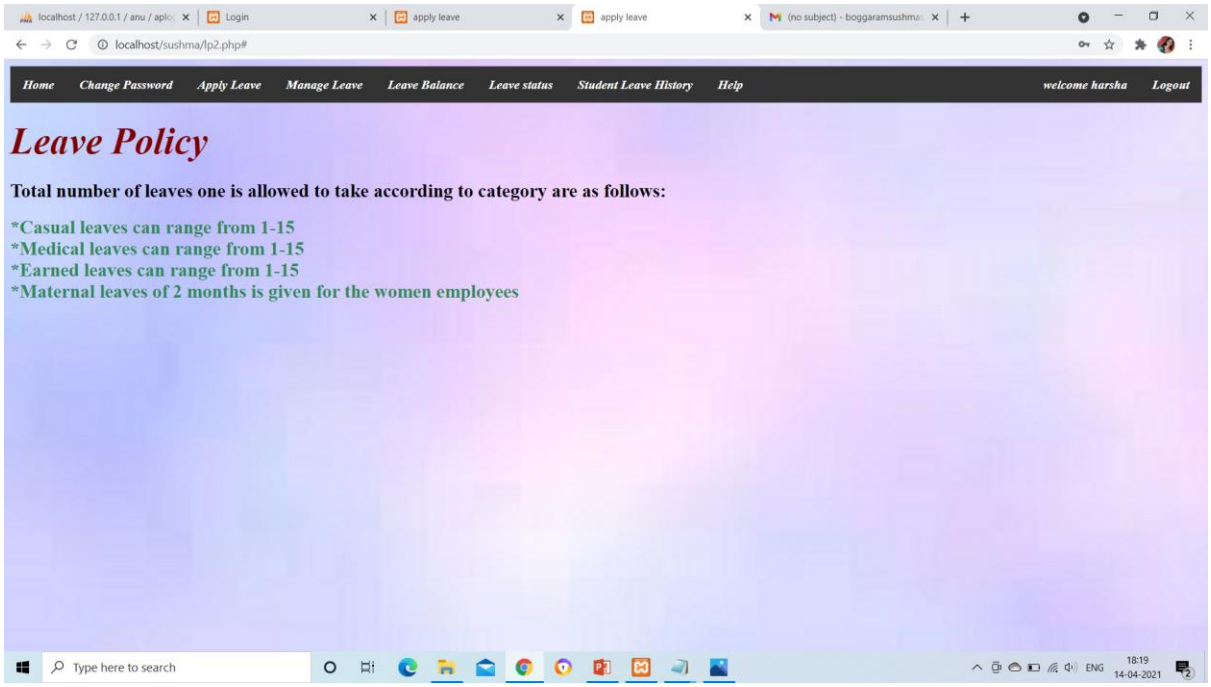
When username or password is wrong:



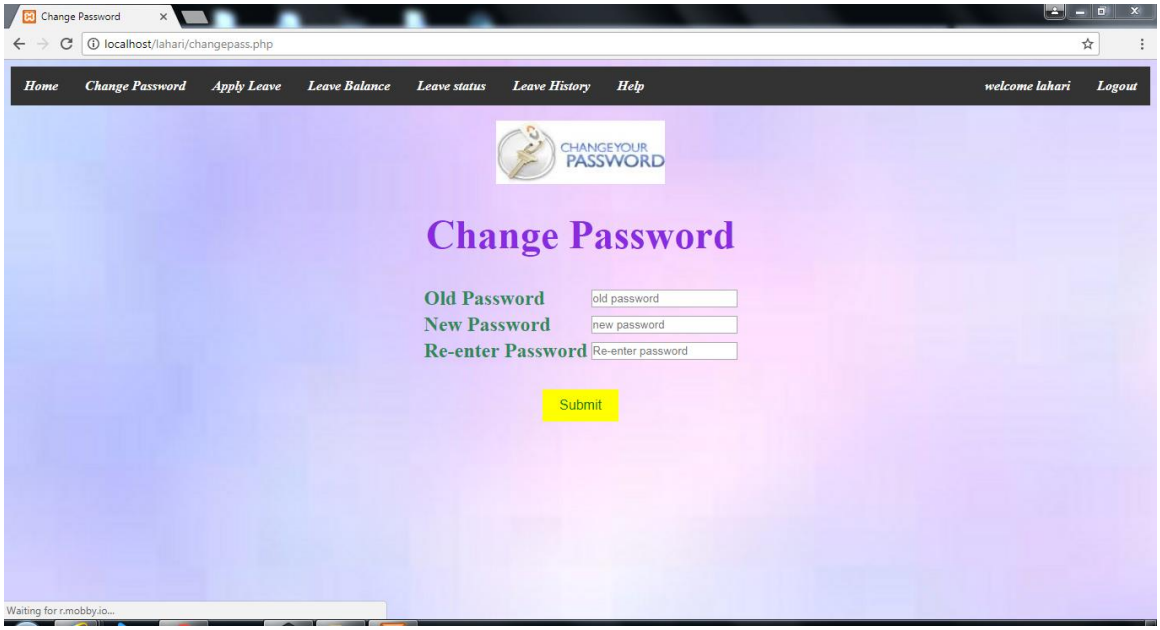
When username and password are not entered:



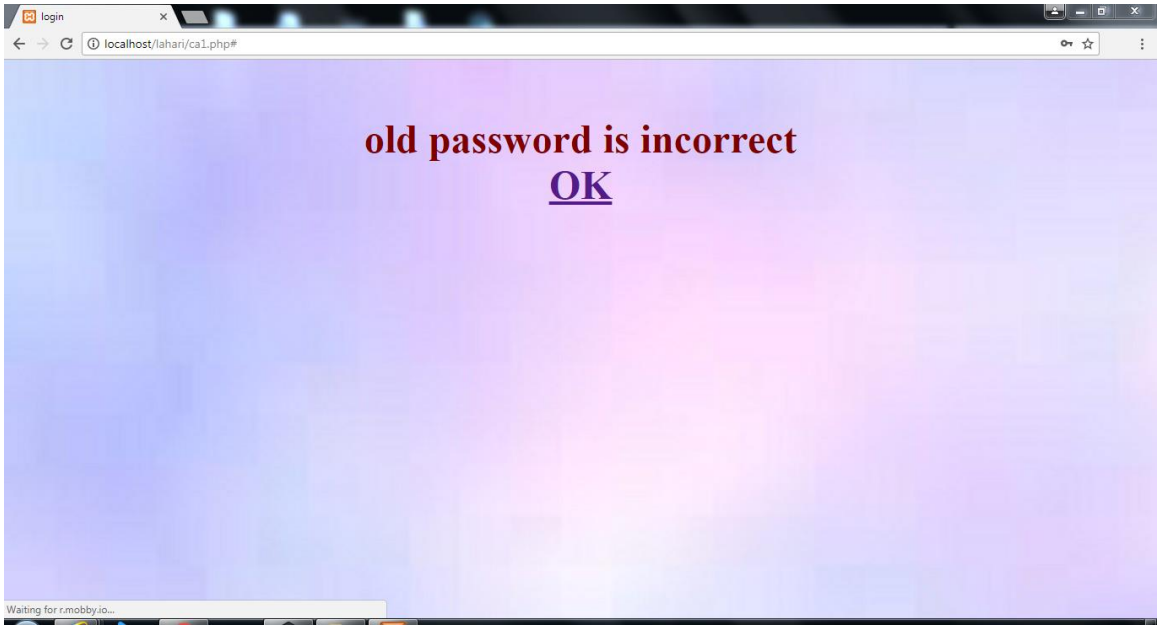
When teacher logs in:



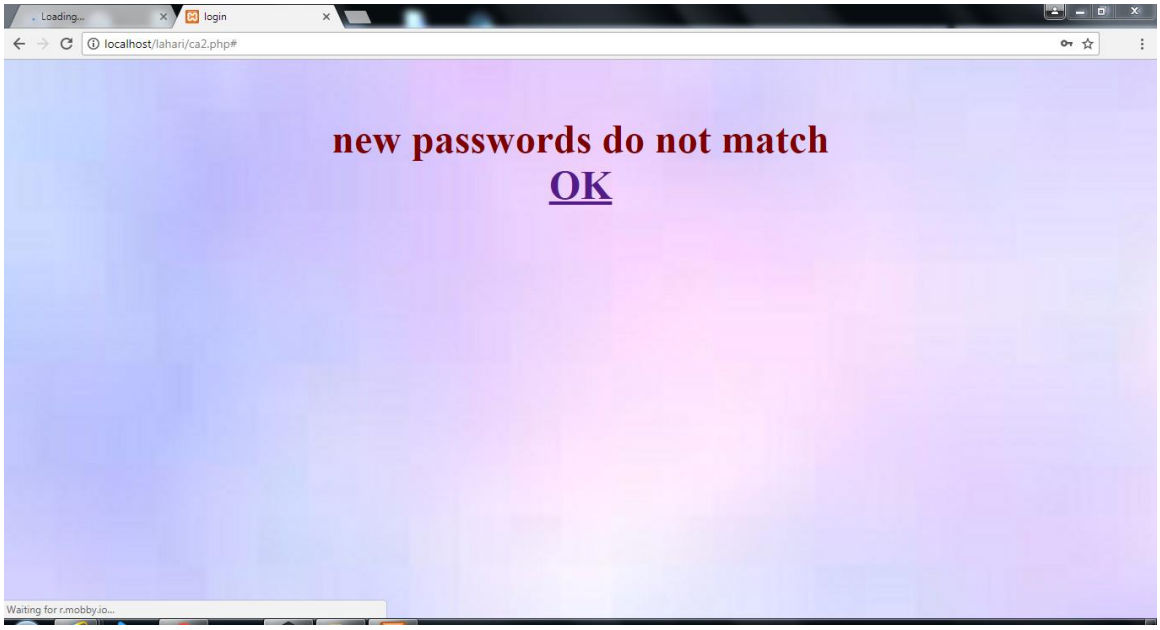
User can change password from change password tab:



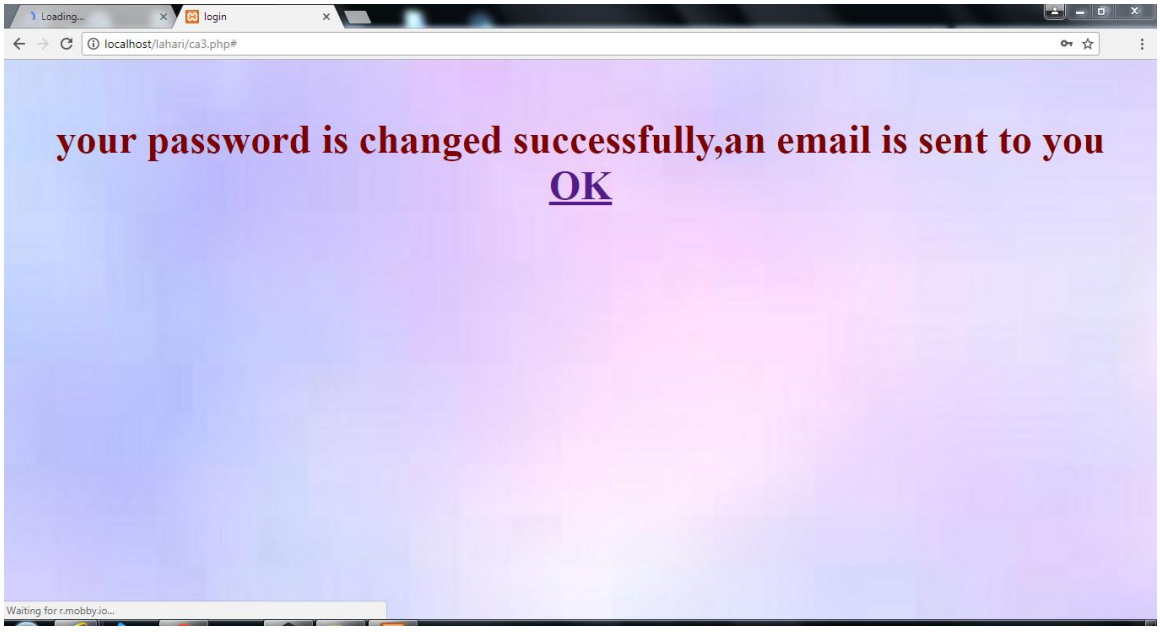
If old password is not correct:



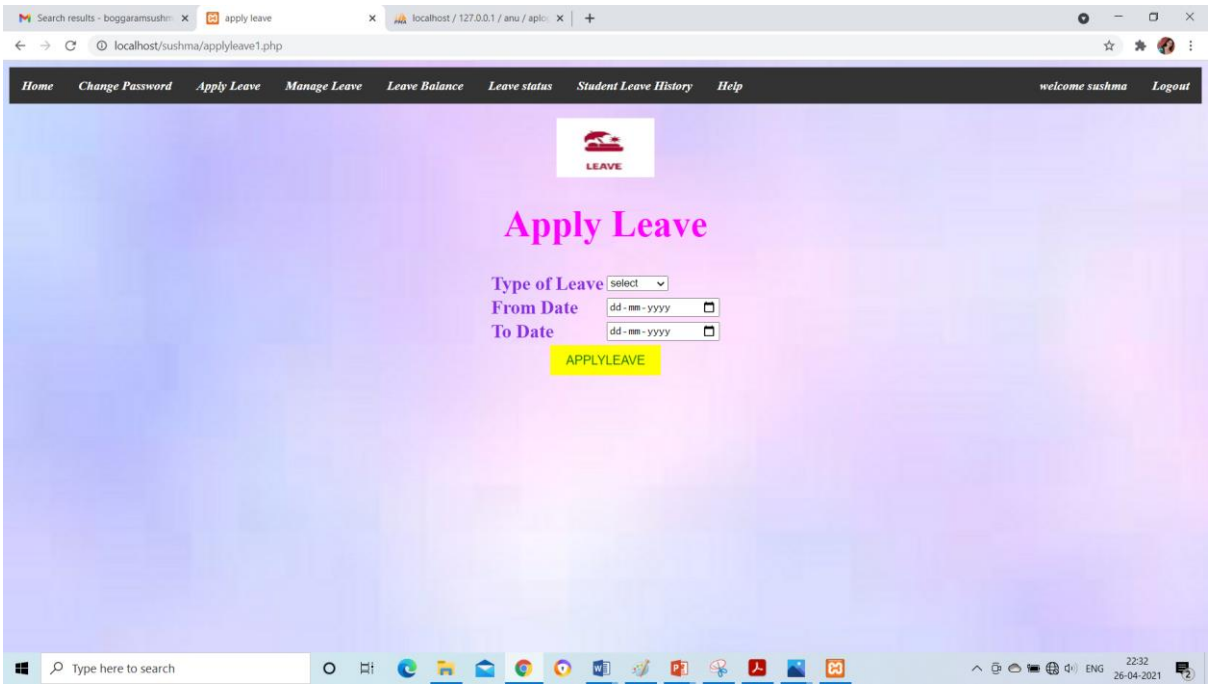
If new passwords do not match:



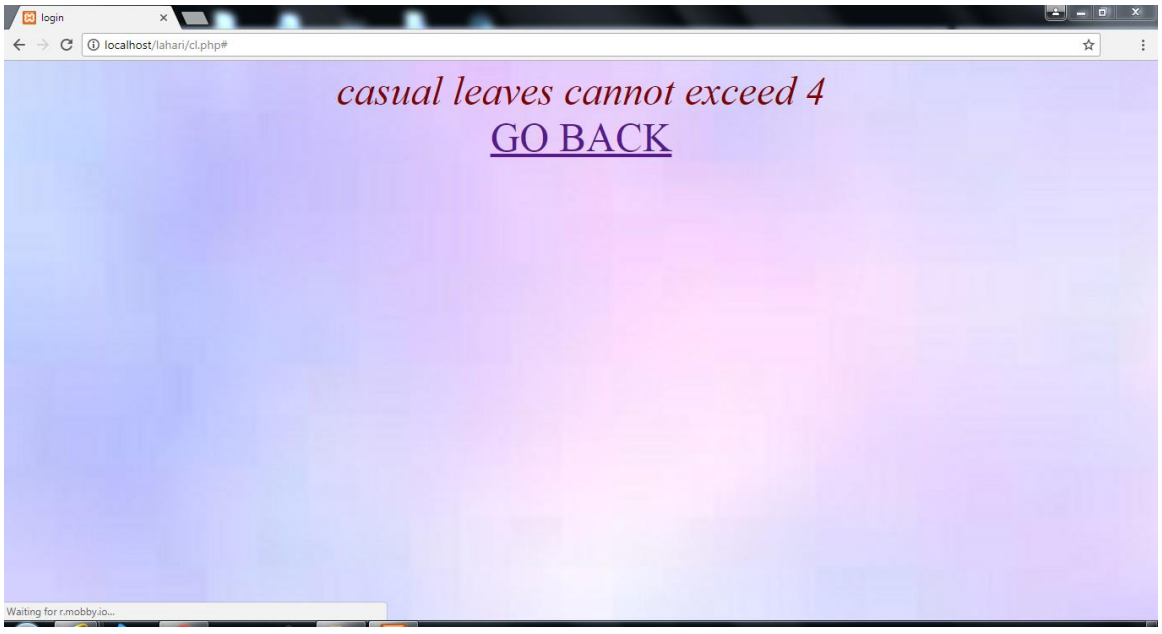
If password is changed successfully:

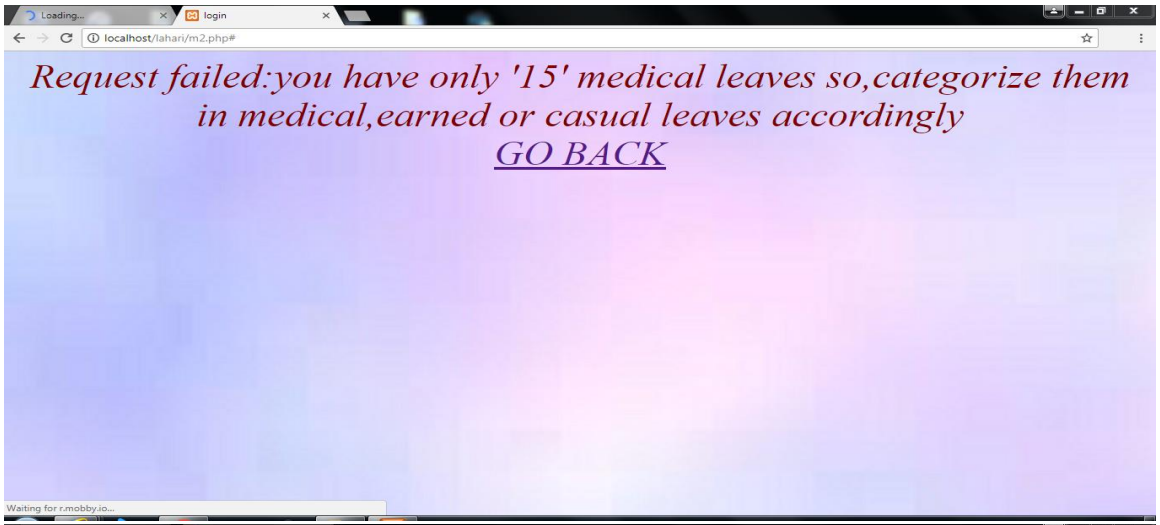


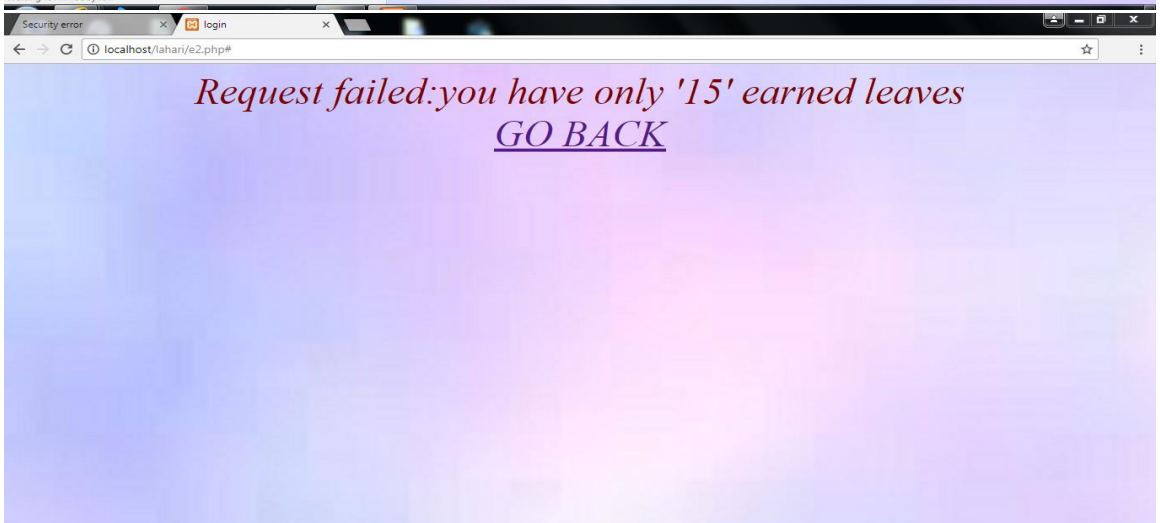
When approver wants to apply leave:

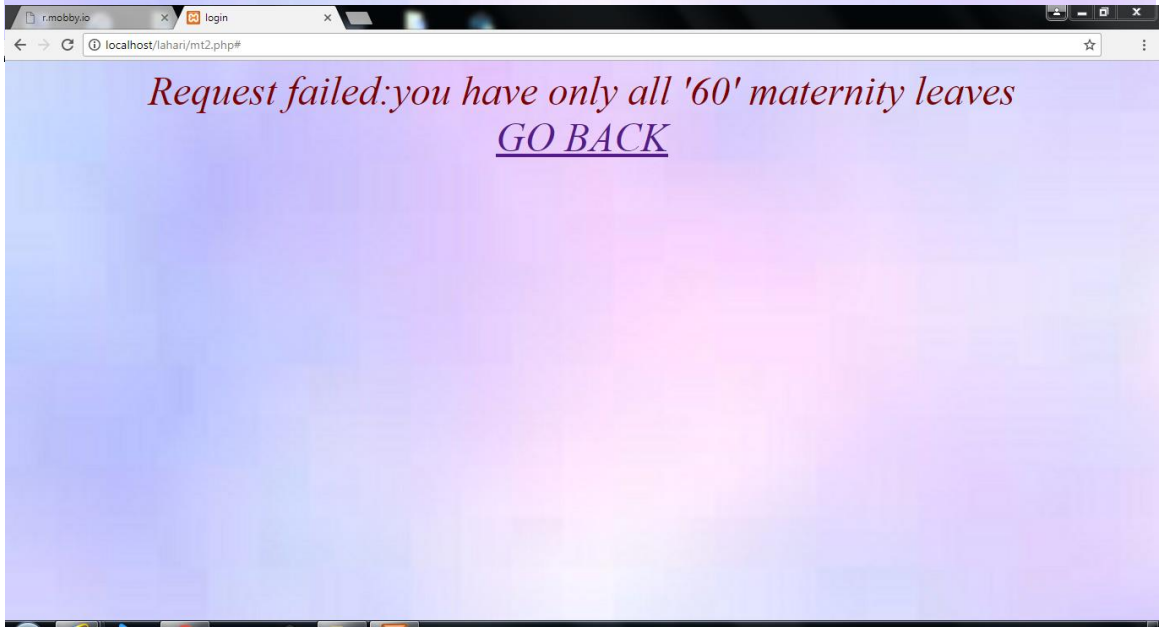


When the necessary conditions are not satisfied while applying leave:

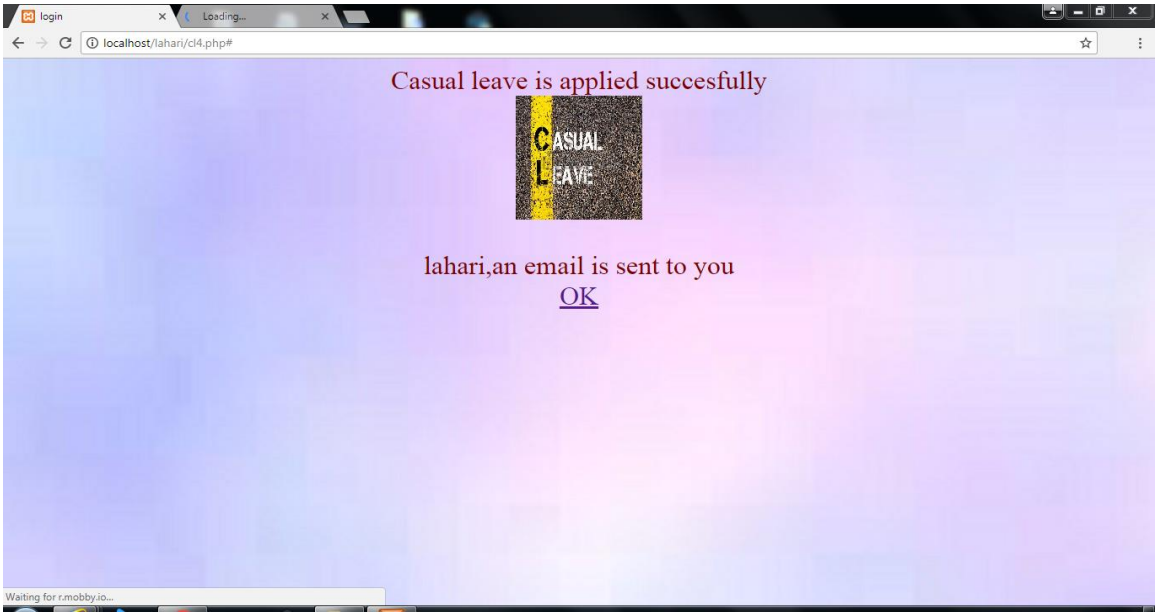


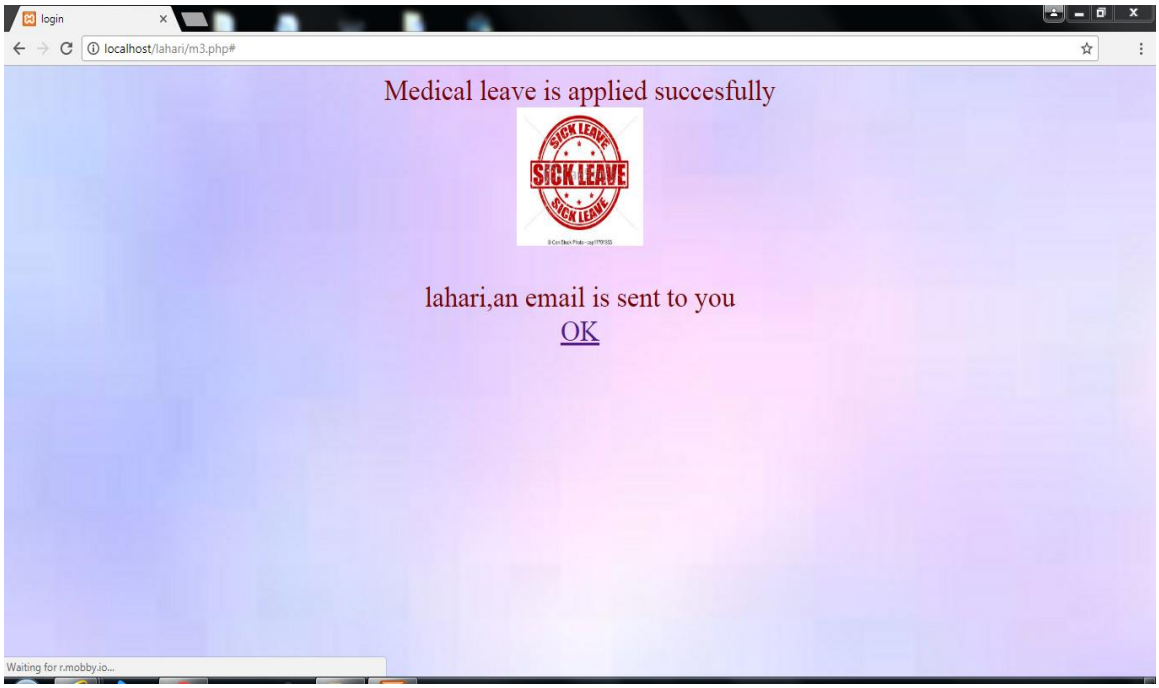

Loading...
login
localhost/lahari/m2.php#
Request failed:you have only '15' medical leaves so,categorize them in medical,earned or casual leaves accordingly
GO BACK
Waiting for r.mobby.io...

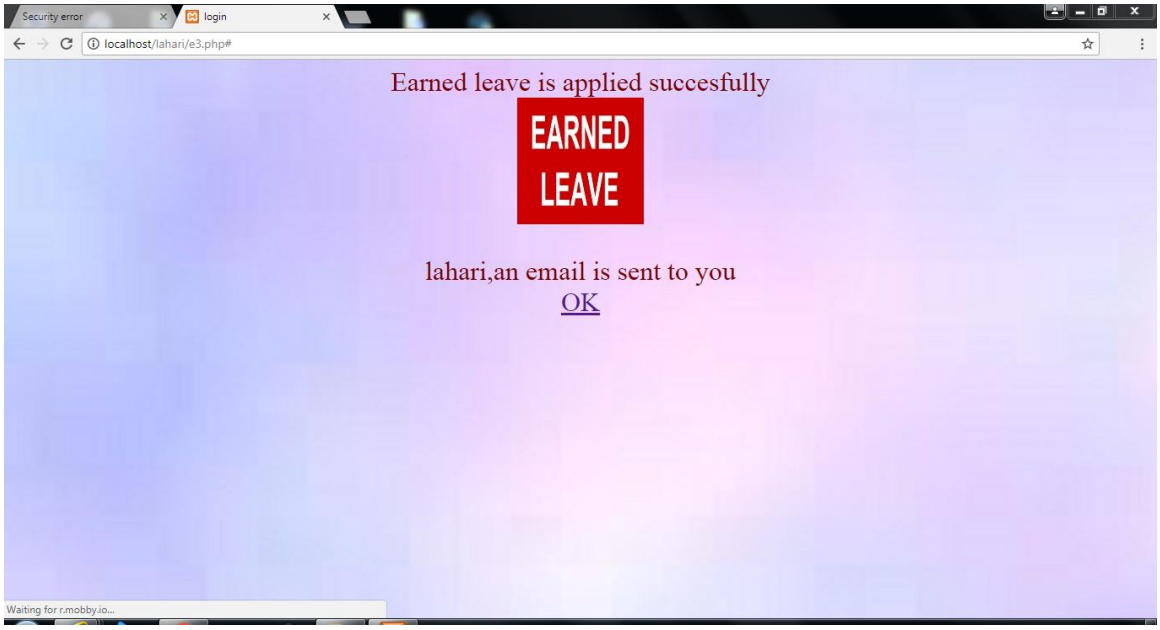

Security error
login
localhost/lahari/e2.php#
Request failed:you have only '15' earned leaves
GO BACK

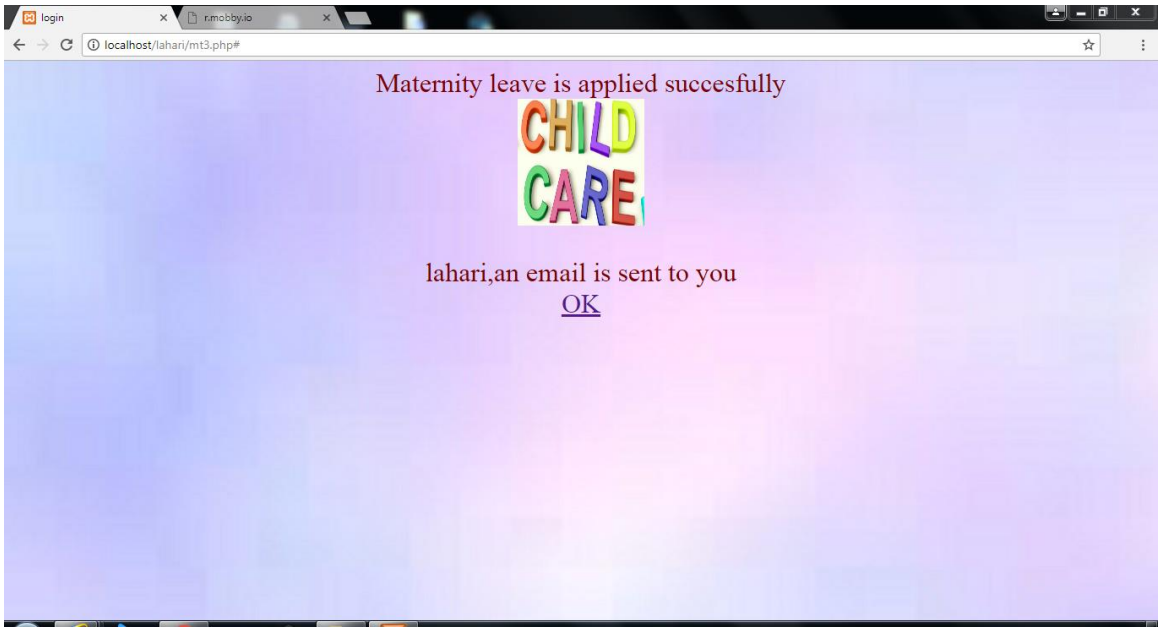

r.mobby.io
login
localhost/lahari/mt2.php#
Request failed:you have only all '60' maternity leaves
GO BACK

When all leave conditions are satisfied then:





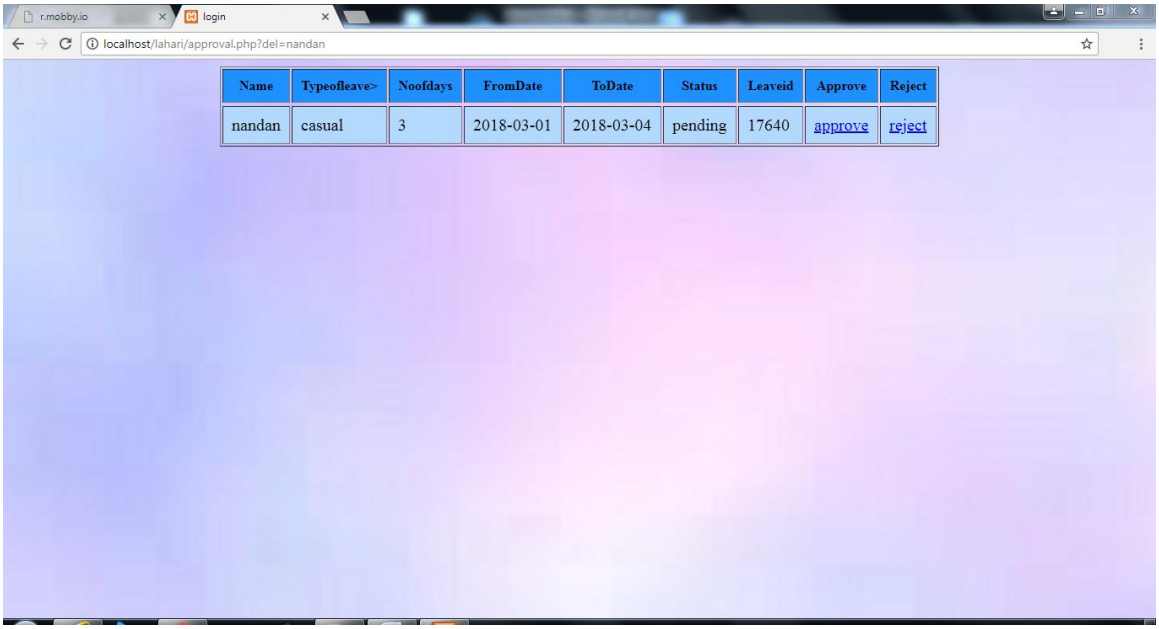

Security error
login
localhost/lahari/e3.php#
Earned leave is applied succesfully
EARNED LEAVE
lahari,an email is sent to you
OK
Waiting for r.mobby.io...


login
r.mobby.io
localhost/lahari/mt3.php#
Maternity leave is applied succesfully
CHILD CARE
lahari,an email is sent to you
OK

When teacher gets the leave requests he manages the leaves of Student from Manage Leave tab:

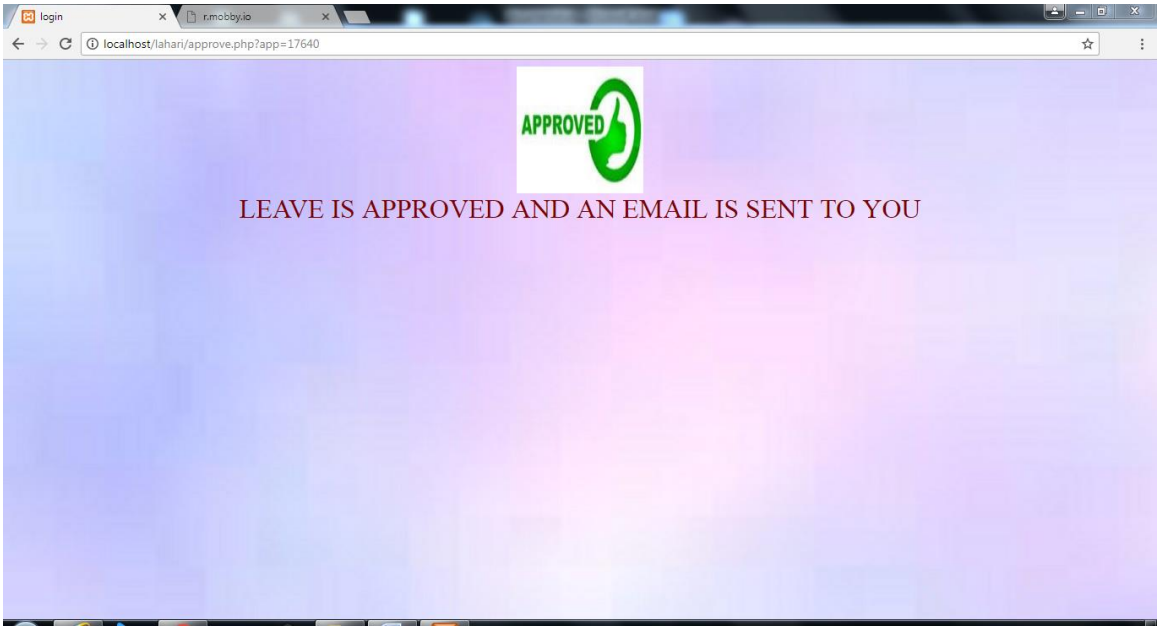
Name	Department	Typeofleave>	Noofdays	FromDate	ToDate	Status	Leaveid
lahari	Developing	casual	2	2018-03-01	2018-03-03	pending	19501
lahari	Developing	casual	2	2018-03-01	2018-03-03	pending	21858
lahari	Developing	medical	5	2018-03-01	2018-03-07	pending	4016
lahari	Developing	earned	6	2018-03-01	2018-03-08	pending	26030
lahari	Developing	maternity	11	2018-03-01	2018-03-14	pending	30428
nandan	testing	casual	3	2018-03-01	2018-03-04	pending	17640
lahari	Developing	medical	2	2018-03-01	2018-03-04	pending	6983

Teacher can approve or Reject by clicking on the name:



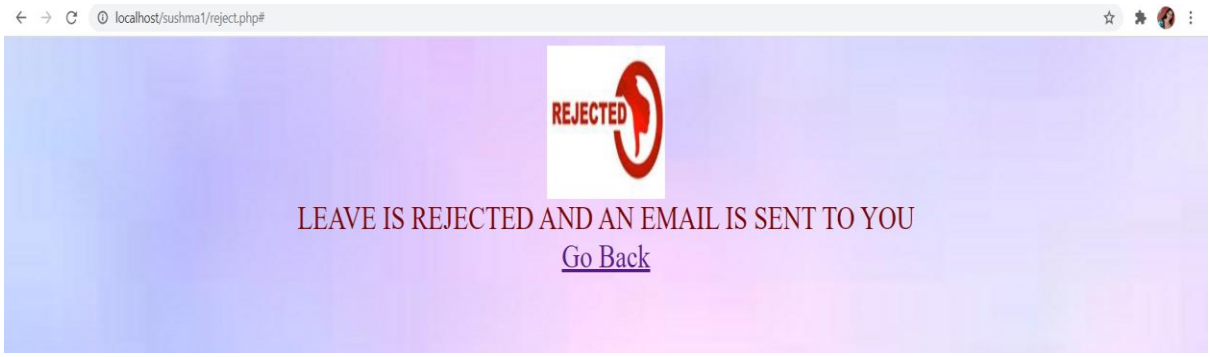
Name	Typeofleave>	Noofdays	FromDate	ToDate	Status	Leaveid	Approve	Reject
nandan	casual	3	2018-03-01	2018-03-04	pending	17640	approve	reject

On clicking approve the leave is approved:



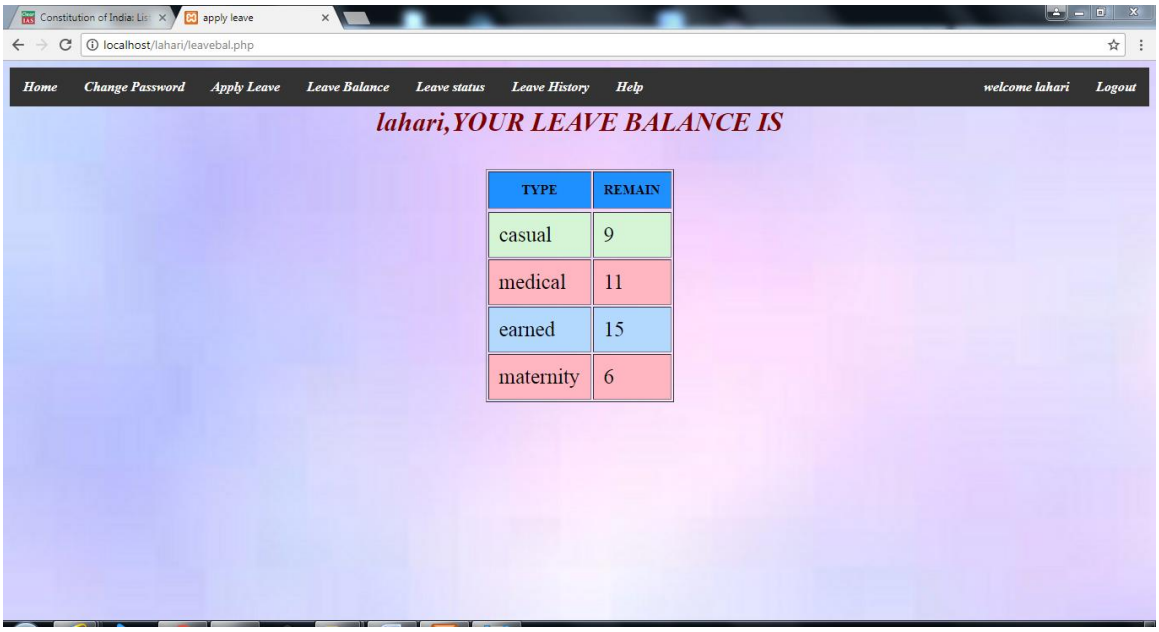
On clicking Reject it will ask for reason of Rejection:

By specifying Reason and clicking on submit Leave is Rejected:

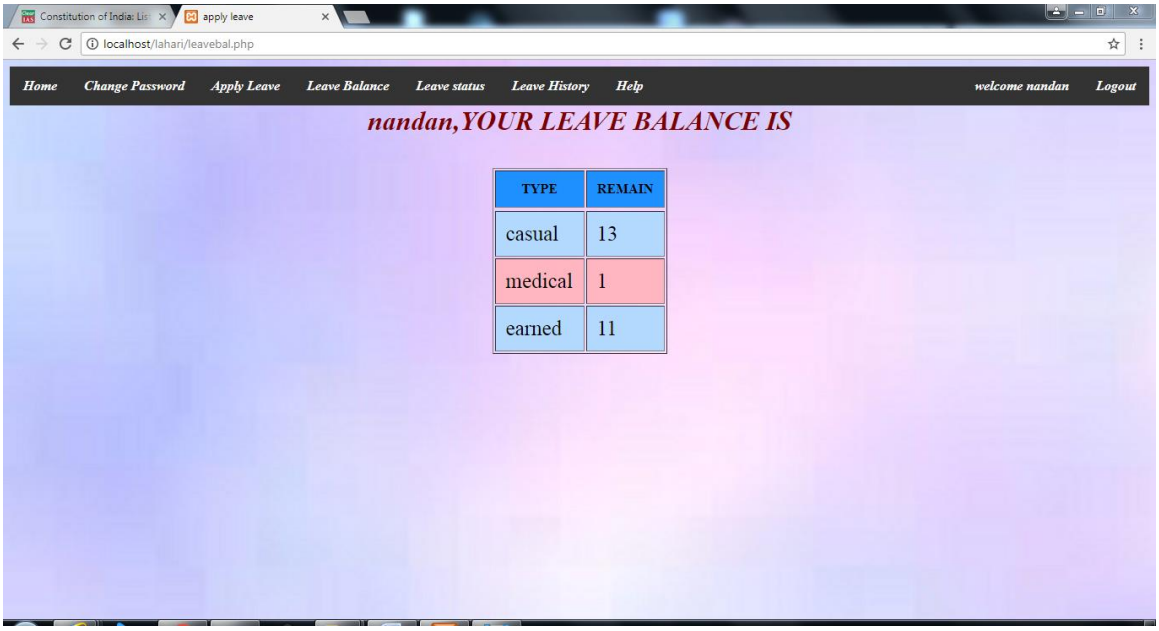


Teacher can view his leave balance from Leave Balance tab:

If gender is “Female” then leave balance is displayed as:

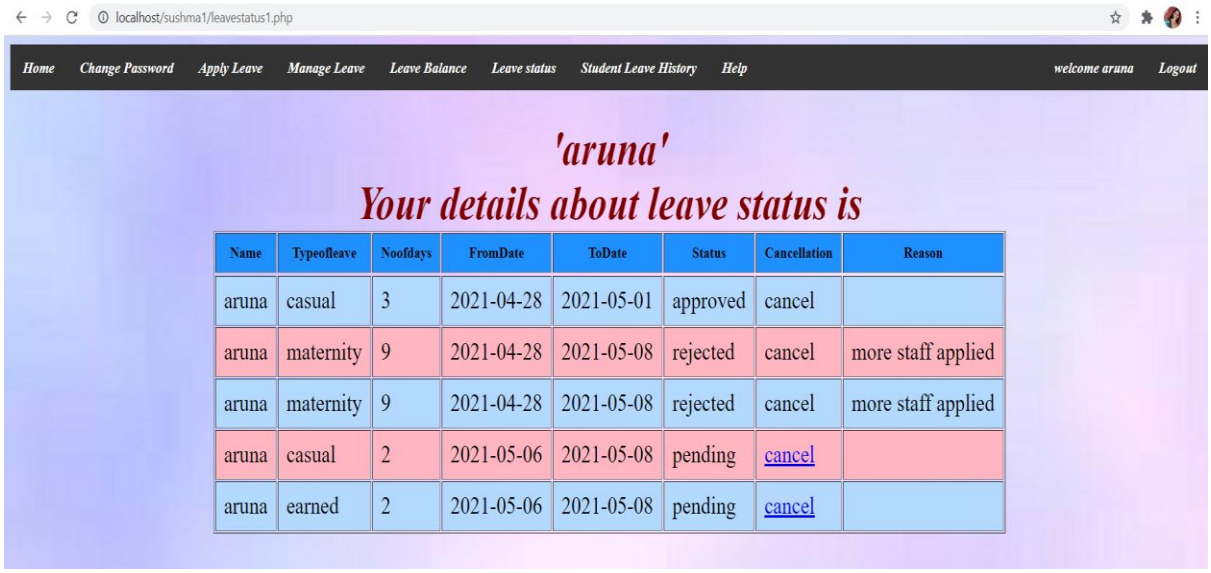


If gender is “Male” then leave balance is displayed as:

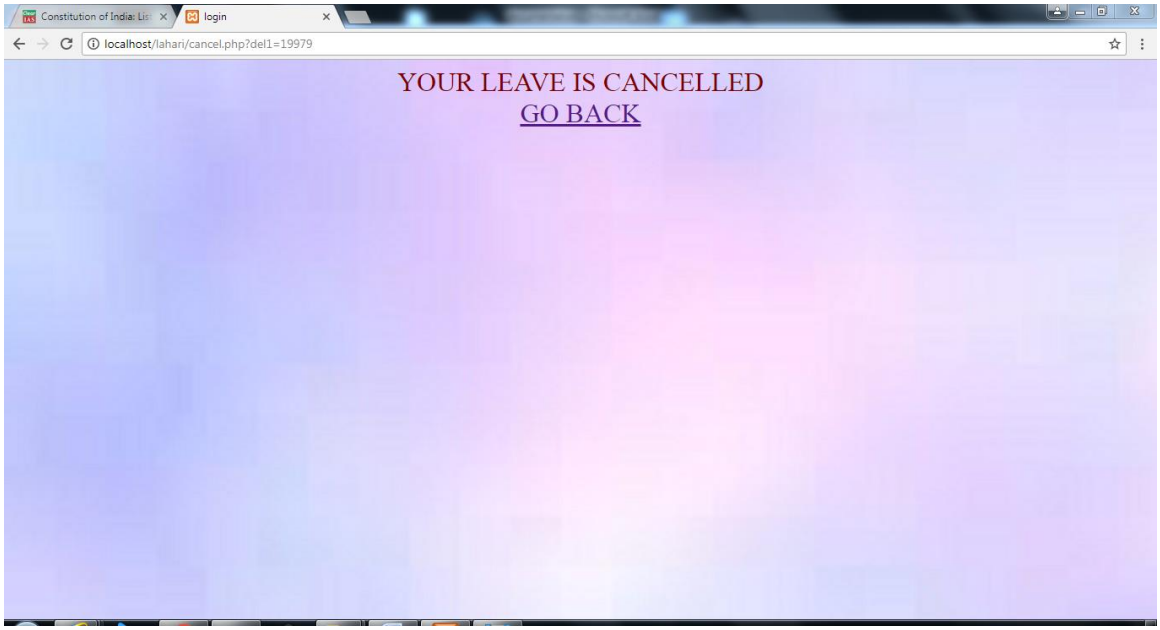


Leave status can be checked from leave status tab:

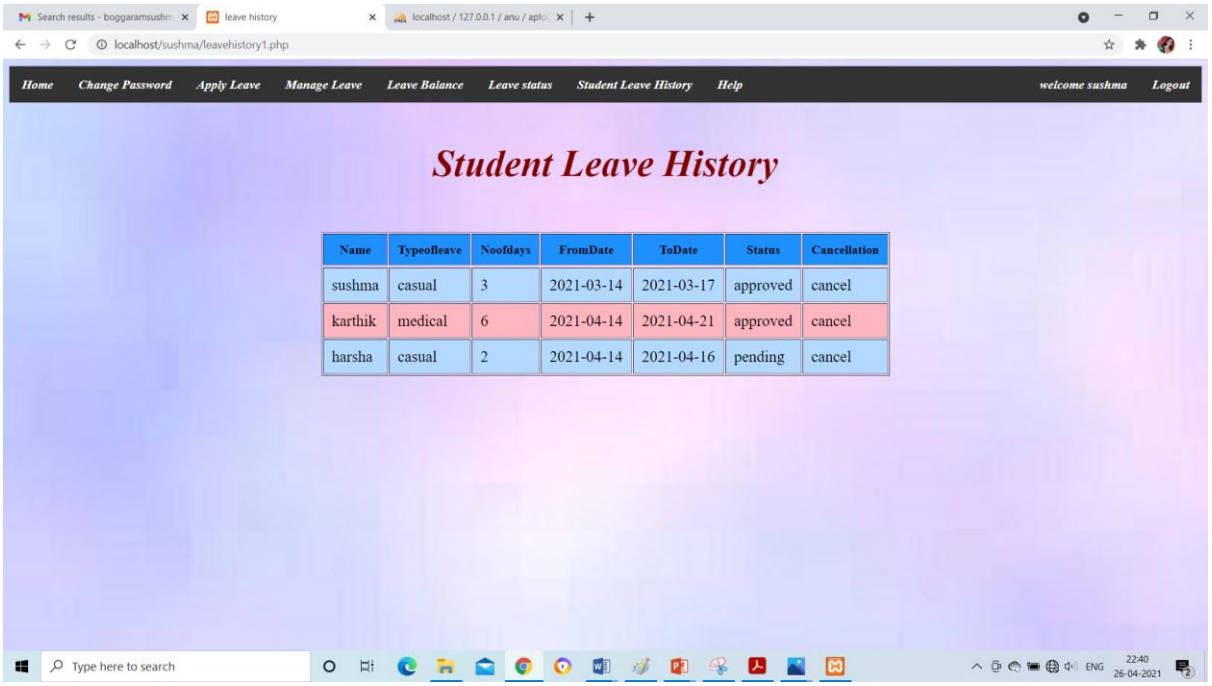
Leave request can be cancelled by clicking on cancel:



On clicking on cancel :



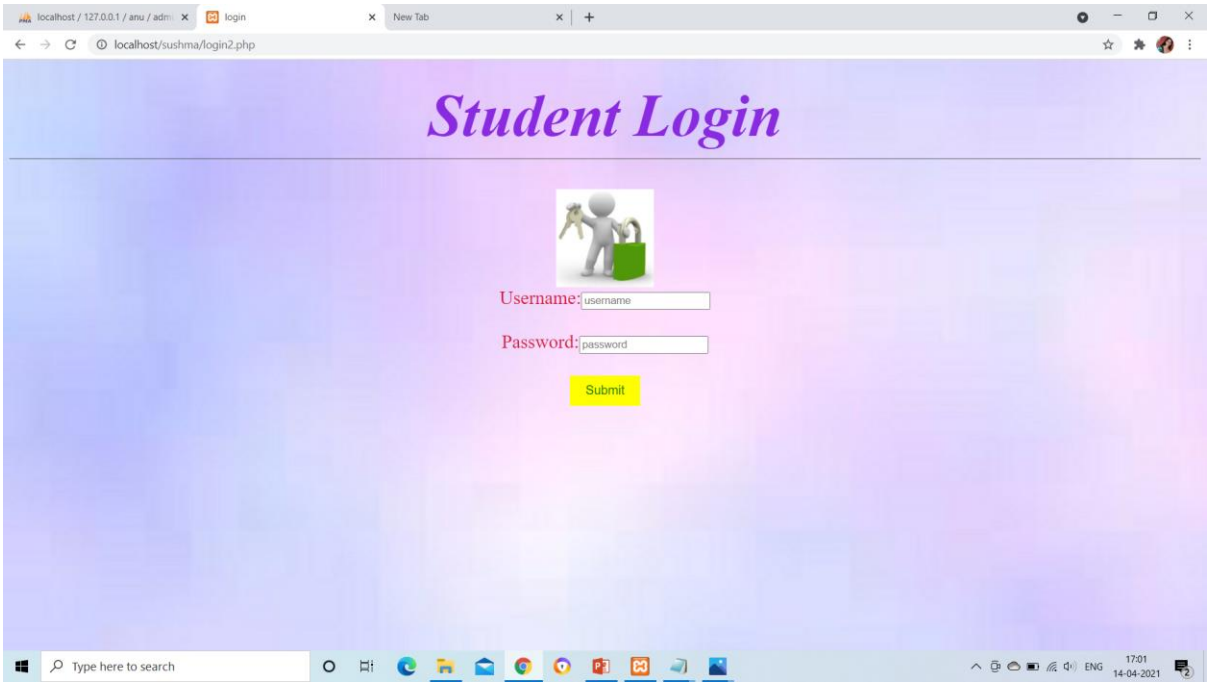
Teacher can view his students' history from Leave History tab:



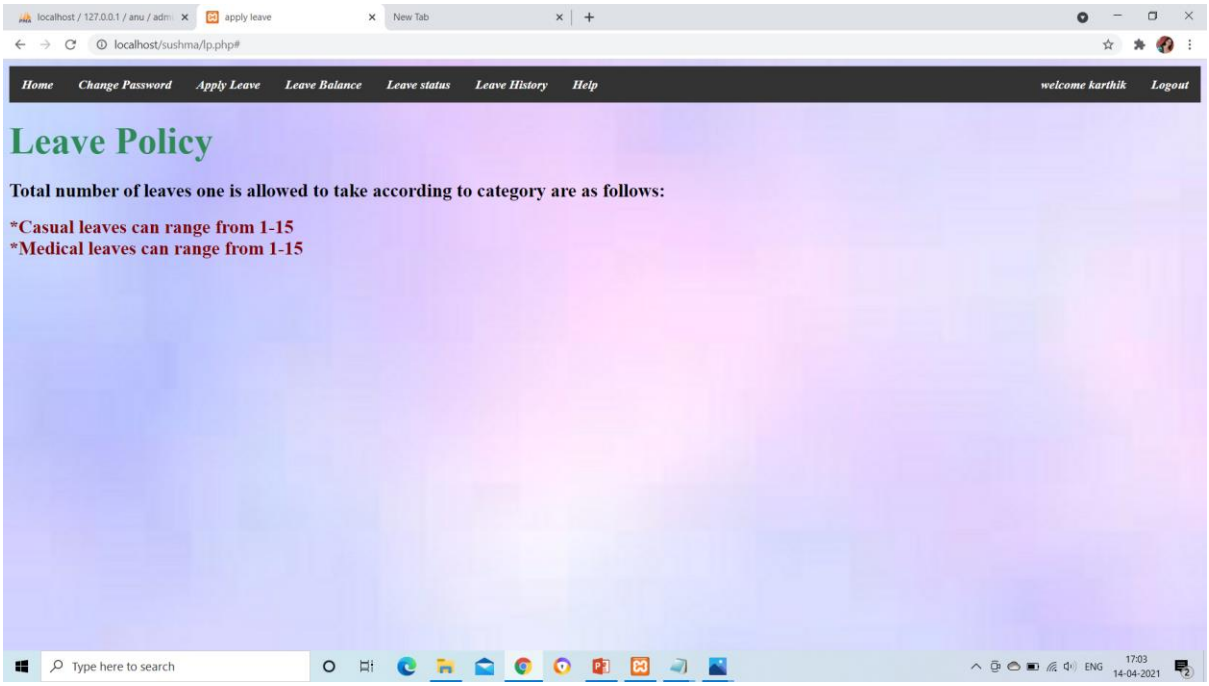
For help regarding the features of leave management system,go to help tab:

Figure 25: User Interfaces of Teacher Login

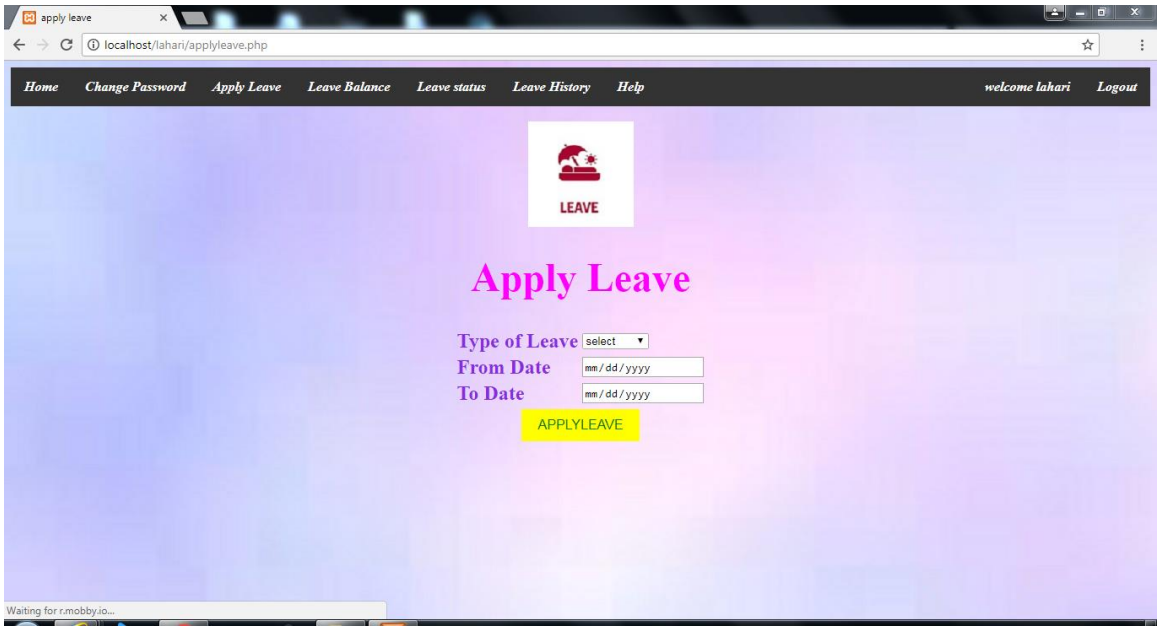
When student Login



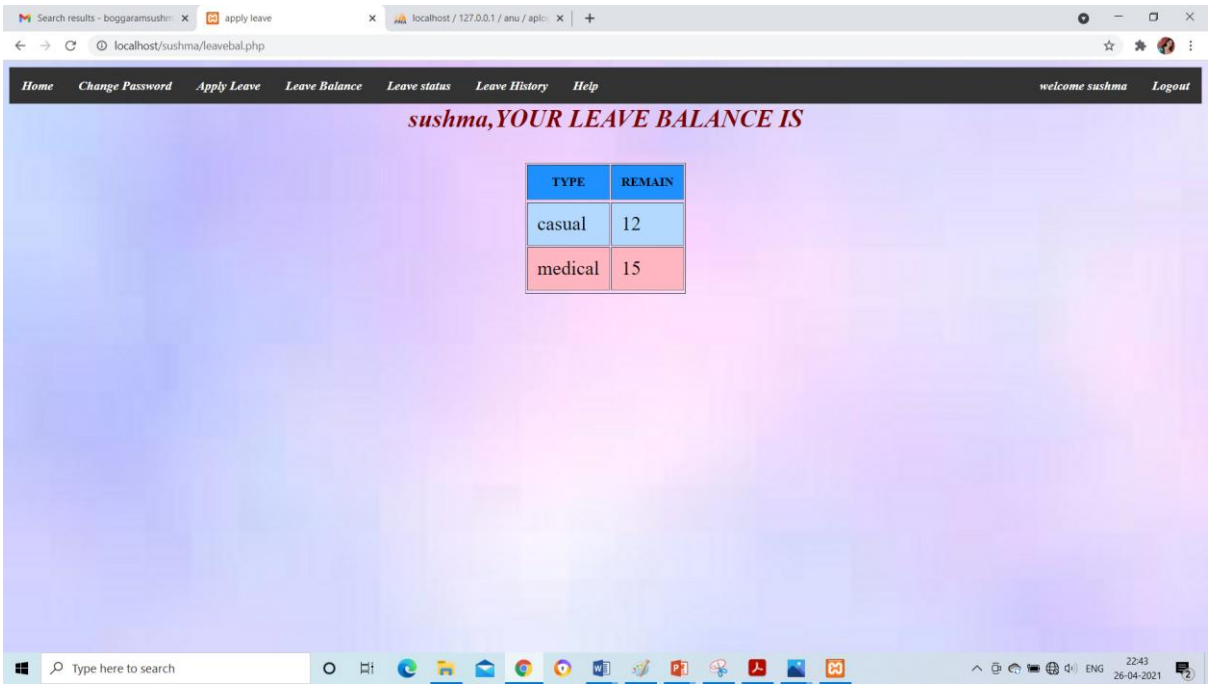
When student logs in:



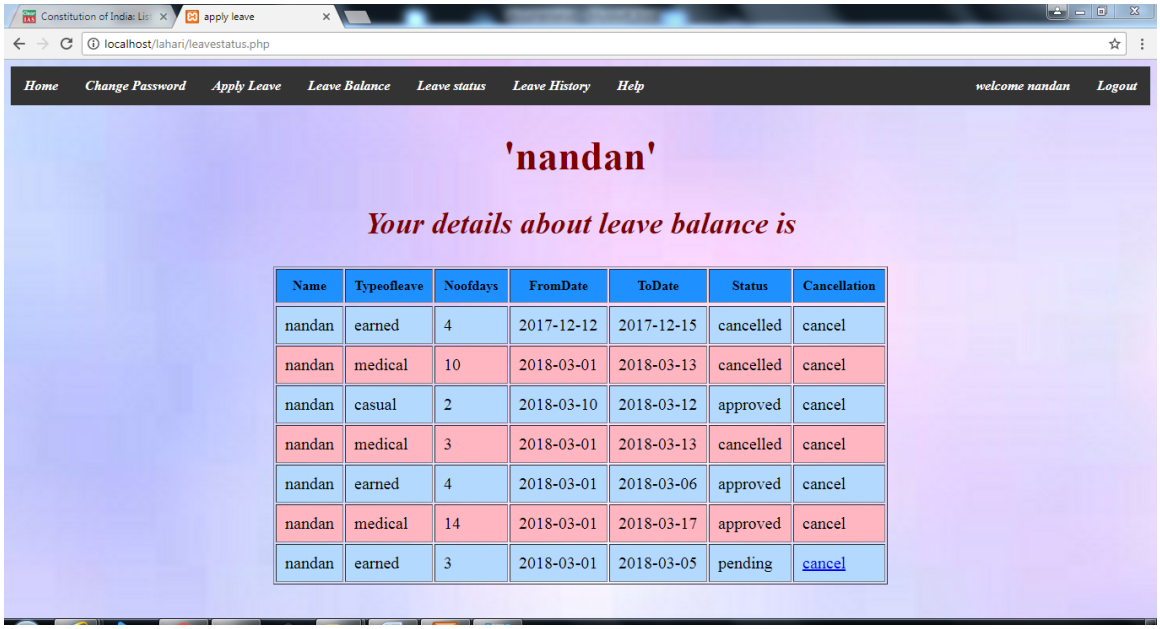
Requestor can apply leave from apply leave tab:



Leave Balance can checked from Leave balance tab:



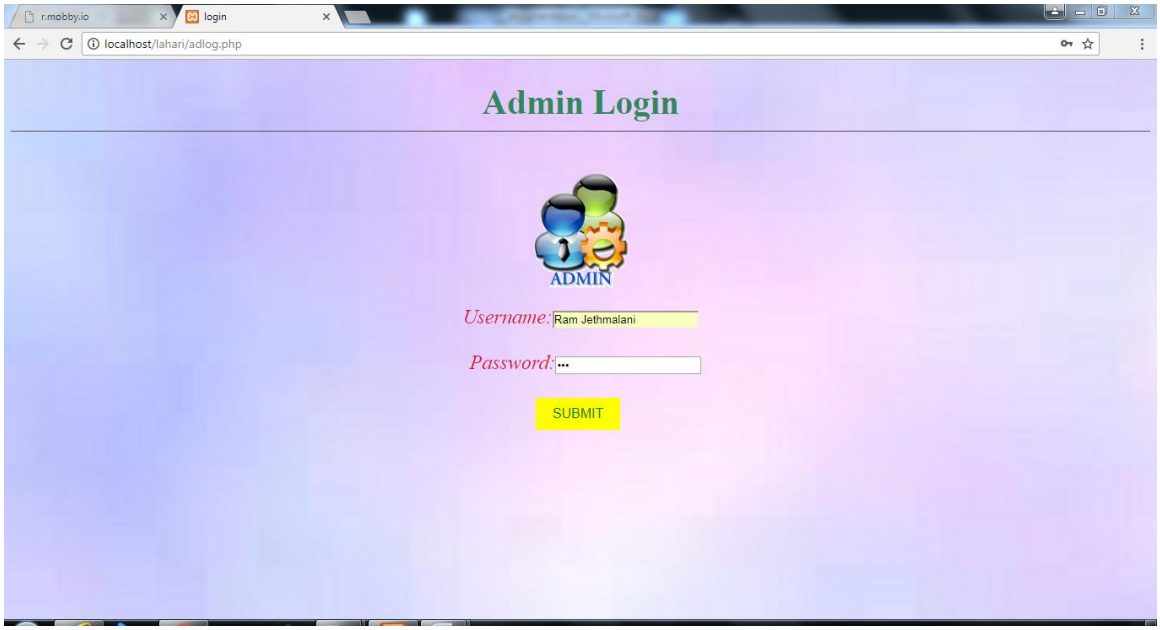
Leave status can be checked from leave status tab:



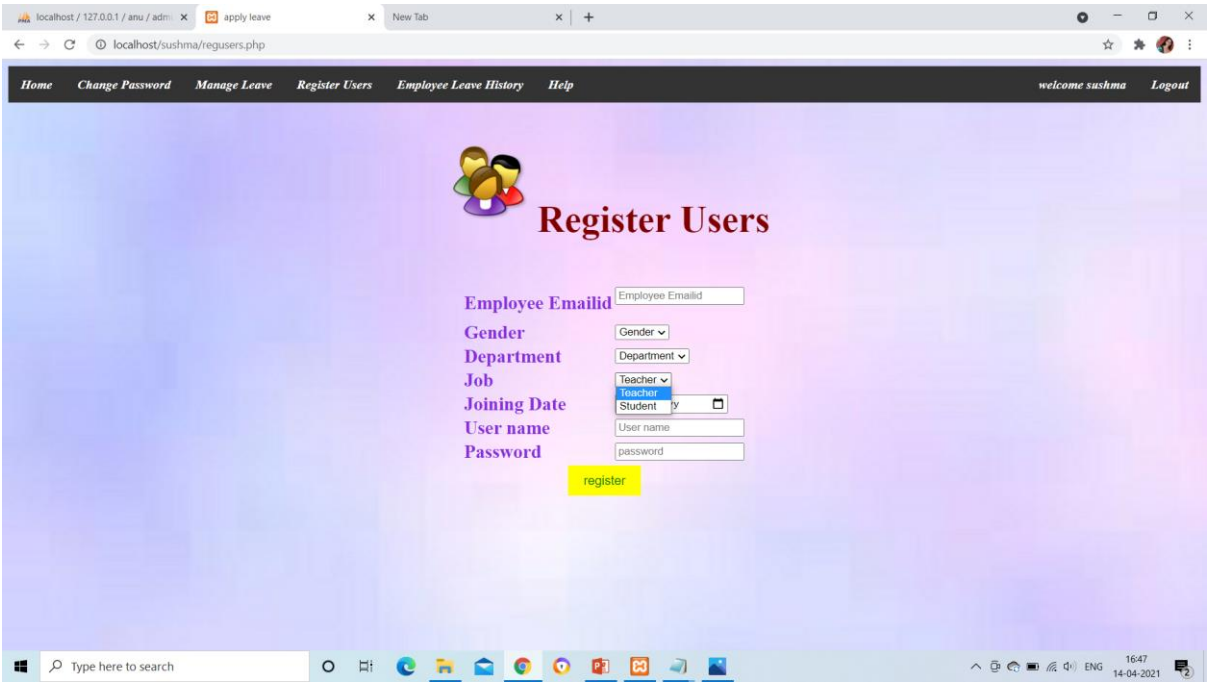
Leave history can be checked from leave history tab:

Figure 26 : User Interfaces of Student Login

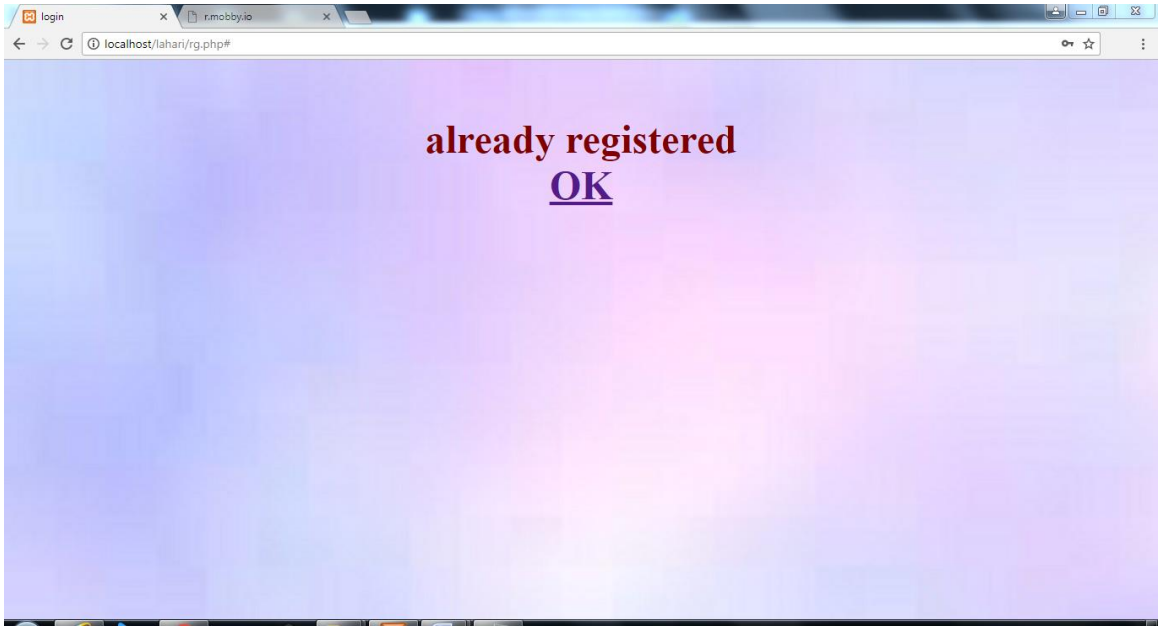
When admin logs in:



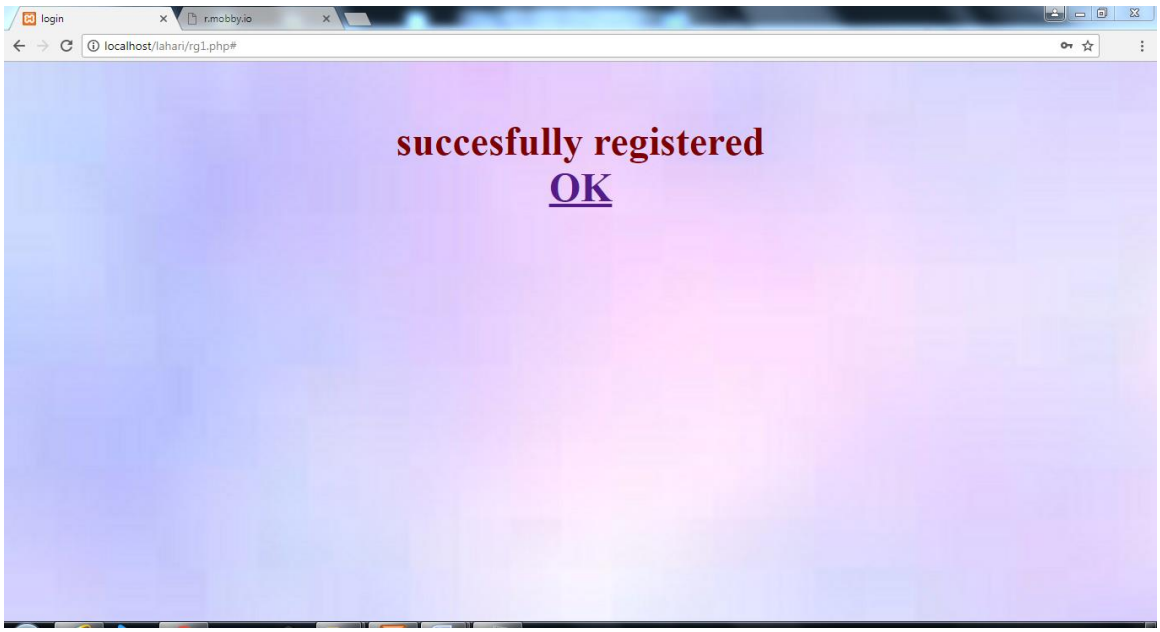
Admin will register users from Register users tab :



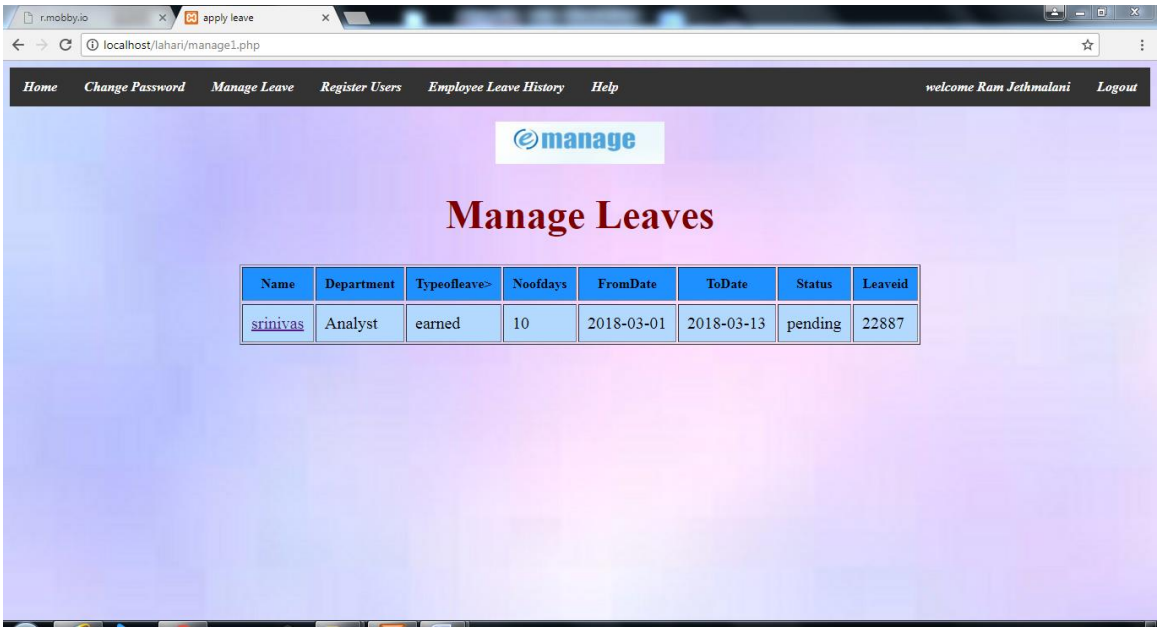
If user is already registered then:



If user is not registered yet then:



Admin can manage leaves that are kept by teacher from manage leave tab:



Admin can view the employees' details from employee leave history tab:

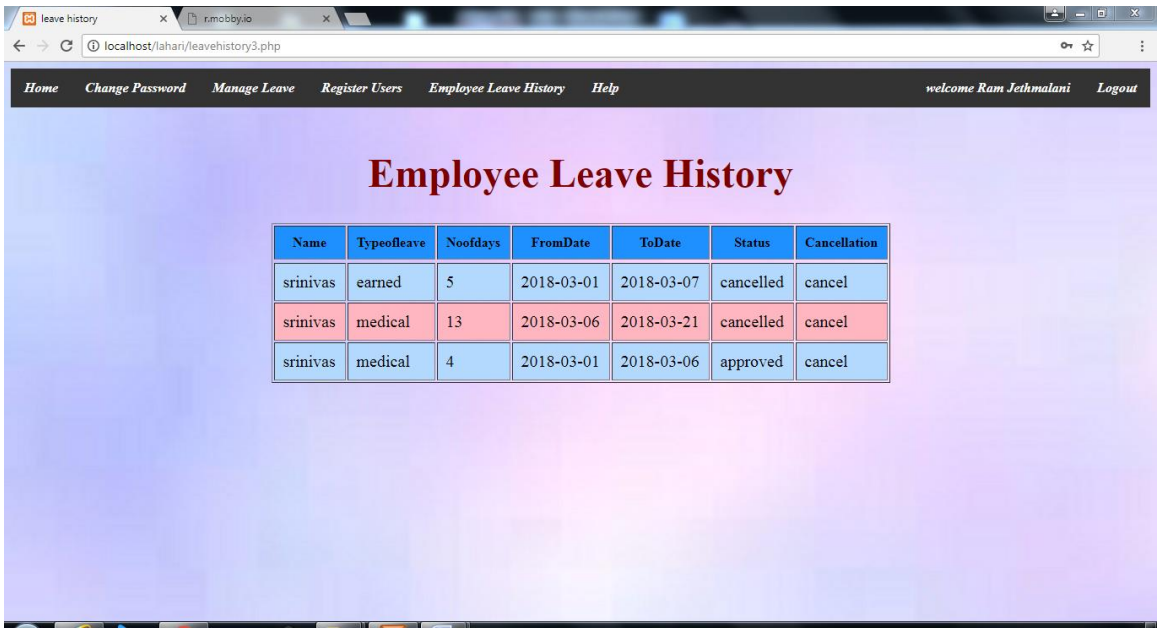


Figure 27: User Interfaces of Admin Login

7

CONCLUSION

Conclusion

The new Leave Management System allows the Users to apply for a leave or cancel a leave and he/she can also view the remaining leaves, the admin can approve or disapprove the leave applied by an employee.

The new system is an improvement on manual leave process in the following ways:

- User can apply leave through online being at home.
- Reduces strain for individuals.
- More gain in profits.
- Issue only with security, but with only few websites.
- Adding maternity leave as a leave type solely for female users.
- Restriction on taking casual leaves during special events

8

REFERENCES

Bibliography

- Object Oriented Analysis and Design 2005 by Mike O'docherty.
- Visual Modeling with Rational Rose 2002 by Teny Quatran.
- Applying UML and patterns: An Introduction to Object Oriented Analysis and Design, Third Edition 2005 by Craig Larman.
- Using UML: Software Engineering with objects and components, Second Edition 2006 by Perdita Stevens, R. J. Pooley.
- The elements of UML style by Scott W. Ambler.
- The Unified Modeling Language User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson.

Web Links:

1. www.w3schools.com/php
2. https://stackoverflow.com