

MINI PROJECT REPORT

ON

Complaint Management System

Submitted in partial fulfillment of requirements to

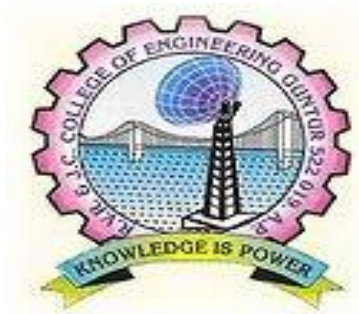
CS 356 - OOAD LAB

By

J.Srilatha(Y16CS852)

G.Thanmay(Y16CS838)

D.Mounika(Y16CS821)



APRIL 2019

R.V.R & J.C.COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Approved by A.I.C.T.E) NAAC A Grade

Chandramoulipuram :: Chowdavaram

GUNTUR – 522 019

R.V.R & J.C.COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that this mini project work titled “**complaint management system**” is done by J.Srilatha(Y16CS852), G.Thanmay (Y16CS838), G.Mounika(y16CS821) in partial fulfillment of the requirements to CS-356, **OOAD Lab** during the academic year 2018-2019.

P.Rama Krishna

Lecturer Incharge

Dr. M.Sreelatha

Prof.&HOD, Dept. of CSE

ACKNOWLEDGEMENTS

*The successful completion of any task would be incomplete without a proper suggestions, guidance and environment. Combination of these 3 factors acts like backbone to our Project “**complaint management system**”.*

*We are very much thankful to **Dr. K.Srinivasu**, principal of **RVR & JC COLLEGE OF ENGINEERING**, Guntur for having allowed delivering this mini project.*

*We express our sincere thanks to **Dr.M.Sreelatha**, Head of the Department of Computer Science and Engineering for her encouragement and support to carry out this mini project successfully.*

*We are very glad to express our special thanks to **P.Rama Krishna**, Lecturer-in charge for the mini project, who has inspired us to select this topic, and also for his valuable advices in preparing this mini project topic.*

*Finally we submit our reserves thanks to Lab staff in **Department of Computer Science and Engineering** . And to all our **friends** for their cooperation during the preparation.*

J.Srilatha (y16cs852)

G.Thanmay(y16cs838)

G.Mounika(y16cs821)

CONTENTS

1. Problem statement	01
----------------------	----

ANALYSIS

2. Requirements elicitation.	02
3. System Requirements Specification	04

Use case view

4. Identification of Actors.	05
5. Identification of Use cases and sub use cases.	08
6. Building requirements model through Use case diagram.	15
7. Flow of Events.	22
8. Prototypes for application	25

9. Activity diagram.	26
----------------------	----

Logical view

10. Identification of Analysis classes.	37
11. Identify the responsibilities of Classes.	38
12. Use case realizations.	40
13. Sequence diagrams.	42
14. Collaboration diagrams	47
15. Identification of attributes and methods of classes.	50
16. Identification of relationships among classes.	51
17. UML Class diagram.	53
18. UML state chart diagram.	57

DESIGN

19. Refining attributes, methods & relationships.	63
20. Implementation diagrams.	
20.a. Component diagrams.	67
20.b. Deployment diagrams	70

20.c.Implementation of domain objects layer and technical service layer	72
20.d. Implementation of user interface layer	78
21. Conclusion	91
22. References	92

1.PROBLEM STATEMENT

Managing complaints is made available by using the concept of web based complaint management system, which includes automated processing system for all the works such as keeping track of complaints and handling complaints which was previously done by using human work power. This complaint management system will be able to work as a platform for escalating complaints. Through this web application, it can be possible to assign and escalate complaints based on date of registration of a complaint.

In case of complaint management, assigning and escalation process will be carried out directly by the system. In charge is responsible for resolving the complaint. Admin will be able to see the candidates who registered the complaint and he is responsible for assigning and escalating complaints. Through this system complaints can be easily handled and managed as it includes automatic escalation of complaints to various employees in the organization “SELCO” with respect to date of registration of complaint.

ANALYSIS

2.REQUIREMENT ELICITATION:

RID	REQUIREMENTS	REQUIREMENT NATURE	Moscow
R1	Intuitive graphical user interface to be offered	Functional	Should Have
R2	User should be login first	Functional	Must Have
R3	User should register a complaint through username and password	Functional	Must Have
R4	User can register a complaint directly to SELCO branches through phone calls or emails	Functional	Could Have
R5	Admin should login	Functional	Must Have
R6	Admin should maintain records and track complaints	Functional	Must Have
R7	Error messages can be displayed for invalid login	Functional	Should Have
R8	Admin should update a	Functional	Must Have

	complaint in SMCR (SELCO Master Complaint Register)		
R9	Only authorized person i.e., admin should modify or edit the SMCR	Functional	Must Have
R10	Only admin can update the database	Functional	Want to
R11	Admin allocate a unique complaint number	Functional	Should Have
R12	In charge should login	Functional	Must Have
R13	In charge should send the complaint to respective branch	Functional	Must Have
R14	In charge should update the complaint in OCR (Open Complaint Register)	Functional	Must Have
R15	OCR can be edit by in charge to update the result of complaint	Functional	Must Have
R16	Admin can view the OCR to update the database whether the complaint is closed or not	Functional	Must Have
R17	In charge can view the SMCR for current status of the complaint	Functional	Should Have
R18	Complaint can be resolved in specified number of days	Functional	Should Have

	otherwise it should be put forth to escalation process		
R19	Database interface should be provided	Functional	Must Have

3. System Requirements Specification :

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 : PHP

Database : My SQL Database

USE-CASE VIEW

4. 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

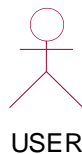
Actors are discovered by examining:

- 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.

GRAPICAL DEPICTION:

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



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 Complaint Management System are:

- 1.User
- 2.Admin
- 3.Incharge

Brief description of actors:

User:

Actor	Brief Description
User	The main objective of this user is to register a complaint. The user who used to register a complaint will login through username and password. Then he registers a complaint. He can able to view the result of the complaint whether it is solved or not.

Admin:

Actor	Brief Description
Admin	He maintains records and tracks registered complaints. Once the complaint is registered the admin update in the "SELCO Master Complain Register (SMCR)", an online google document. Every complaint is allocated a unique "complaint number" by admin.. He is given sole authority to edit, modify and update the SMCR, to ensure the complaints are closed only when they are resolved. He verifies the information provided by the branch in the OCR and dates the same in the SMCR.

In charge:

Actor	Brief Description
In charge	Once the complaint is addressed the in charge updates the same in the “Open Complaint Register (OCR)”, an online google document. The OCR can be edited by the In charge (through an online data base) to input the details of the lodged complaints.

5. IDENTIFICATION OF USE-CASES AND SUB USE-CASES

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 use case 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 use case 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. Use case 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 on details.
- 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 value 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.

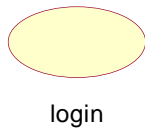
Use-cases identified for our system are:

- Login
- Register complaint
- Check complaint status
- View SMCR
- View OCR
- Update OCR
- Manage complaints
- Update SMCR

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

Use case name	Brief Description
Login	The user who wants to register a complaint must login and register a complaint and to check the status of the complaint.

UML Notation:



2. Use-case name: **Register complaint**

Use case name	Brief Description
Register complaint	The user register a complaint

UML Notation:



Register complaint

3.Use-case name: **Check Complaint Status**

Use case name	Brief Description
Check Complaint Status	The user checks the status of a complaint

UML Notation:

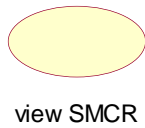


check complaint status

4.Use-case name: View SMCR

Use case name	Brief Description
View SMCR	Admin update the complaints in SMCR

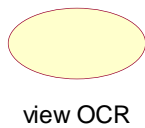
UML Notation:



5.Use-case name: View OCR

Use case name	Brief Description
View OCR	Admin verify the information provided by the in charge and update same in SMCR and also in charge update the complaint in OCR when the complaint is addressed.

UML Notation:



6.Use-case name: Update OCR

Use case name	Brief Description
Update OCR	In charge update the complaints in OCR

UML Notation:



update OCR

7.Use-case name: Manage complaints

Use case name	Brief Description
Manage complaints	Admin manages the complaint

UML Notation:



Manage complaints

8.Use-case name: Update SMCR

Use case name	Brief Description
Update SMCR	Admin update the complaints in SMCR Whether it is solved or not

UML Notation:



update SMCR

9. Use-case name: Assign complaints

Use case name	Brief Description
Assign complaints	Admin assigns the complaint

UML Notation:



assign complaints

6. 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:

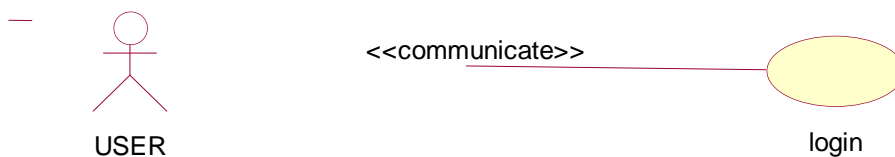
If you prefer, you can also customize the toolbox to include the bi-directional tool to the use-case toolbox.

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:

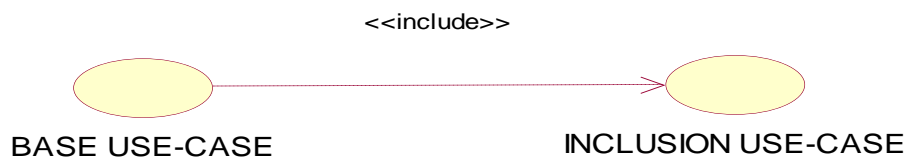
A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning. Typically, on class diagrams, a dependency relationship indicates that the operations 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 modeled as dependencies by using the Extend stereotype.

An *extend* relationship is used to show

- Optional behavior
- 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:

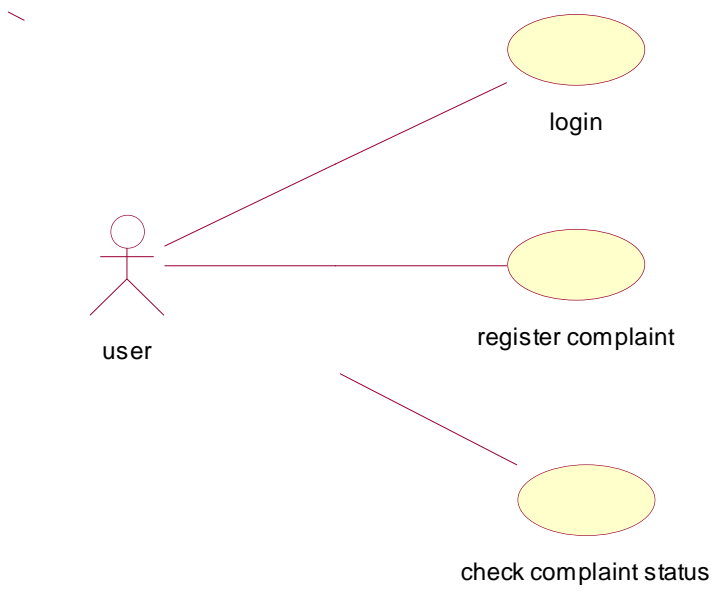
- An extend relationship shows optional functionality or system behavior.
- A base use case does not need to acknowledge any specific extended use cases

Finally, we can conclude,

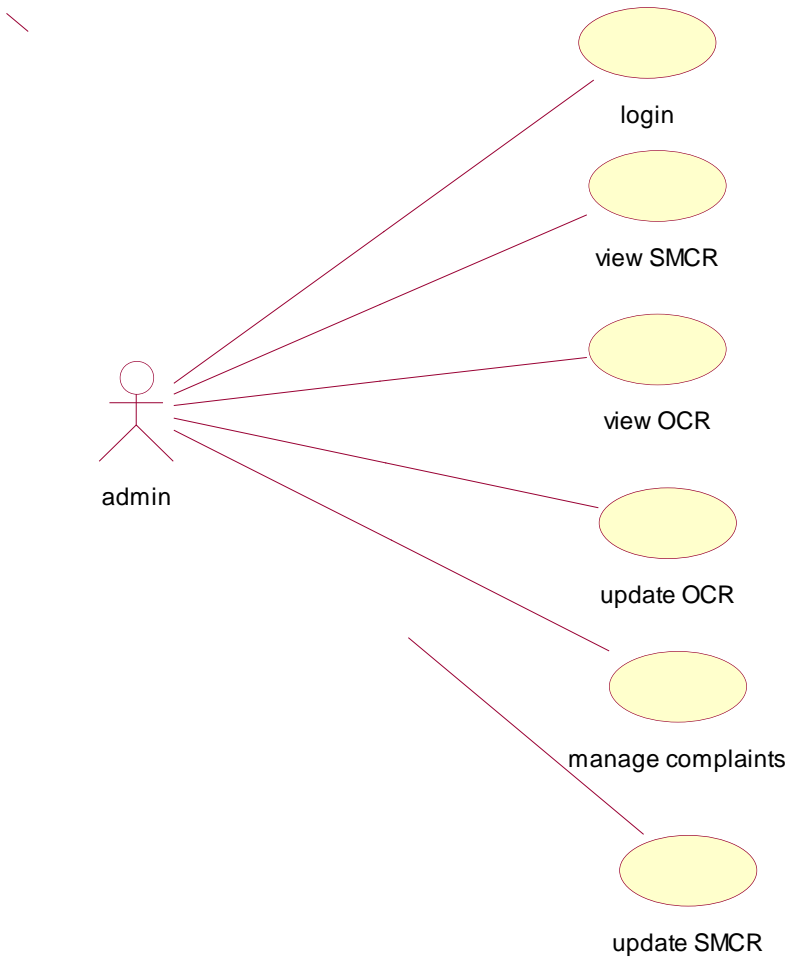
- **«extend»** is used when you wish to show that a use case provides additional functionality that may be required in another use case.
- **«include»** applies when there is a sequence of behavior that is used frequently in a number of use cases, and you want to avoid copying the same description of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

.EX: Basic use case diagram for Complaint Management System:

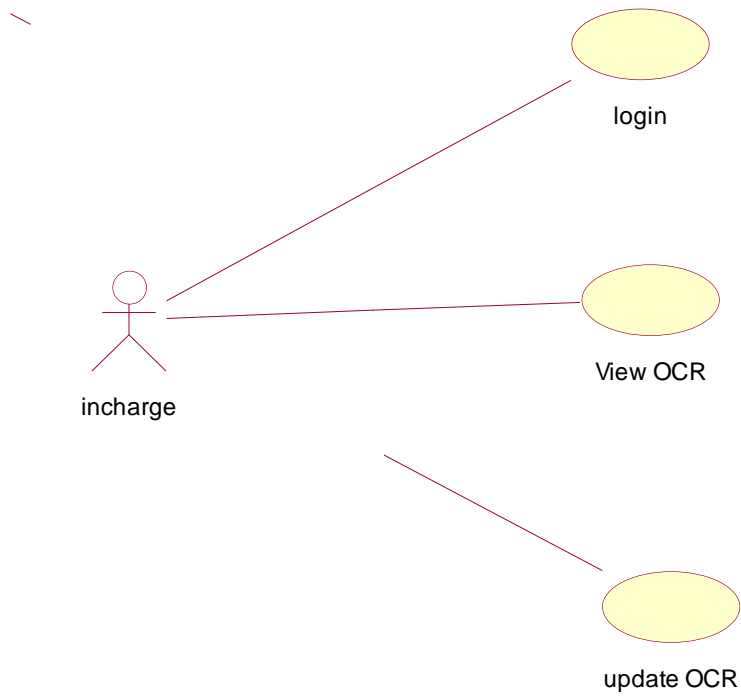
User:



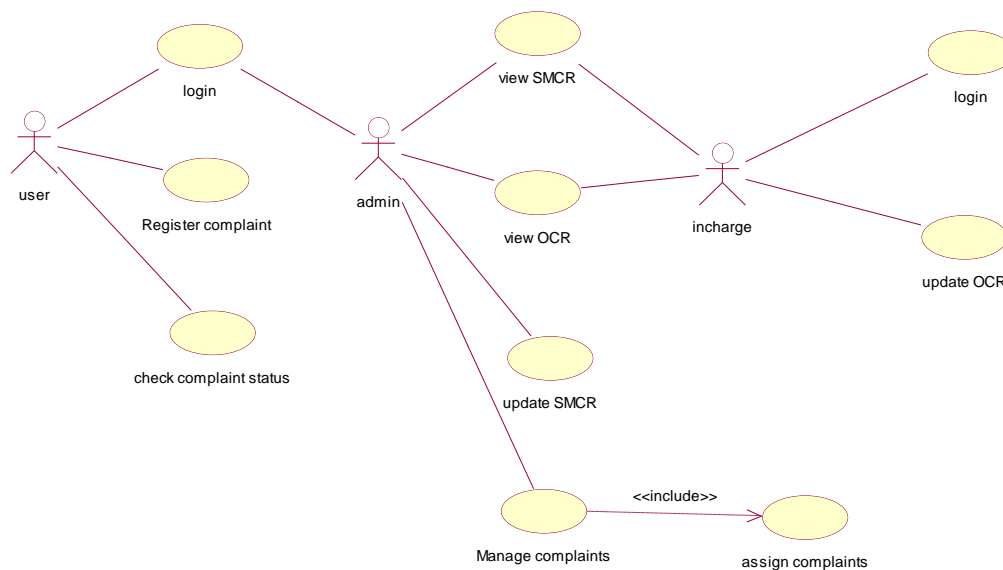
Admin:



Incharge:



EX:The use case diagram with stereotypes:



7. 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 events 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 starts 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
 - Post-Conditions
 - Extension Points

Use case:	Login/Logoff
Id:	1
Brief description:	This use case describes how a user logs into the Complaint Management System. The actors starting this use case are customer, In charge and the administrator of the system

Primary actor	customer, Administrator
Secondary actor	In charge
Pre condition:	The actor must be a selco member
Flow of events:	<ol style="list-style-type: none"> 1. The system validates the actor's password and logs him/her into the system. 2. The system displays the Main Form and the use case ends.
Post condition:	None
Alternate flow:	<p>1.Invalid Name / Password</p> <p>If in the basic flow the system cannot find the name or the password is invalid, an error message is displayed. The actor can type in a new name or password or choose to cancel the operation, at which point the use case ends</p>

Use case:	Updating of Complaints
Id:	2
Brief description:	The Admin can choose either to assign complaints to in charge or update SMCR or view OCR.
Primary actor	Administrator
Secondary actor	None
Pre condition:	The Admin login to the Complaint Management system so that he or

	she can do the updating of complaints.
Flow of events:	1. Admin logs into the Complaint Management system. 2. Admin updates complaints in the forms.
Post condition:	The status is set to completed only if it is closed in SMCR .
Alternate flow:	None

8. 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 *typhus*, meaning *form or model*. In a non-technical context, a prototype is an especially representative example of a given category.

9. 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:

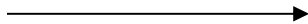
Activity diagrams contain activities, transitions between the activities, decision points, and synchronization bars. An activity represents the performance of some behavior in the workflow. In the UML, activities are represented as rectangles with rounded edges, transitions are drawn as directed arrows,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



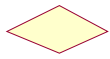
TRANSITIONS:

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 to 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

SWIMLANES:

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.

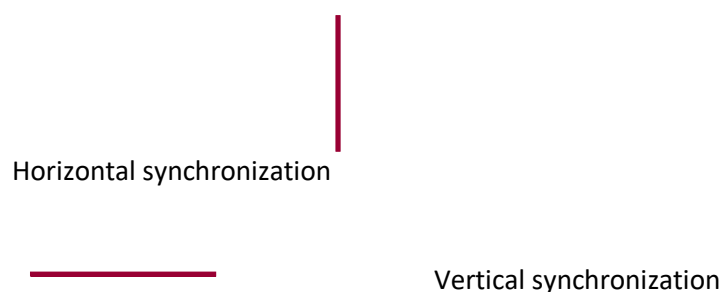
When a swim lane is dragged onto an activity diagram, it becomes a swim lane view. Swim lanes appear as small icons in the browse while swim lane views appear between the thin, vertical lines with a header that can be renamed and relocated.

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.



The activity diagram for overall system is:

- At first the system has to provide authentication to the users through the registration process.
- After providing the authentication the system and admin have to maintain their information.
- Then user will login into their homepage through username and password.
- Then the user will register a complaint.
- Admin can be able to see the list of complaints registered by the user and maintain the records and track the complaint
- After logging in user register a complaint.
- After that the user can view the status of the complaint and logout.
- Once the complaint is registered Admin update the complaint in the “SELCO Master Complain Register (SMCR)”, an online google document.
- The Admin is given sole authority to edit, modify and update the SMCR, to ensure the complaints are closed only when they are resolved.
- Admin allocate a unique complaint number to a complaint.
- Admin maintains the status of the complaint in SMCR.
- In charge updates the complaints in “Open Complaint Register (OCR)”, an online google document.
- The OCR can be edited by the in charge (through an online data base) to input the details of the lodged complaints
- Admin verifies the OCR and update the same status in SMCR
- Admin logout from the page.

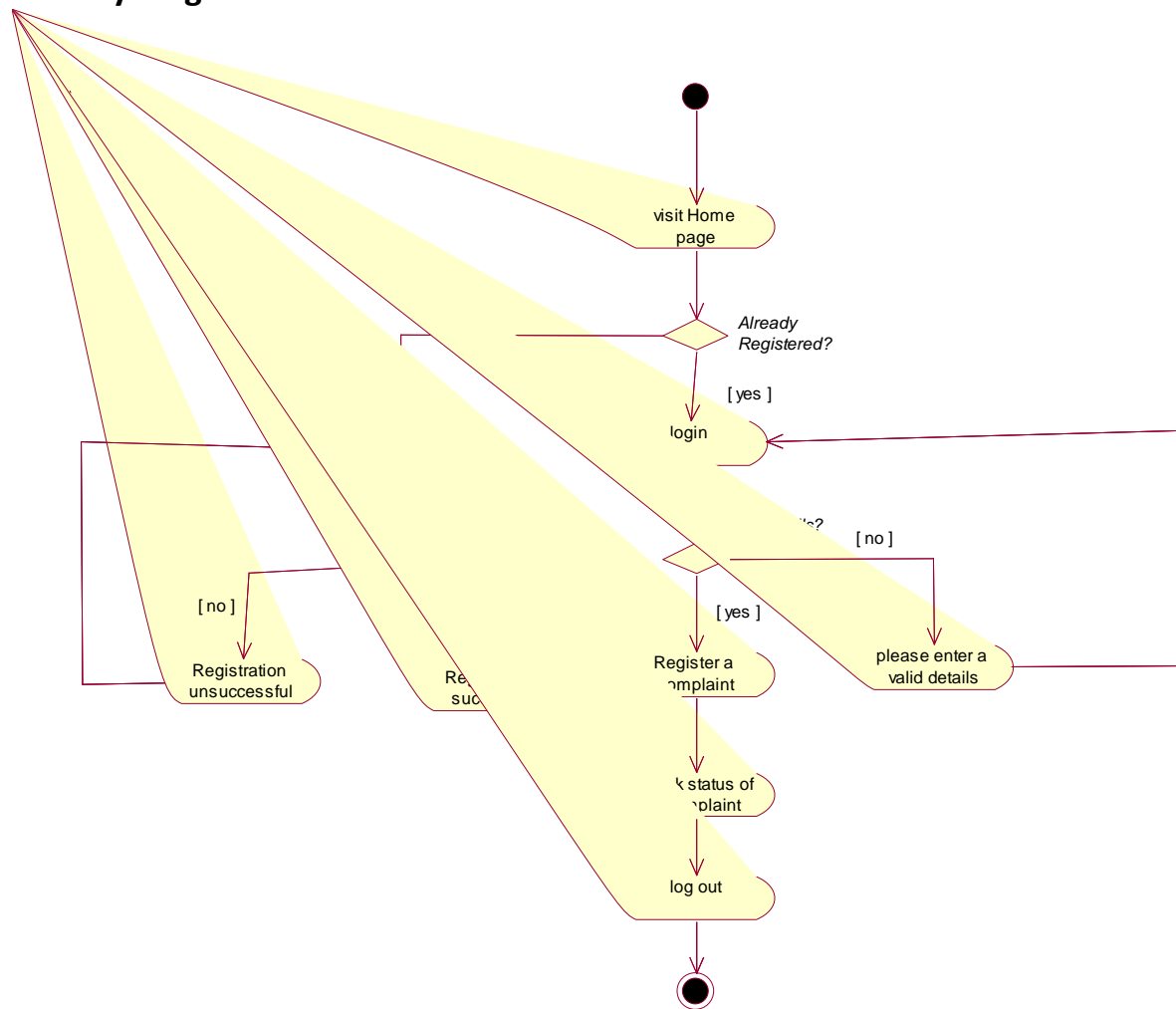
Modeling a workflow in an activity diagram can be done several ways; however, the following steps present just one logical process:

- 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

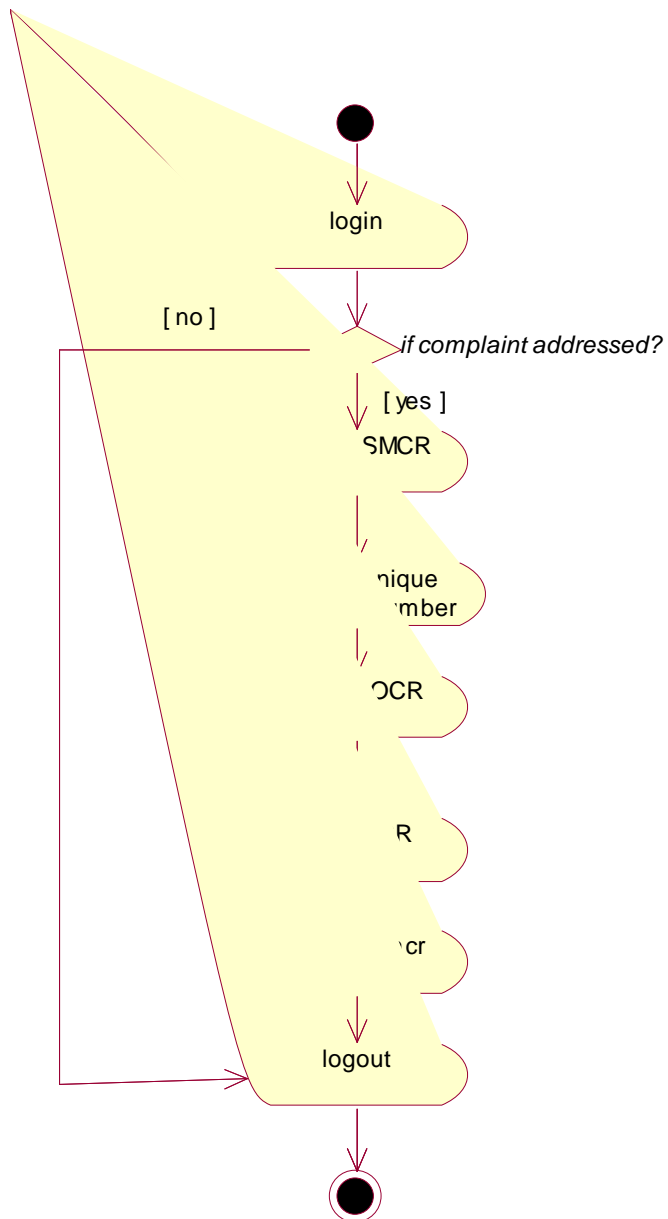
designate the beginning and ending of the workflow. State and end states clarify the perimeter of the workflow.

- 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.

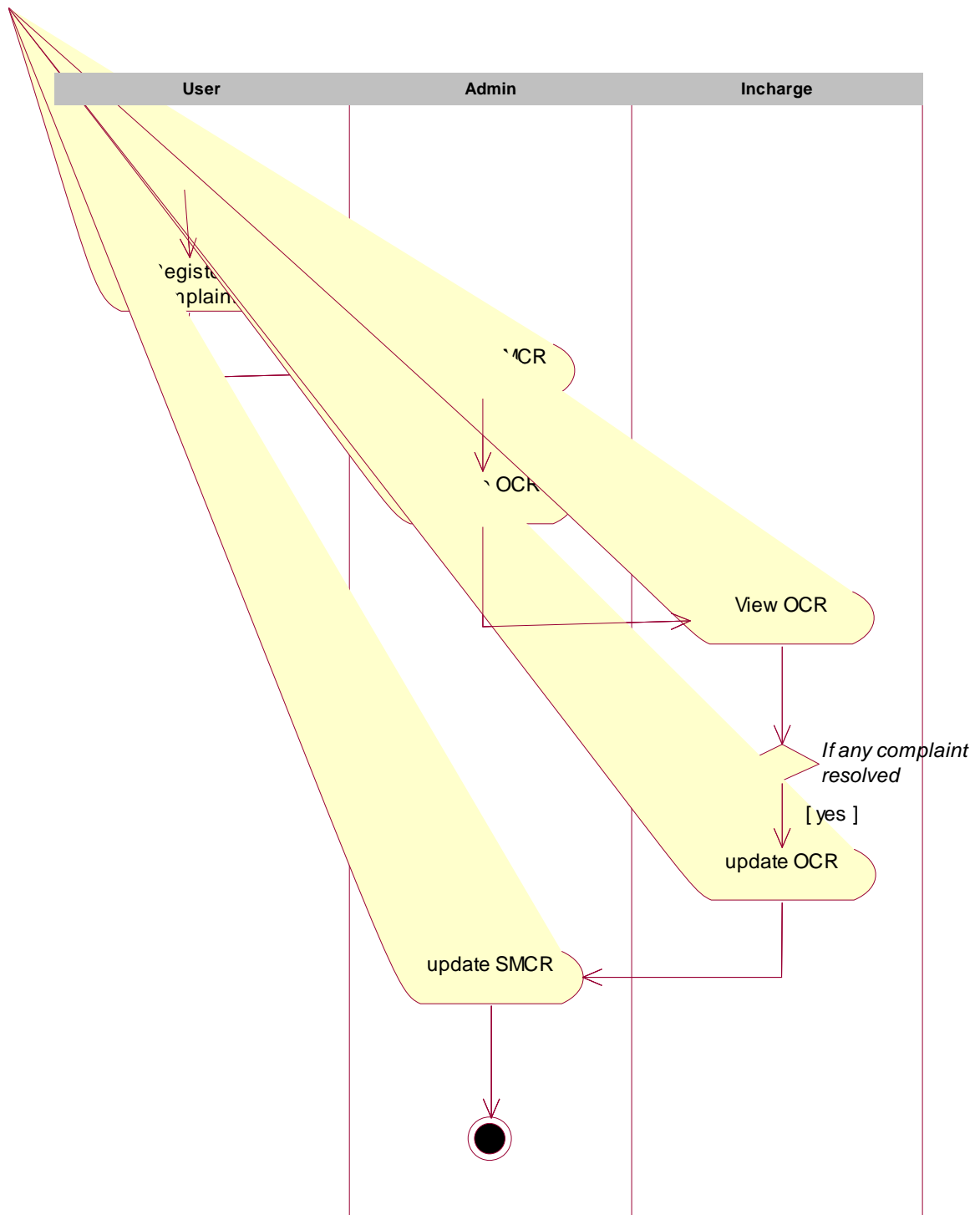
Activity diagram for user:



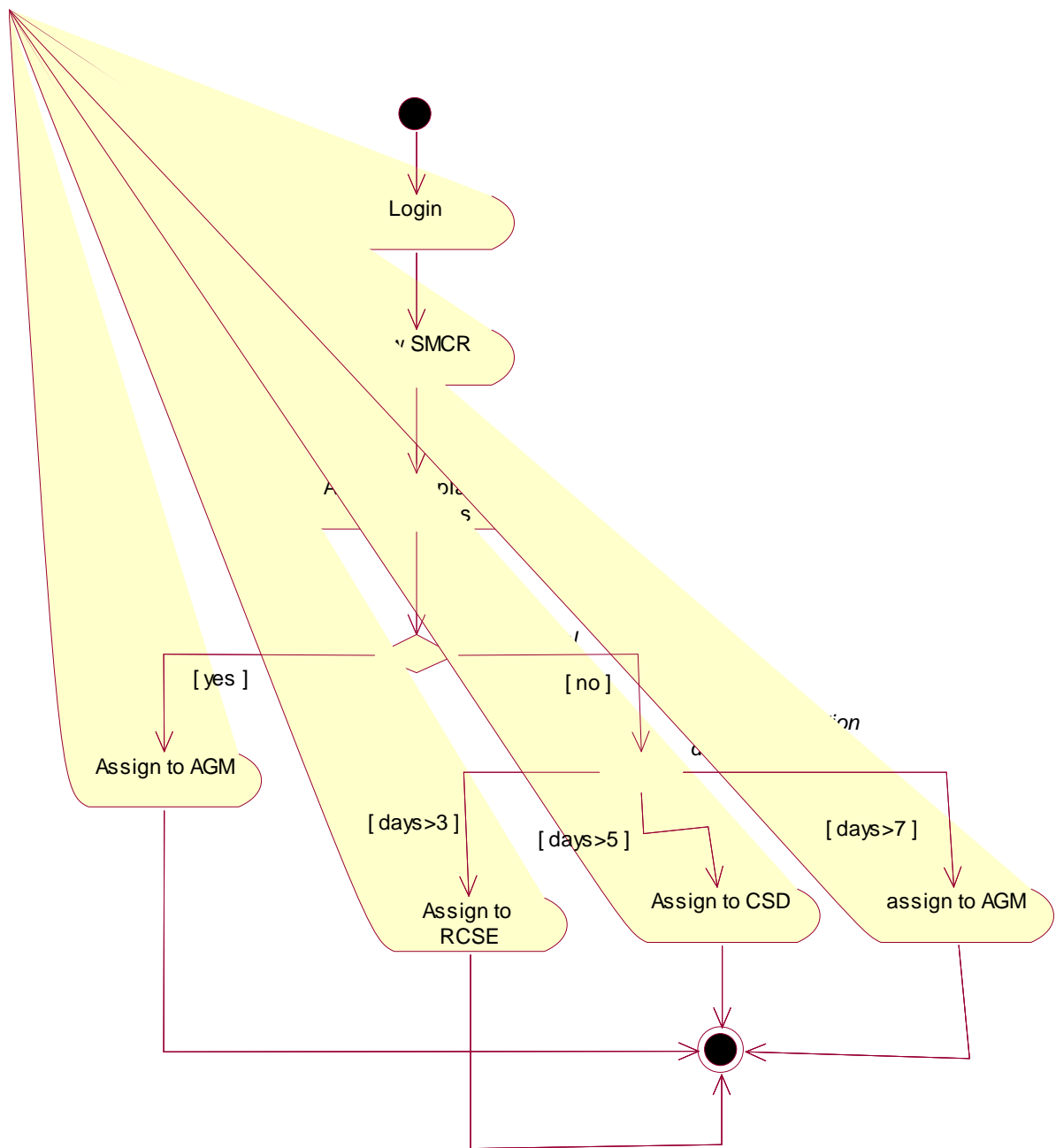
Activity diagram for Admin:



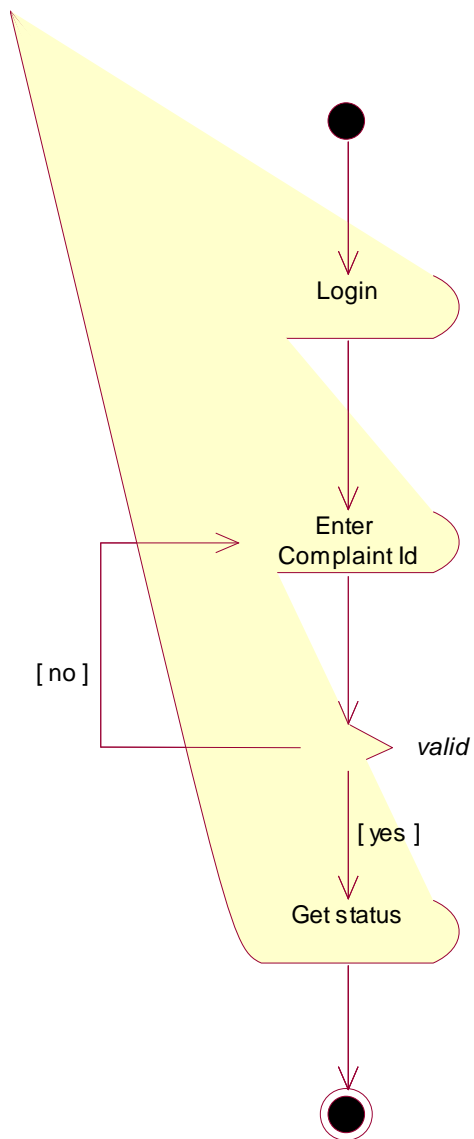
ACTIVITY DIAGRAM FOR CMS:



ACTIVITY DIAGRAM FOR MANAGE COMPLAINTS:



ACTIVITY DIAGRAM FOR CHECK COMPLAINT STATUS:



LOGICAL VIEW

10. IDENTIFICATION OF ANALYSIS CLASSES

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

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.

1. 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.

2. 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 patterns 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.

11. IDENTIFICATION OF RESPONSIBILITIES 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 to 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.

12. 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.

13. 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 behavior

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 objects 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.

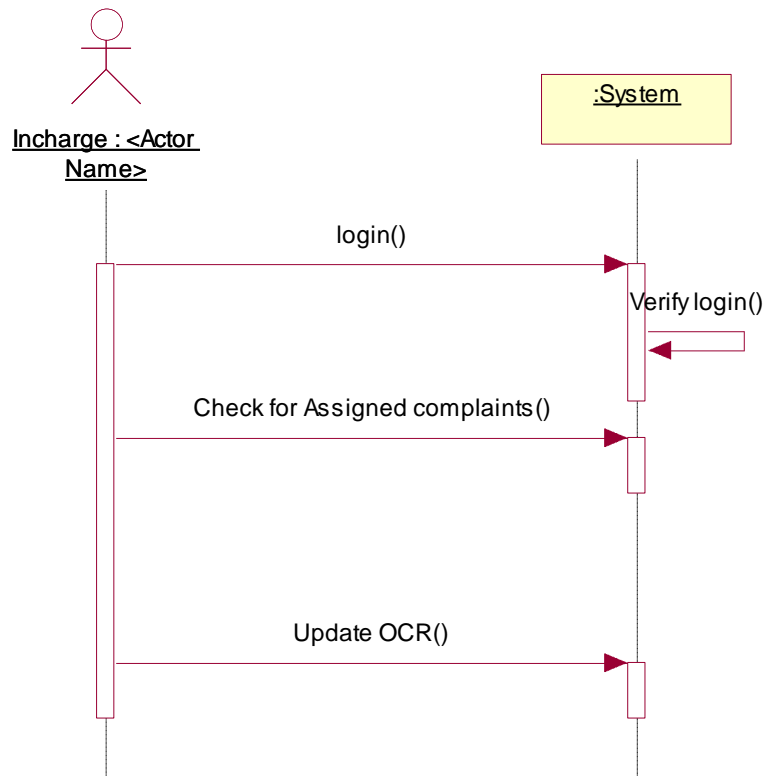
Message to self: A message to self is a tool that sends a message from one object back to the same object. It does not involve other objects because the message returns to the same object. The sender of a message is the same as the receiver.

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.

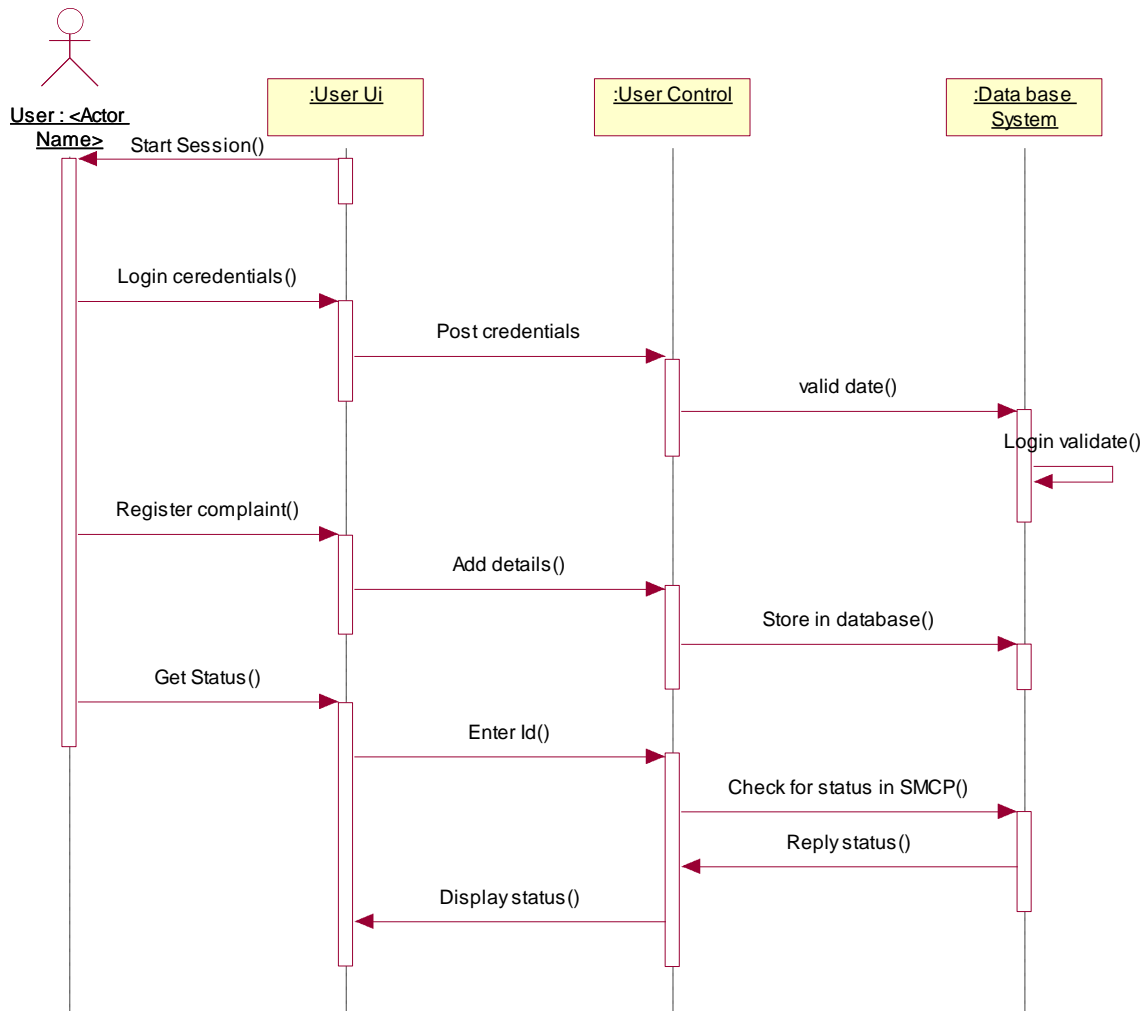
Note

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

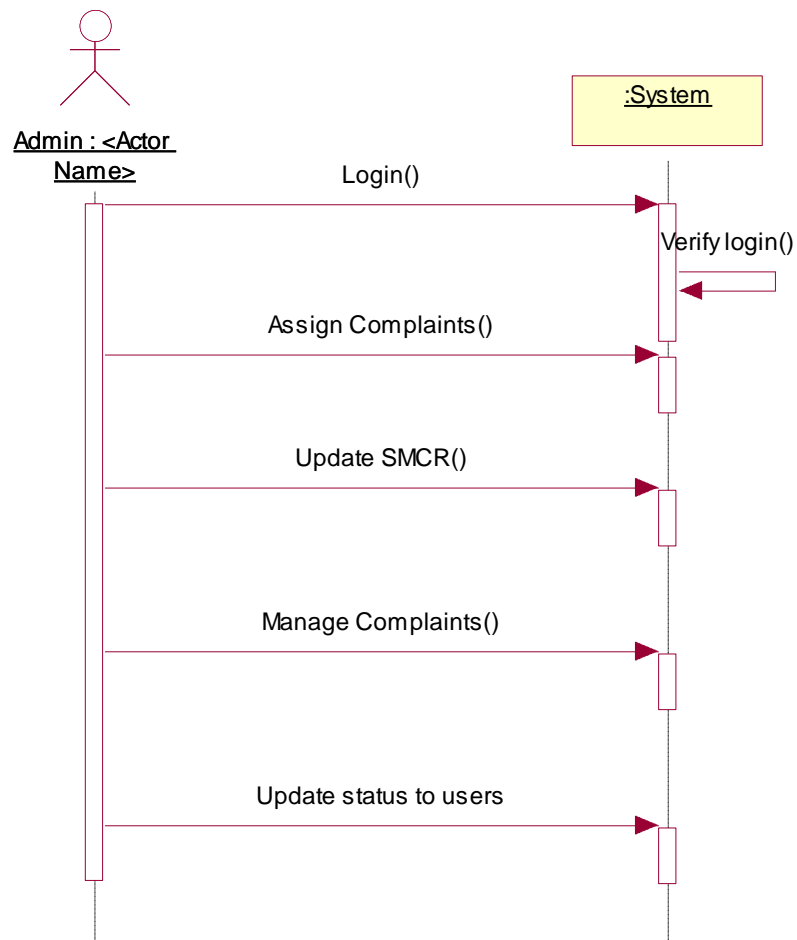
Sequence diagram for In charge:



Sequence diagram for User:



Sequence diagram for Admin:



14. COLLOBORATION DIAGRAM

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.

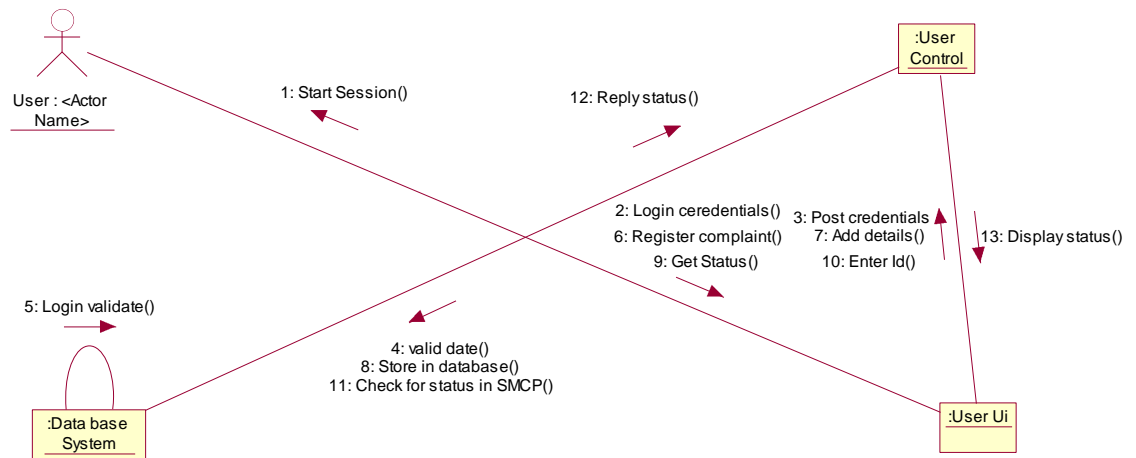
Each collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object like entities in the current model.

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

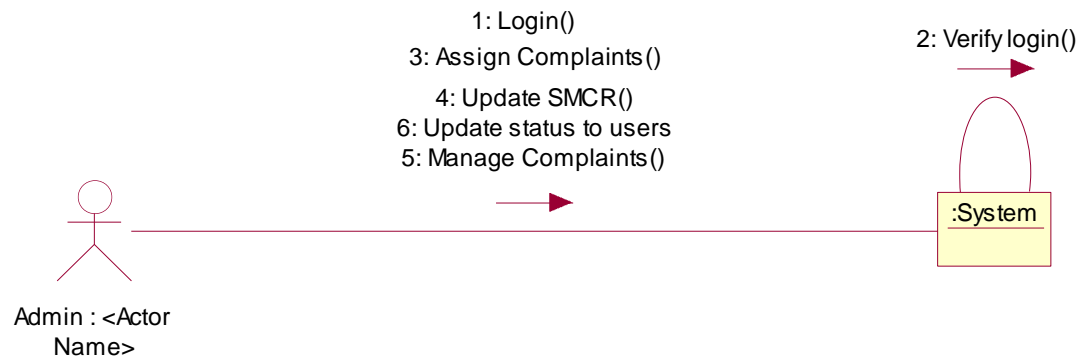
- Objects drawn as rectangles
- Links between objects shown as lines connecting the linked objects
- Messages shown as text and an arrow that points from the client to the supplier

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

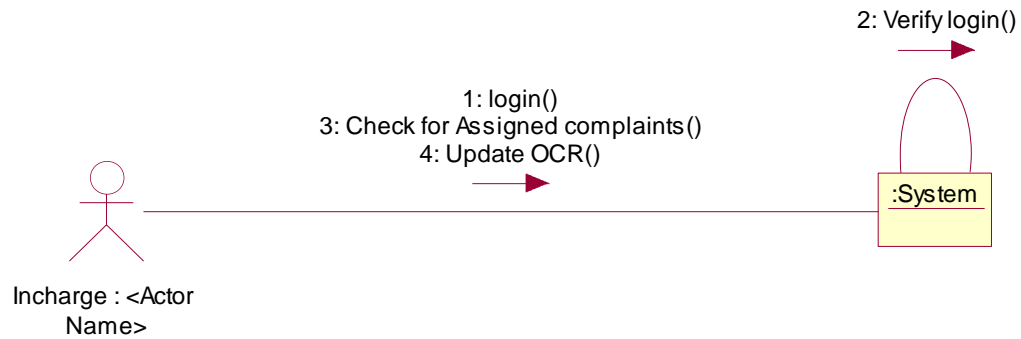
Collaboration diagram for User:



Collaboration diagram for Admin:



Collaboration diagram for Incharge:



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 shows object interaction in a time-based sequence.
- A collaboration diagram shows object interactions organized around the objects and their links to each other.

15 IDENTIFICATION OF METHODS AND ATTRIBUTES OF CLASSES

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 User: User Name, Address, MobileNumber, Password
- Attributes for admin: User Name, Password
- Attributes for Incharge: UserName, Designation, Password

The responsibilities identified in our system are:

- Methods for administrator:
getRegisteredComplaints(),assigncomplaints(),updatesOCR(),updates SMCR(),escalates complaints(),return status(), logout().
- Methods for User: registerComplaint(),getStatus() , logout().
- Methods forIncharge: getAssignedComplaints(),updatesOCR(),logout().

16. IDENTIFICATION OF RELATIONSHIPS AMONG CLASSES

NEED FOR RELATIONSHIPS AMONG CLASSES:

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.

17. 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.

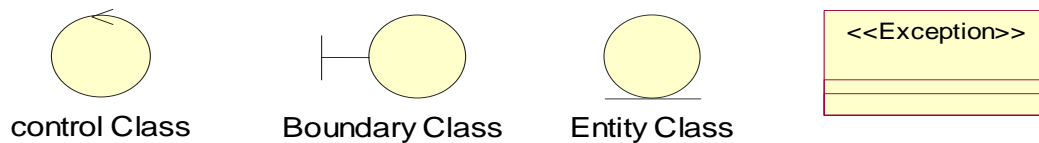
In the UML, classes are represented as compartmentalized rectangles.

- 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 case.

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.

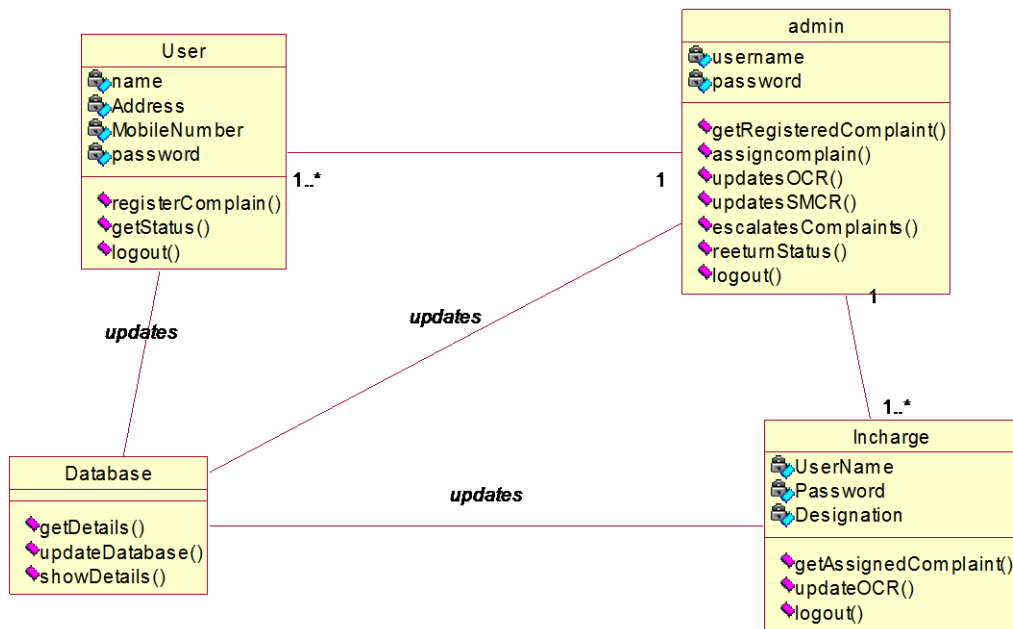
For example, a Student class could be documented as follows: *A student is someone currently registered to take classes at the university and this information will be provided for billing purpose.*

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 Complaint Mangement System:



18. 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.

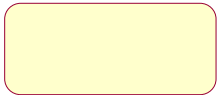
A state chart diagram shows the states of a single objects, the events or messages that cause a transition from one state to another and the actions that result from a state change. As I activity diagram, state chart diagram also contains special symbols for start state and stop state.

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 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)`

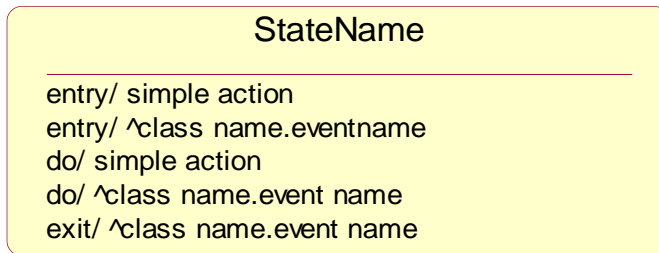
Only one event is allowed per transition, and one action per event.

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;



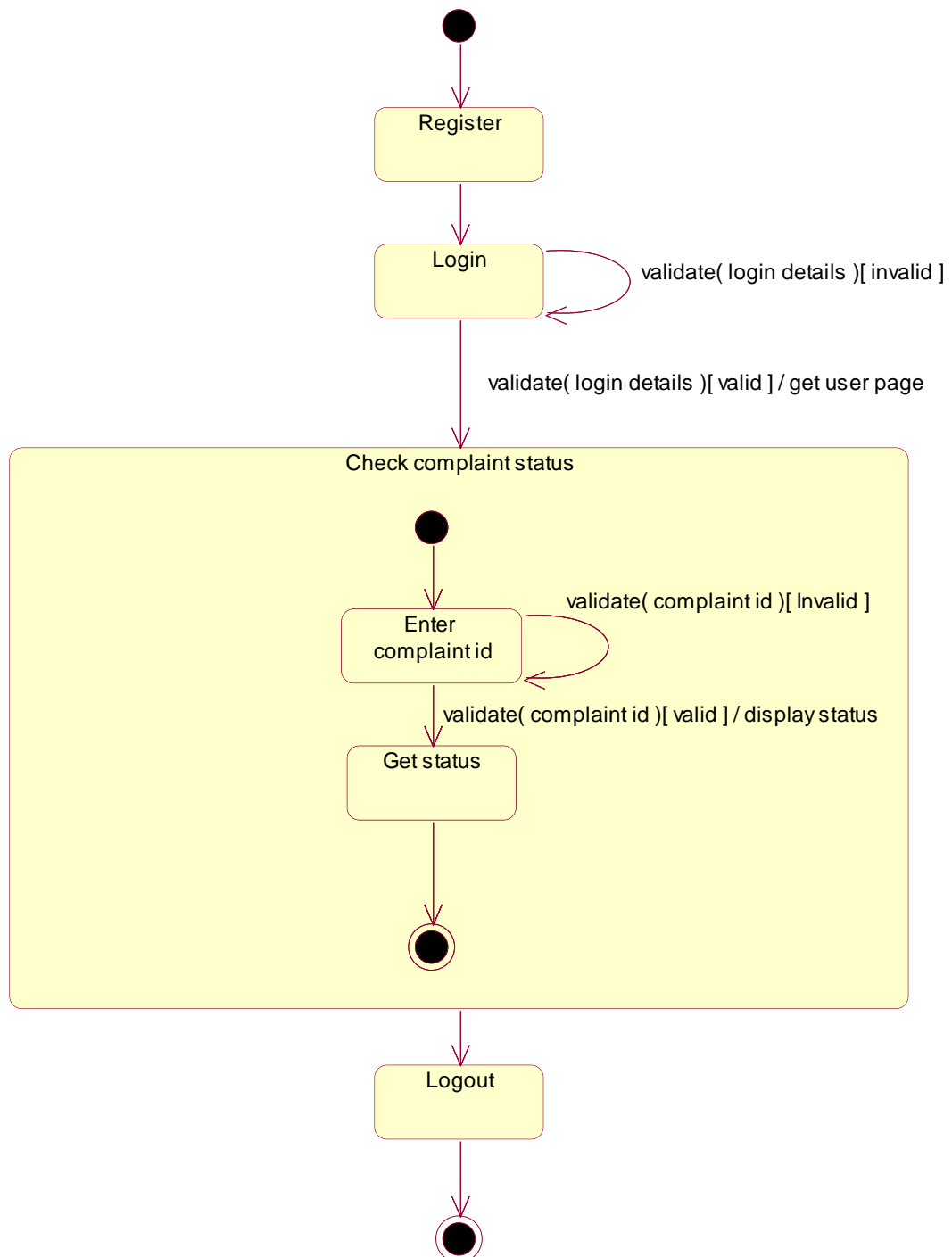
Purpose of state chart diagrams:

- 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.

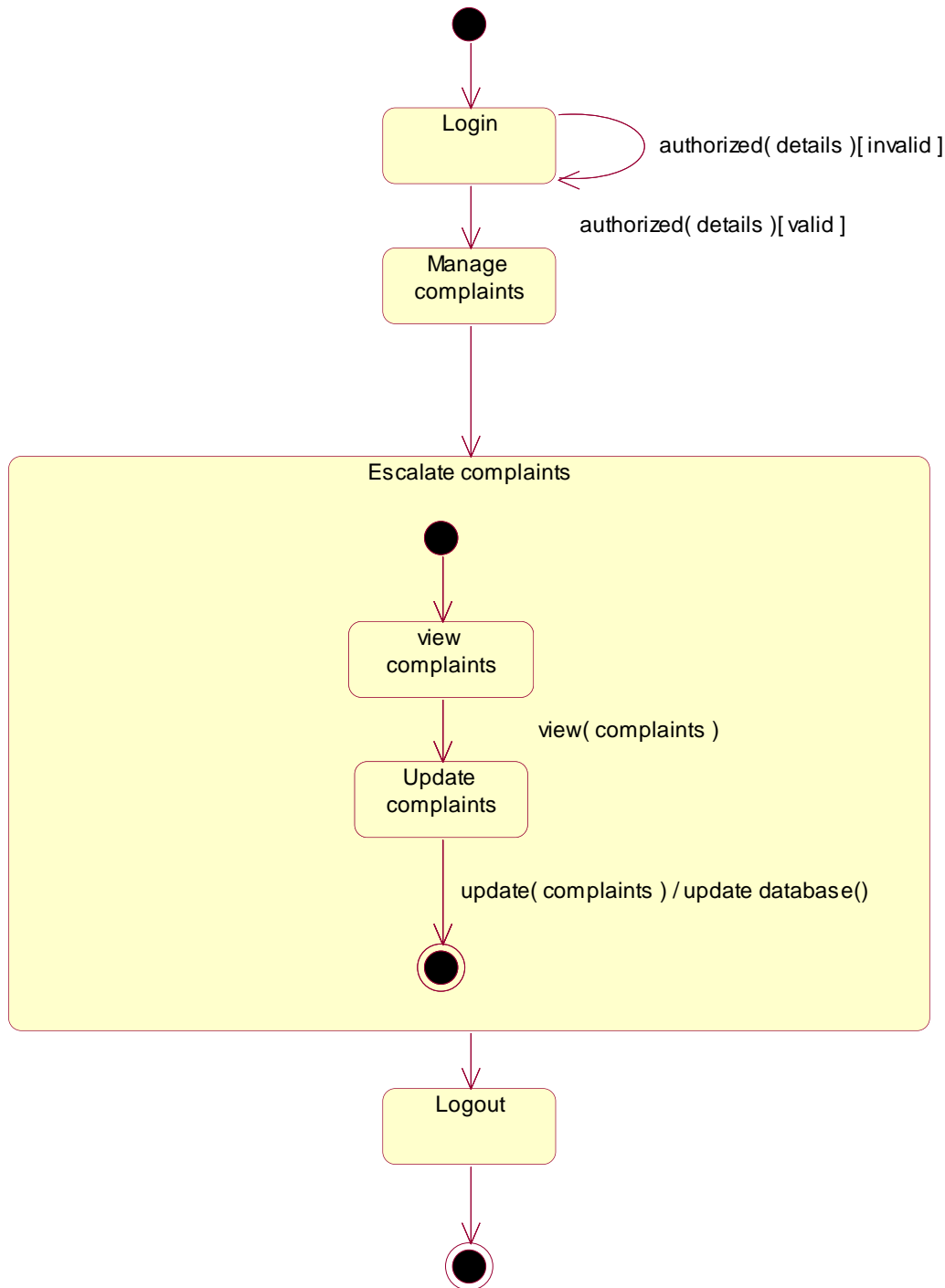
Elements of state chart diagrams:

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

Statechart diagram for check complaint status:



state chart diagram for manage complaints:



DESIGN

19. 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 [0 ... 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.

The following are the different kinds of visibilities, their symbols and their meaning.

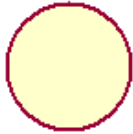
Visibility symbol	Visibility	Meaning
+	Public	The feature (an operation or an attribute) is directly accessible by an instance of any class.
-	Private	The feature may only be used by an instance of the class that includes it.
#	Protected	The feature may be used either by instances of the class that includes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

Interfaces :

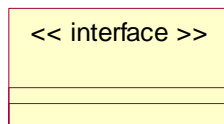
Generally a class may present more than one external interface to other classes or the same interface may be required from more than one class. An interface in UML is a group of externally visible (i.e. public) operations. The interface contains no internal structure, it has no attributes, no associations 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

20. IMPLEMENTATION DIAGRAMS.

20 .a. 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.

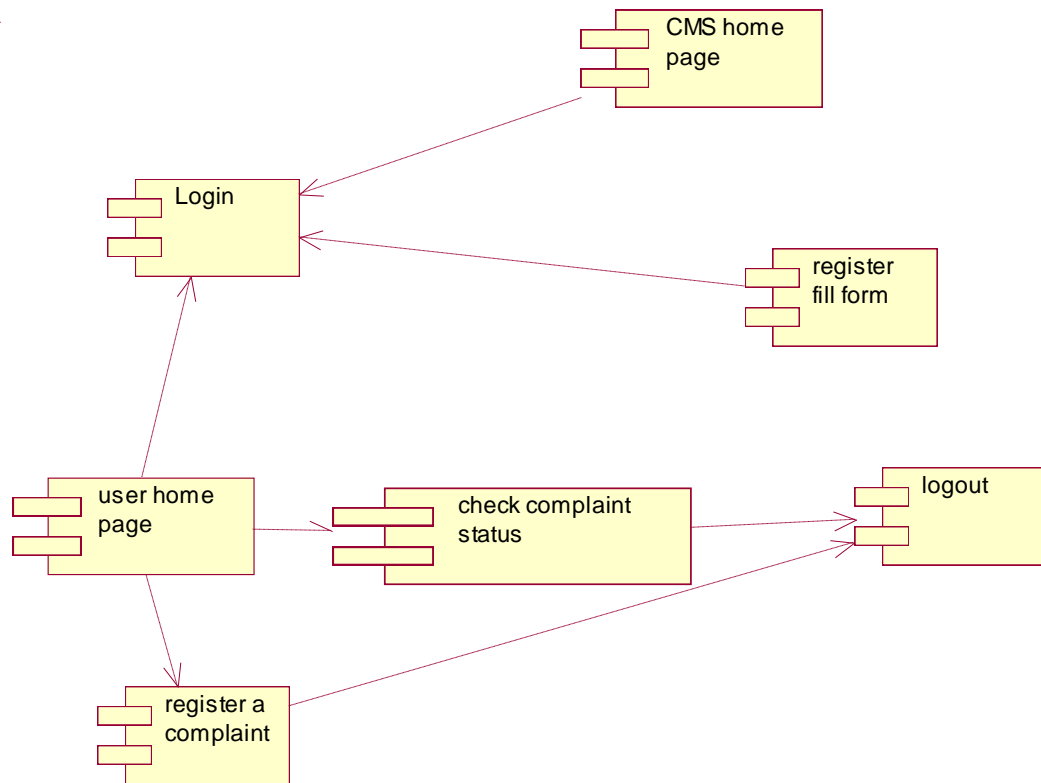
Types of components:

- Deployment component
- Work product component
- Executable component

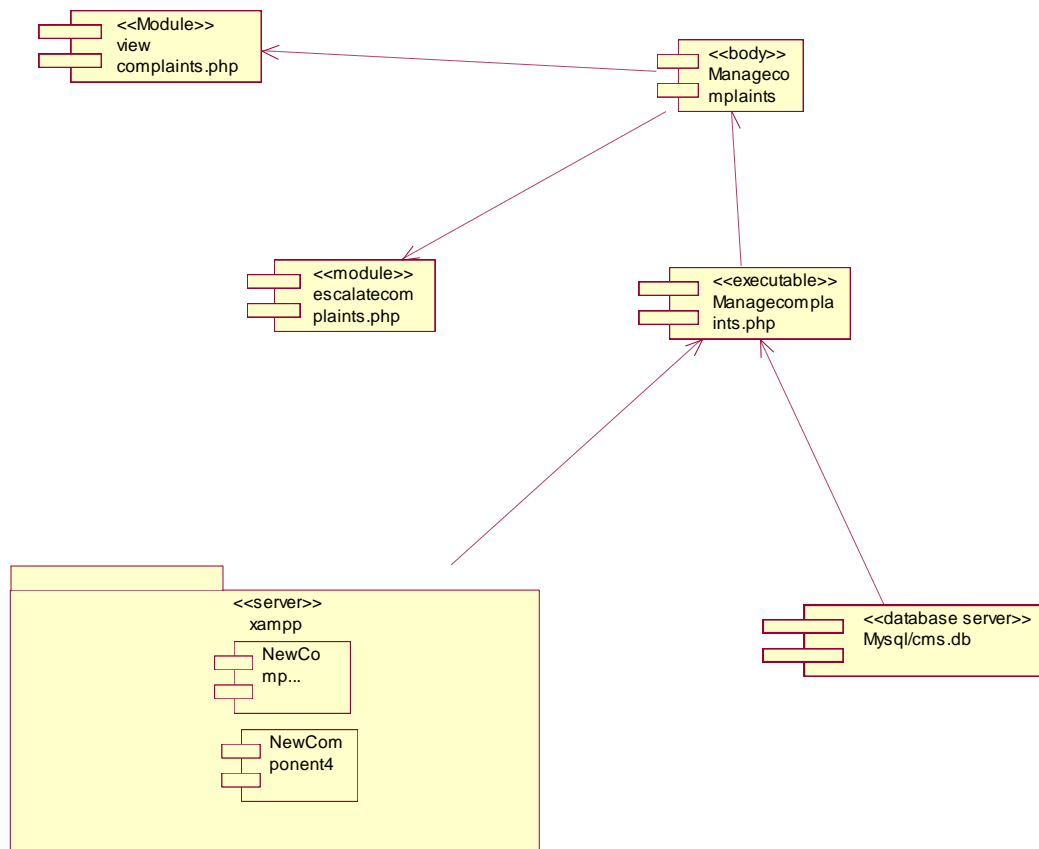
Uses:

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

Component diagram for user:



Component diagram for Manage complaints:



20 .b. DEPLOYMENT DIAGRAMS:

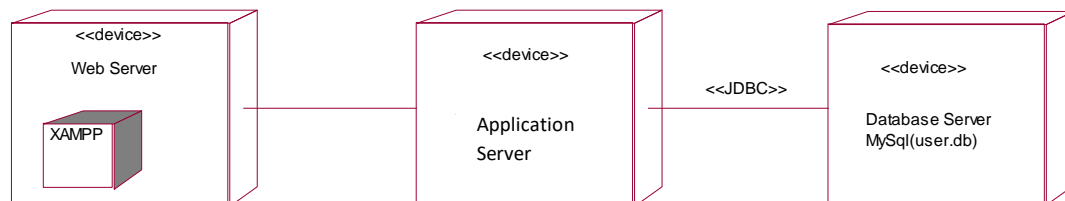
- The second type of implementation diagram provided by UML is the deployment diagram. Deployment diagrams are used to show the configuration of run-time processing elements and the software components and processes that are located on them.
- Deployment diagrams are made up of nodes and communication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



IMPLEMENTATION OF DOMAIN OBJECTS LAYER AND

TECHNICAL SERVICE LAYER:

code for home page:

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="home_css.css">
</head>
<body>
<center>
<div id="div1">
<imgsrc="logo.jpg" id="logo">
Complaint managment system
</div>
</center>
<div id="div2">
<div id="div3">
<ul>
<li><a href="home_html.html">Home</a></li>
<li><a href="user_login.php">User</a></li>
<li><a href="incharge_login.php">Incharge</a></li>
<li><a href="admin_login.php">Admin</a></li>
<li><a href="help.html">Help</a></li>
</ul>
```

```
</div>
<div id="div4">
<center>
<h1>SELCO</h1>
<imgsrc="cms_new.jpg">
</center>
</div>
</div>
</body>
</html>
```

Code for register:

```
<html>
<head>
<title>
login form</title>
<style>
table{
border:none;
margin-top:130px;
color:#2E2E2E;
}
td{
font-size:30px;
}
```



```
#but1{
font-size:30px;
width:100;
height:40;
background-color:#4B8A08;
color:white;
}

h1{
font-size:40px;
margin:30px;
color:#FF00BF;
text-decoration:underline;
}

.row{
height:30px;
width:200px;
border-radius:3px;
font-size:20px;
}

body{
background-image:url("bg9.jpg");
background-repeat:no-repeat;
background-attachment:fixed;
width:100%;
height:100%;
```

```

font-family:calibri;

font-size: 40px;

}

</style>

</head>

<body>

<center>

<h1>Register Here</h1>

<table border=0px cellspacing=0 cellpadding=0>

<form action="register_validate.php" method="post">

<tr>

<td>Username: </td>

<td><input type="text" name="uname" class="row" placeholder="Enter username"></td>

<tr><td>&nbsp;</td></tr>

<td>Password: </td>

<td><input type="password" name="pass" class="row" placeholder="Enter password"></td>

</tr>

<tr><td>&nbsp;</td></tr>

<tr>

<td>Confirm password: </td>

<td><input type="password" name="confirm" class="row" placeholder="Enter password"></td>

</tr>

<tr><td>&nbsp;</td></tr>

<tr>

<td colspan="2" rowspan="2" align="center"><button type="submit" name="submit"
id="but1">submit</button></td>

```

```
</tr>
</form>
</center>
</body>
</html>
```

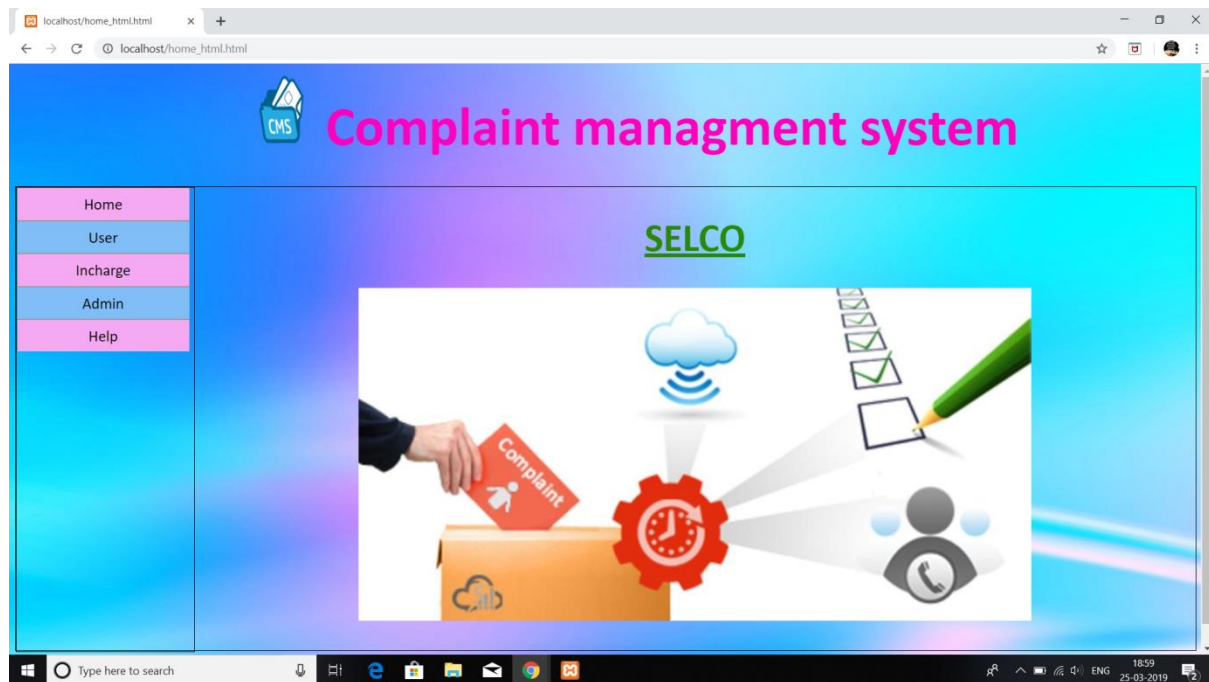
Register_validate.php

```
<?php
    $con=mysqli_connect('localhost','root','');
    if(!$con)
    {
        echo 'not connected';
    }
    if(!mysqli_select_db($con,'cms'))
    {
        echo 'database not selected';
    }
    $username=$_POST['uname'];
    $password=$_POST['pass'];
    $confirm=$_POST['confirm'];
    if($password!=$confirm)
    {
        header("Location:user_register1.php");
    }
}
```

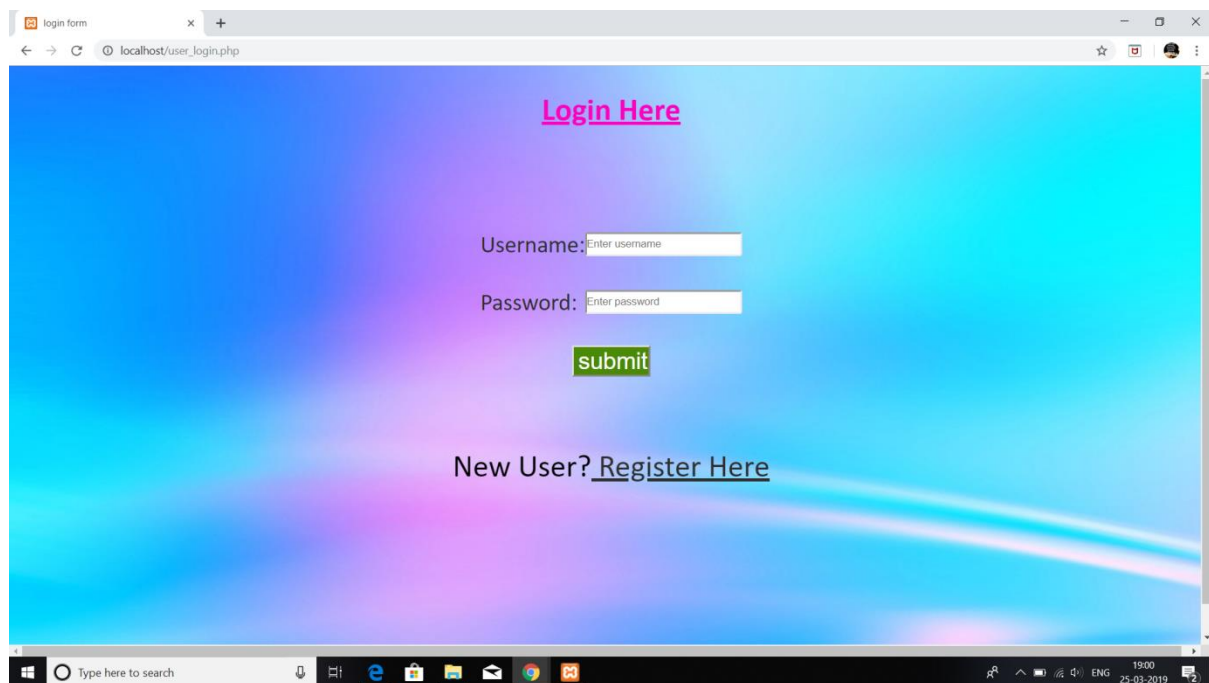
```
}  
else  
{  
    $sql="INSERT INTO user_login (username,password) VALUES ('$username','$password')";  
    if(mysql_query($con,$sql))  
    {  
        header("Location: reg_login.php");  
    }  
    else{  
        header("Location: user_register1.php");  
    }  
}  
?  
>
```

(d) IMPLEMENTATION OF USER INTERFACE LAYER

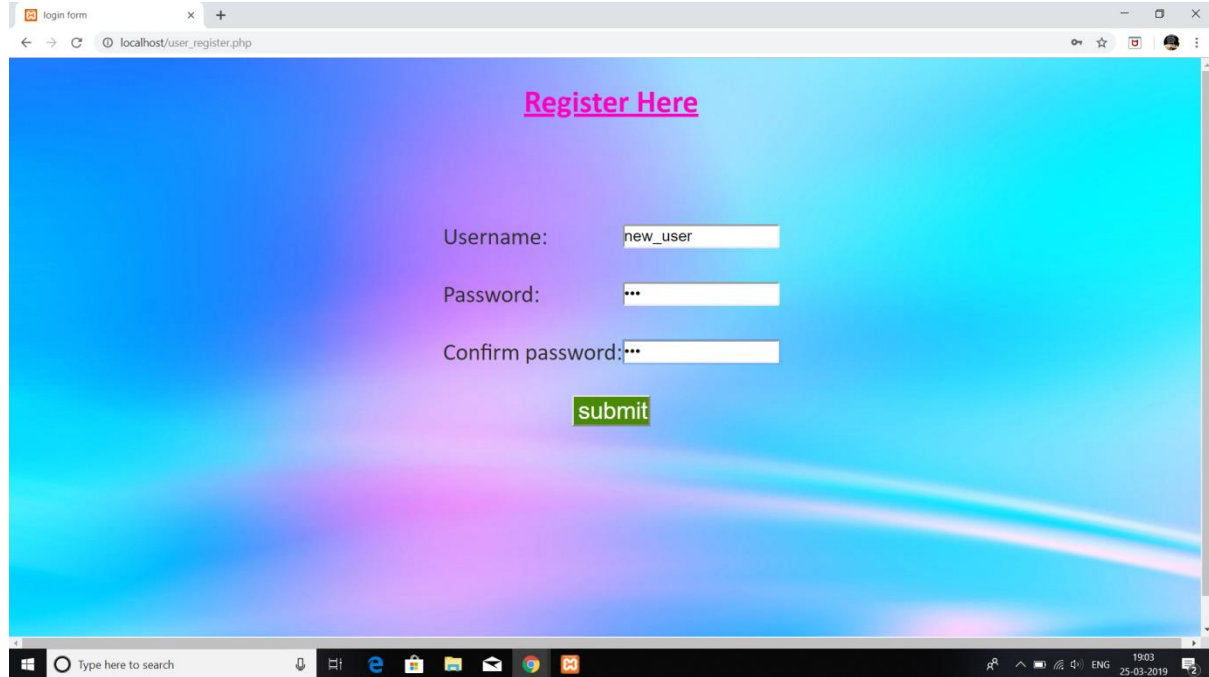
Home page:



New user Registration:



New user Registration:



The screenshot shows a web browser window with the address bar displaying 'localhost/user_register.php'. The page has a blue and purple gradient background. At the top, the text 'Register Here' is written in pink. Below this, there are three input fields: 'Username:' with the value 'new_user', 'Password:', and 'Confirm password:'. A green 'submit' button is located below the input fields. The Windows taskbar is visible at the bottom, showing the search bar and various application icons.

login form

localhost/user_register.php

[Register Here](#)

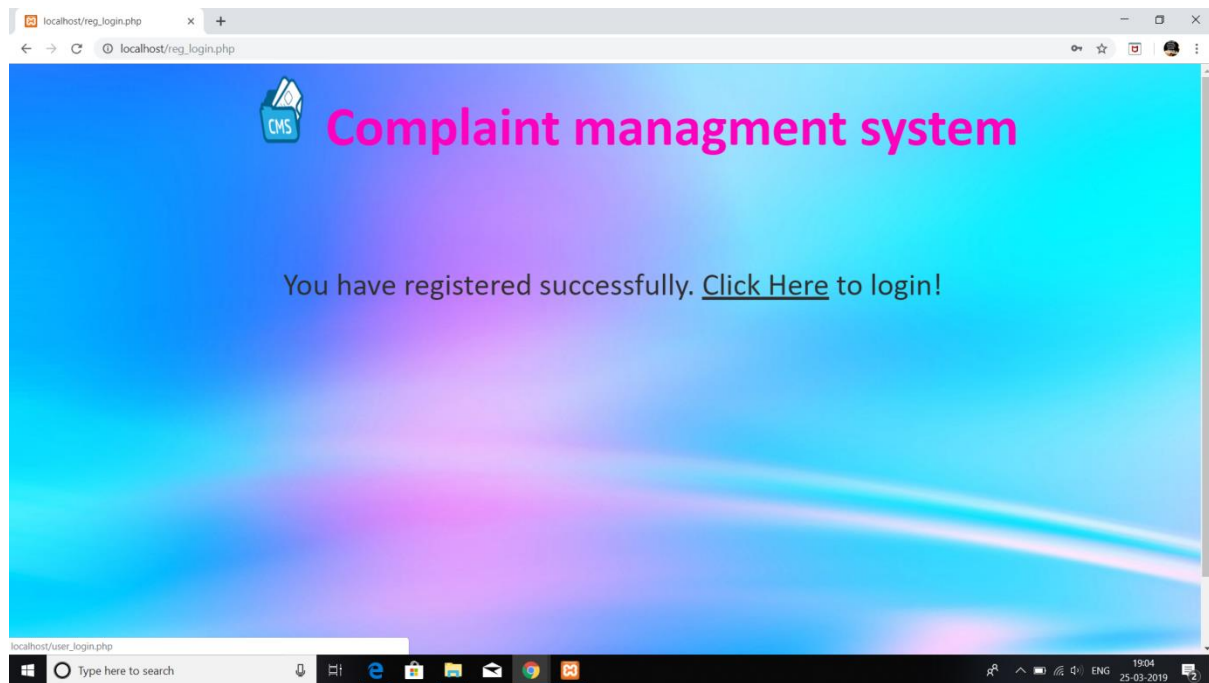
Username: new_user

Password: ...

Confirm password: ...

submit


User login after registration:



The screenshot shows a web browser window with the address bar displaying 'localhost/reg_login.php'. The page has a blue and purple gradient background. At the top left, there is a small icon of a document with the letters 'CMS' on it. To the right of the icon, the text 'Complaint managment system' is written in pink. Below this, the text 'You have registered successfully. [Click Here](#) to login!' is displayed. The Windows taskbar is visible at the bottom, showing the search bar and various application icons.

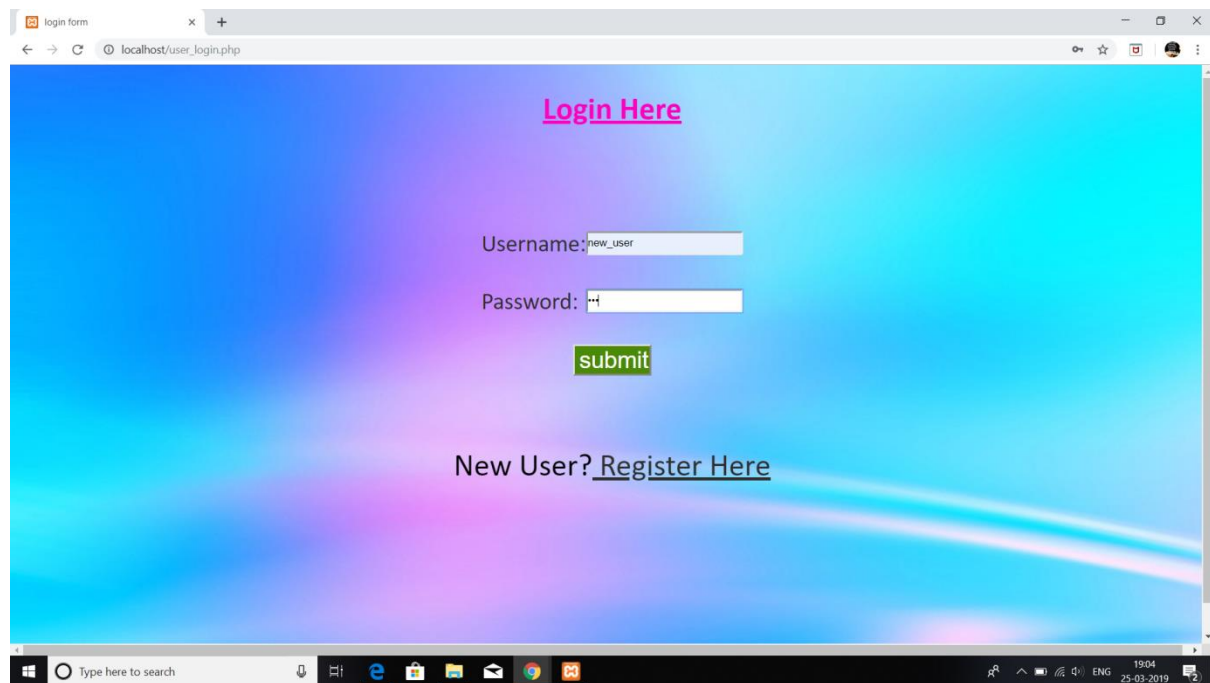
localhost/reg_login.php

localhost/reg_login.php

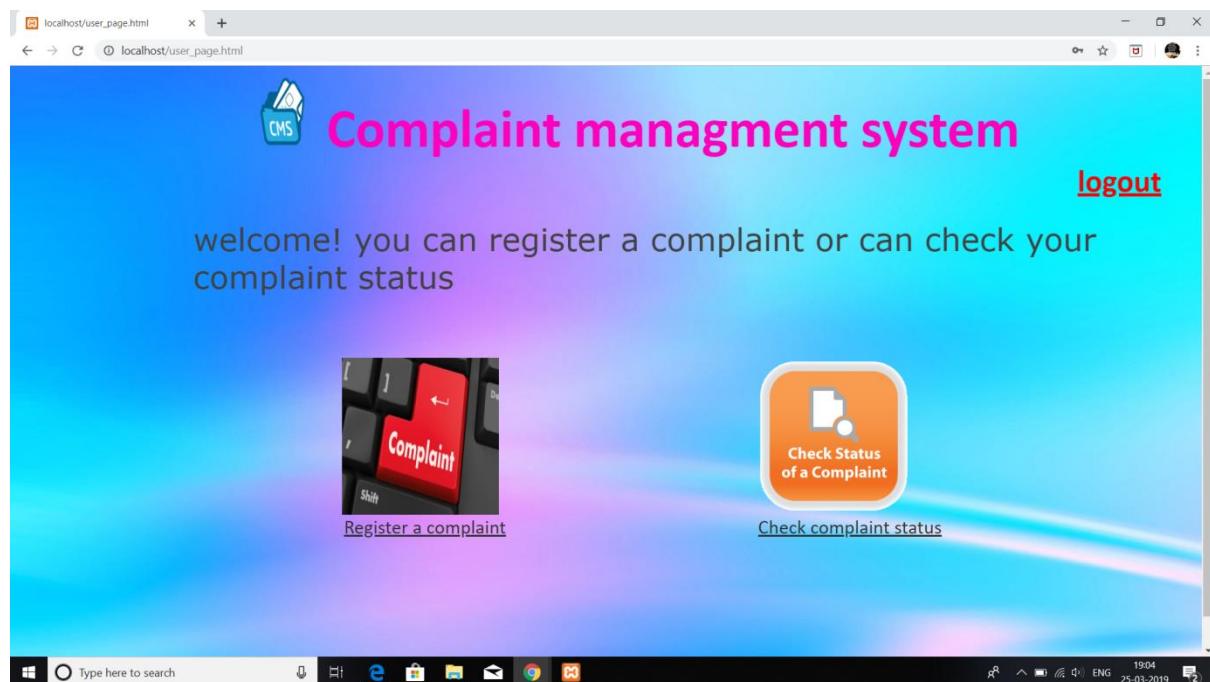
 Complaint managment system

You have registered successfully. [Click Here](#) to login!

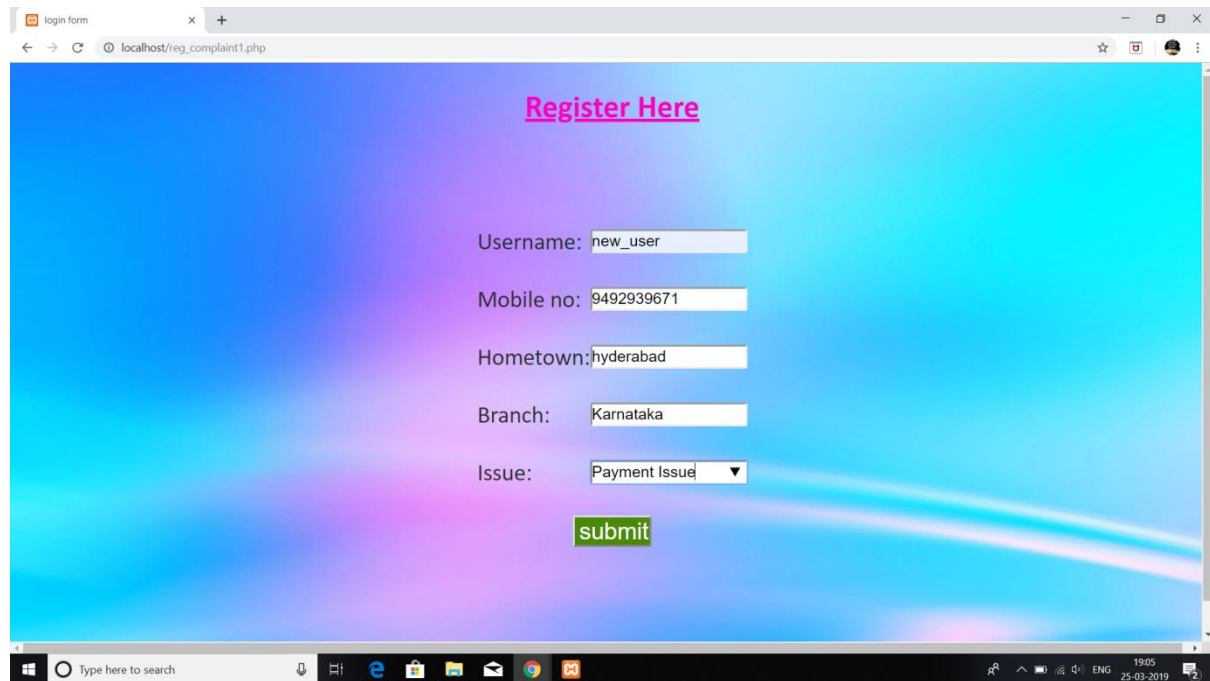
User login page after registration:



User home page:

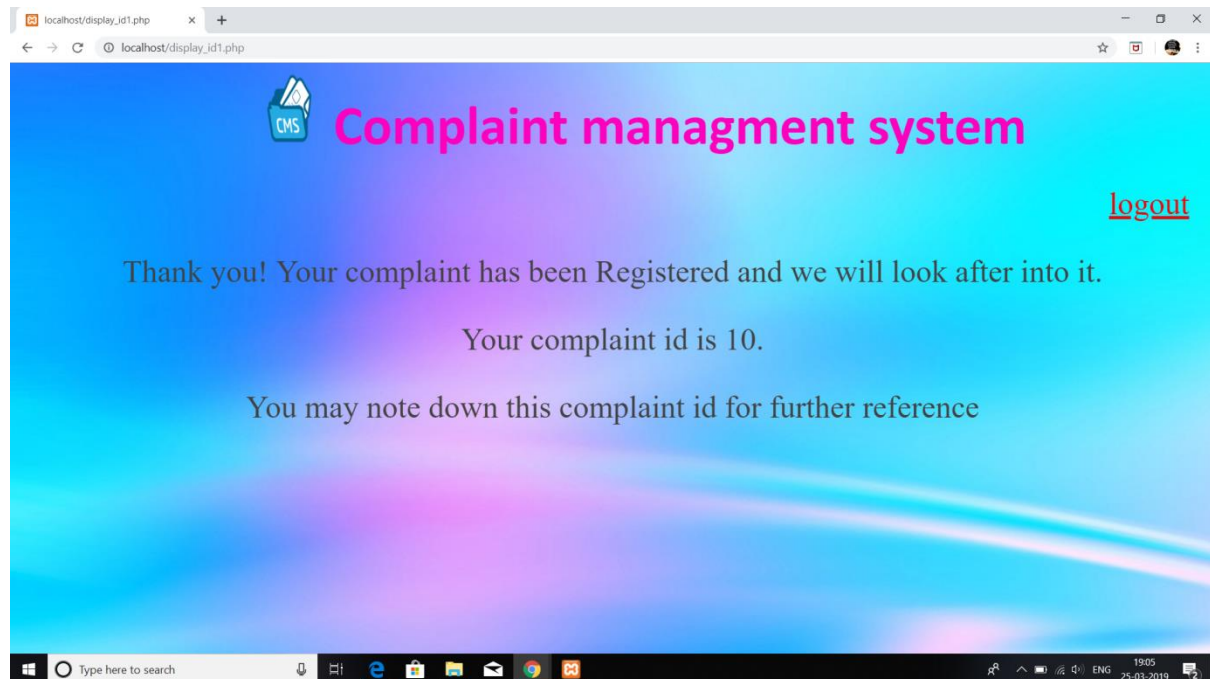


Registering complaint :



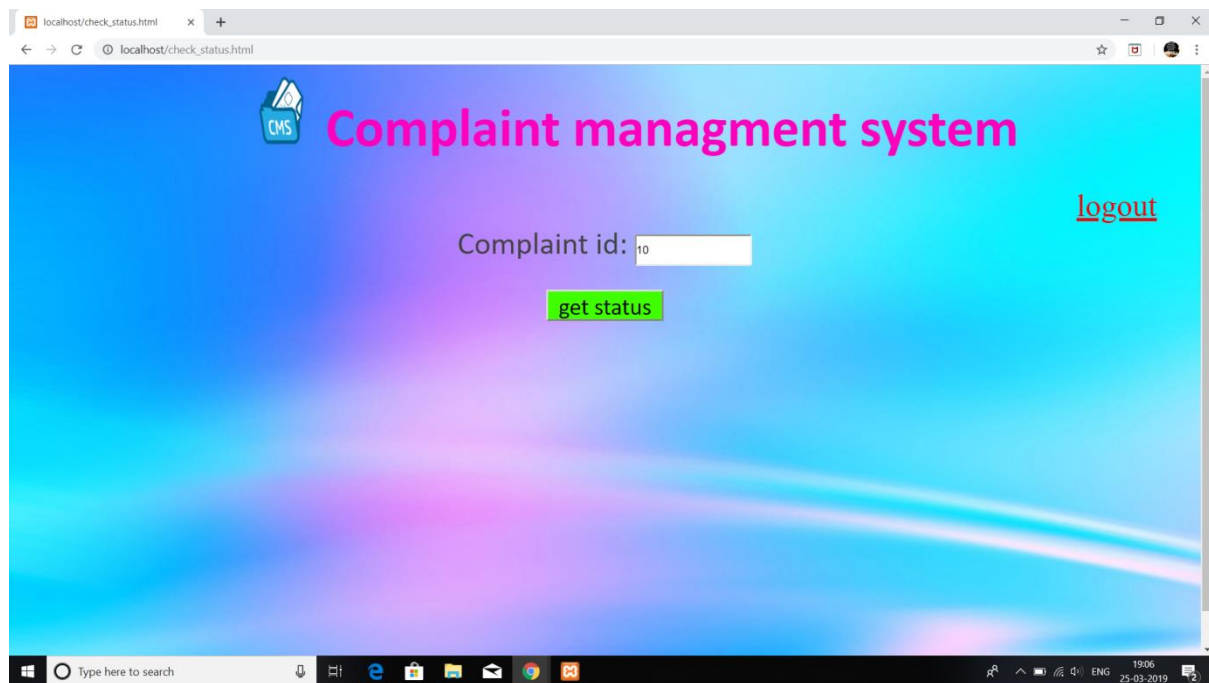
The screenshot shows a web browser window with the address bar displaying 'localhost/reg_complaint1.php'. The page has a blue and purple gradient background. At the top center, there is a link 'Register Here' in red text. Below it, there is a registration form with the following fields: 'Username:' with the value 'new_user', 'Mobile no:' with the value '9492939671', 'Hometown:' with the value 'hyderabad', 'Branch:' with the value 'Karnataka', and 'Issue:' with a dropdown menu showing 'Payment Issue'. A green 'submit' button is located below the form fields. The Windows taskbar is visible at the bottom of the browser window.

Generation of unique complaint id:

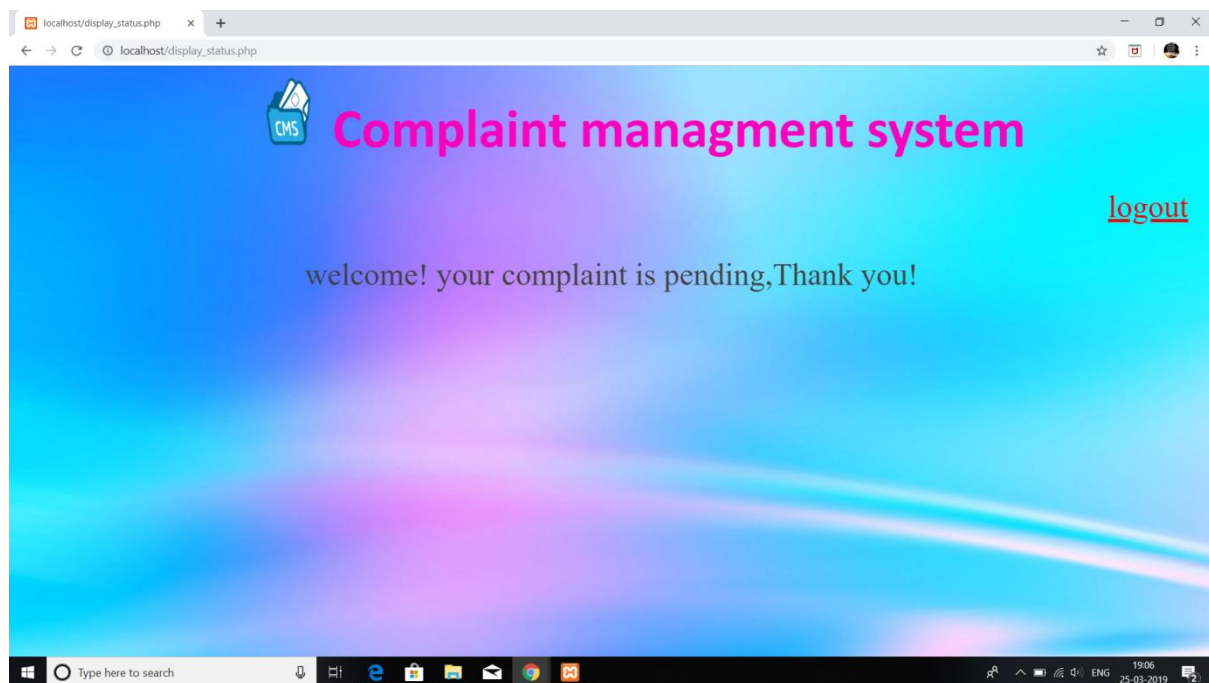


The screenshot shows a web browser window with the address bar displaying 'localhost/display_id1.php'. The page has a blue and purple gradient background. At the top left, there is a logo with a blue box containing 'CMS' and a stack of money. To the right of the logo, the text 'Complaint managment system' is displayed in red. In the top right corner, there is a red 'logout' link. The main content of the page is a confirmation message: 'Thank you! Your complaint has been Registered and we will look after into it.' followed by 'Your complaint id is 10.' and 'You may note down this complaint id for further reference'. The Windows taskbar is visible at the bottom of the browser window.

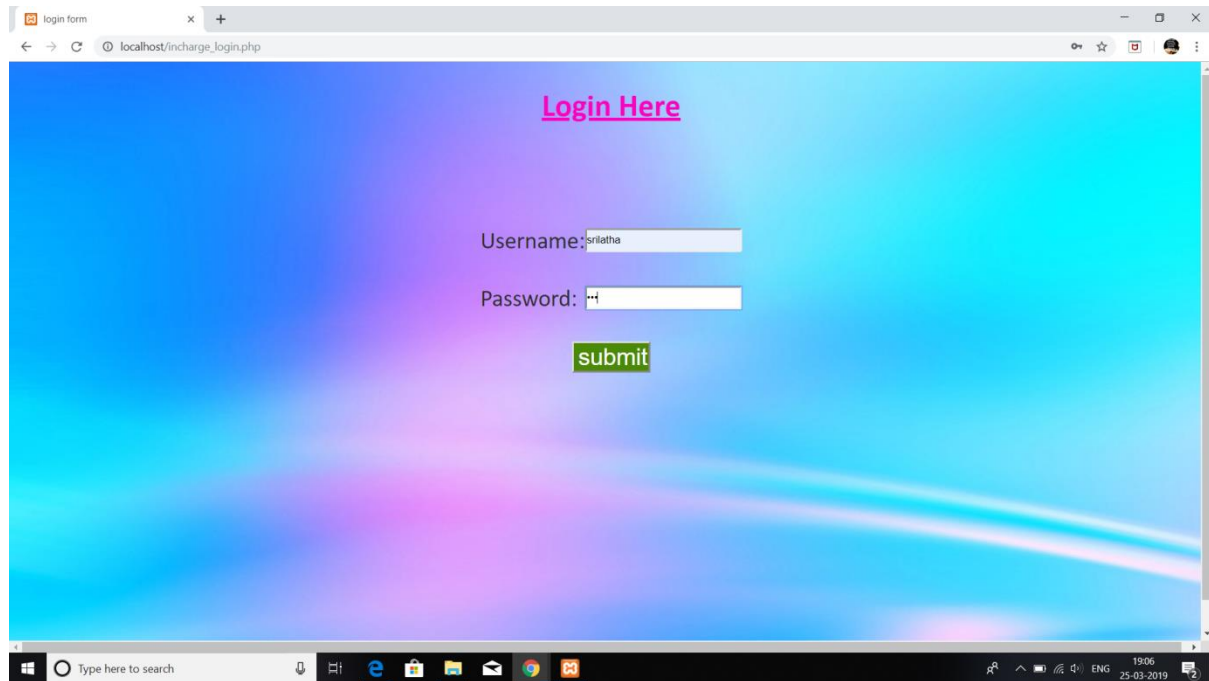
Checking complaint status:



Displaying status:



Incharge login:



A screenshot of a web browser displaying a login form. The browser's address bar shows 'localhost/incharge_login.php'. The page has a blue and purple gradient background. At the top center, the text 'Login Here' is displayed in pink. Below it, there are two input fields: 'Username' with the value 'sriatha' and 'Password' with a masked value '···'. A green 'submit' button is positioned below the password field. The Windows taskbar at the bottom shows the search bar and various application icons.

login form

localhost/incharge_login.php


[Login Here](#)

Username: sriatha

Password: ···

submit

Incharge home page:



A screenshot of a web browser displaying the home page of a 'Complaint management system'. The browser's address bar shows 'localhost/incharge.html'. The page has a blue and purple gradient background. At the top left, there is a CMS icon. The title 'Complaint management system' is displayed in pink. On the top right, there is a 'logout' link. The main text reads 'welcome! you can view OCR or update OCR form'. Below this text, there are two buttons: 'VIEW OCR' with a magnifying glass icon and 'UPDATE OCR' with a red 'UPDATE' button icon. The Windows taskbar at the bottom shows the search bar and various application icons.

localhost/incharge.html

localhost/incharge.html

CMS

Complaint management system

[logout](#)

welcome! you can view OCR or update OCR form

VIEW

UPDATE

[VIEW OCR](#)

[UPDATE OCR](#)

View OCR page:



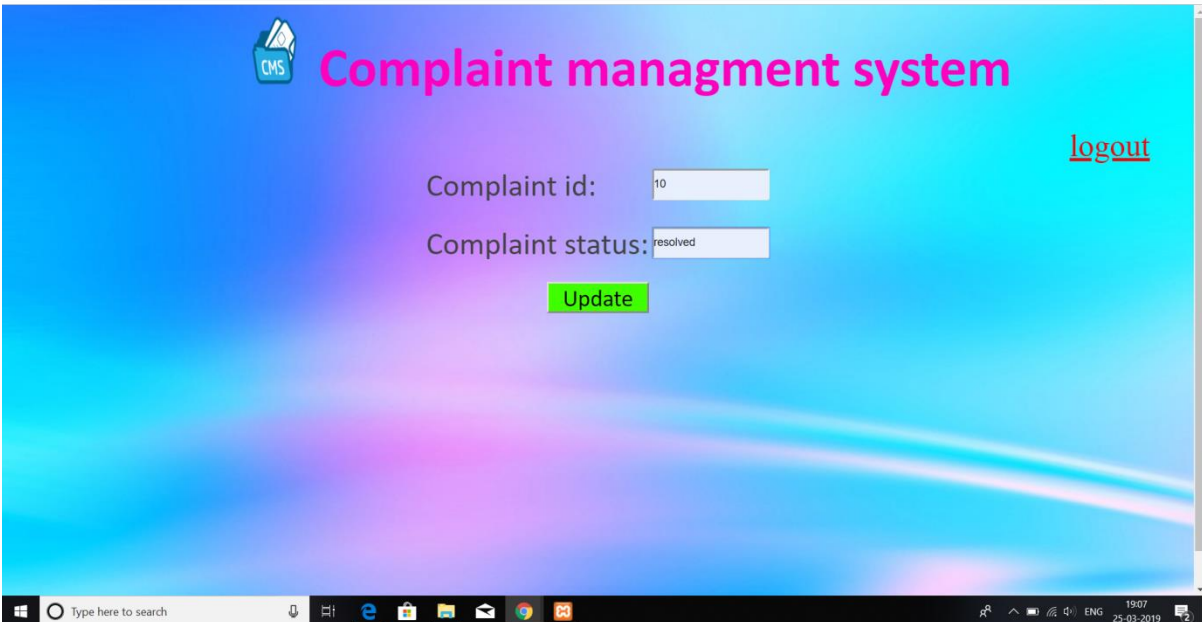
Complaint management system

[logout](#)

Open Complaints Register

Complaint ID	username	Issue	Branch	Incharge	Date of registration	Status
2	sowji	Delievery or Collection Issue	Tamil Nadu	AGM	2019-03-13	pending
3	pooja	Product Warranty Issue	Maharastra	AGM	2019-03-13	pending
4	priya	Product Issue	Bihar	AGM	2019-03-11	pending
5	keerthi	Product Warranty Issue	Bihar	AGM	2019-03-15	pending
6	thanmay	Product Warranty Issue	Bihar	CSD - Head, AM	2019-03-17	pending
7	mounika	Payment Issue	Tamil Nadu	AGM	2019-03-14	pending
8	manisha	Product Warranty Issue	Karnataka	CSD - Head, AM	2019-03-17	pending
9	sample	Product Issue	Kerala	OA, CSE/Sr CSE	2019-03-22	pending
10	new_user	Payment Issue	Karnataka	OA, CSE/Sr CSE	2019-03-25	pending

Updating OCR:



Complaint management system

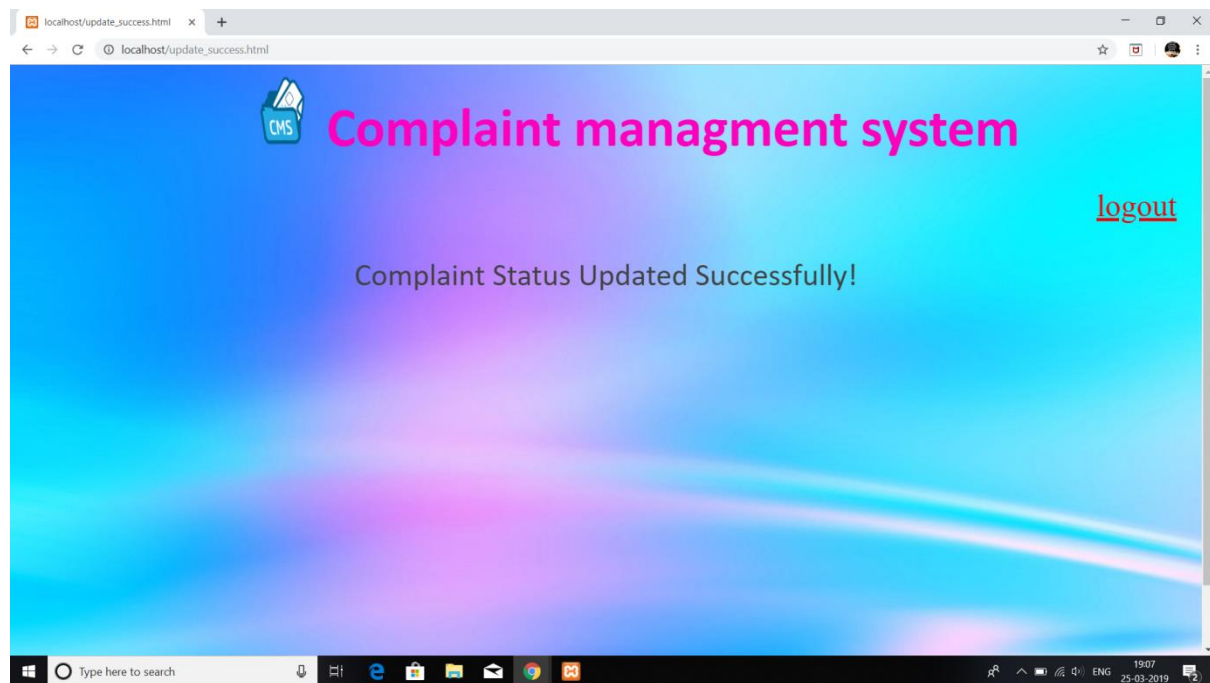
[logout](#)

Complaint id:

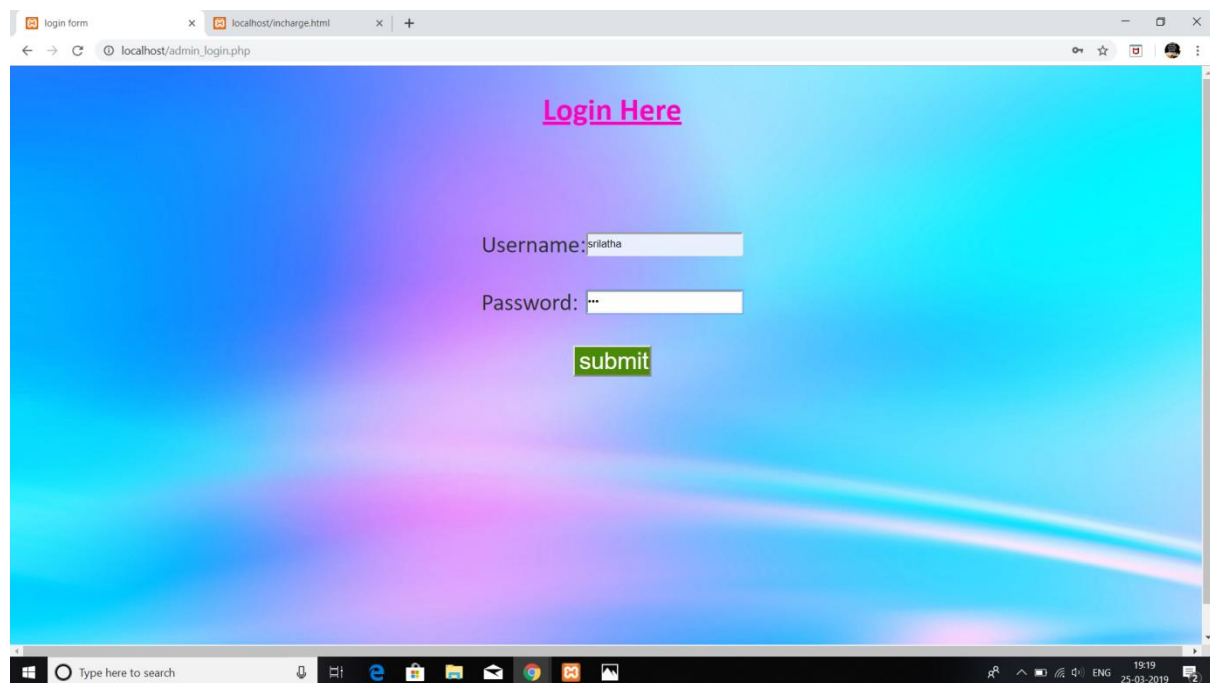
Complaint status:

[Update](#)

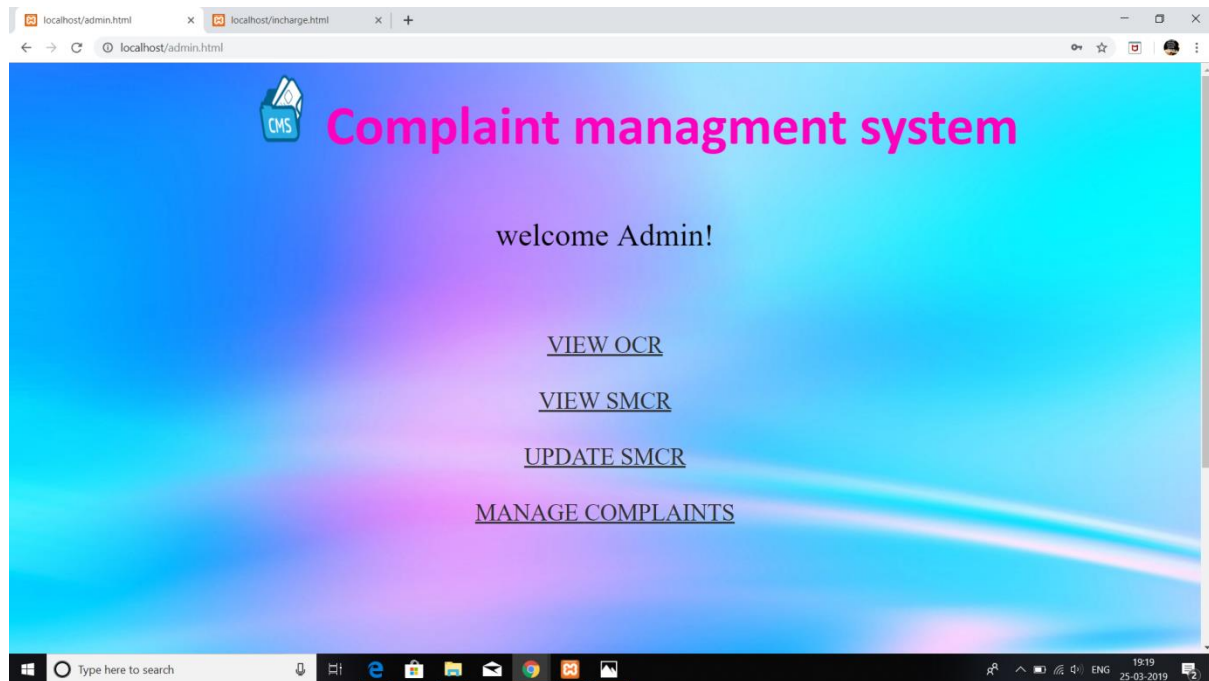
Confirmation :



Admin login:



Admin home page:



View OCR page:



View SMCR :

Complaint management system

[logout](#)

SELCO Master Complaints Register

Complaint ID	username	Issue	Branch	Incharge	Date of registration	Status
2	sowji	Delievery or Collection Issue	Tamil Nadu	AGM	2019-03-13	pending
3	pooja	Product Warranty Issue	Maharastra	AGM	2019-03-13	pending
4	priya	Product Issue	Bihar	AGM	2019-03-11	pending
5	keerthi	Product Warranty Issue	Bihar	AGM	2019-03-15	pending
6	thanmay	Product Warranty Issue	Bihar	AGM	2019-03-17	pending
7	mounika	Payment Issue	Tamil Nadu	AGM	2019-03-14	pending
8	manisha	Product Warranty Issue	Karnataka	AGM	2019-03-17	pending
9	sample	Product Issue	Kerala	RCSE, BM	2019-03-22	pending
10	new_user	Payment Issue	Karnataka	OA, CSE/Sr CSE	2019-03-21	pending

Updating SMCR:

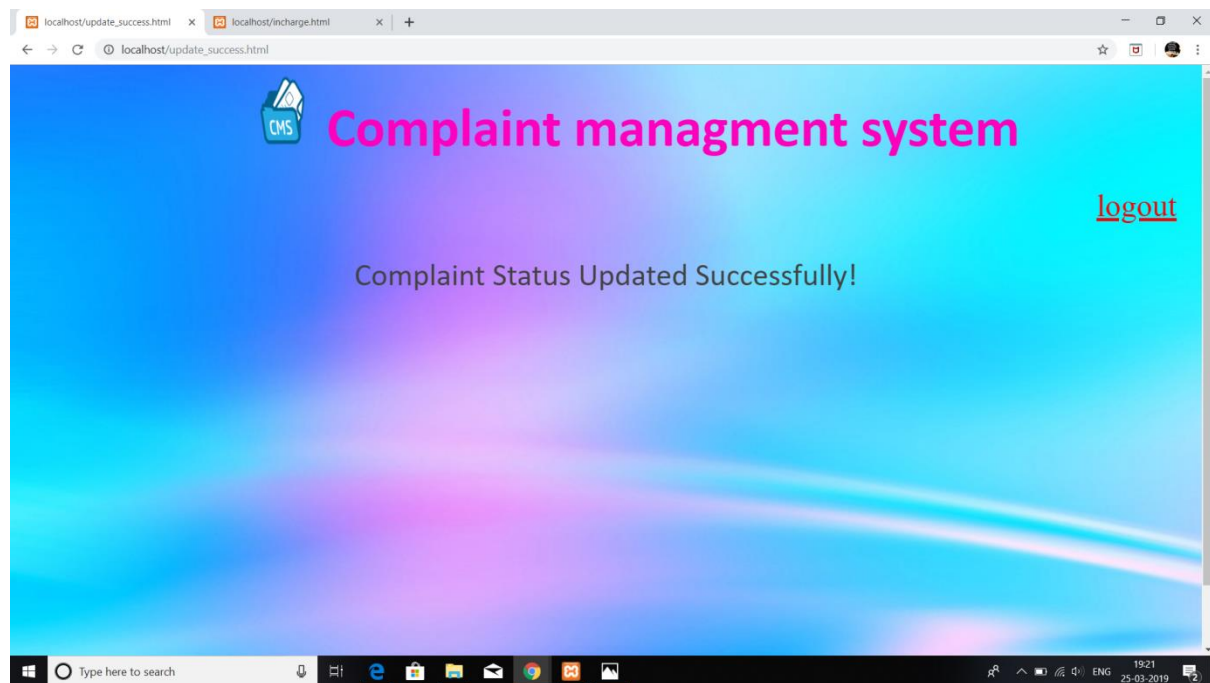
Complaint management system

[logout](#)

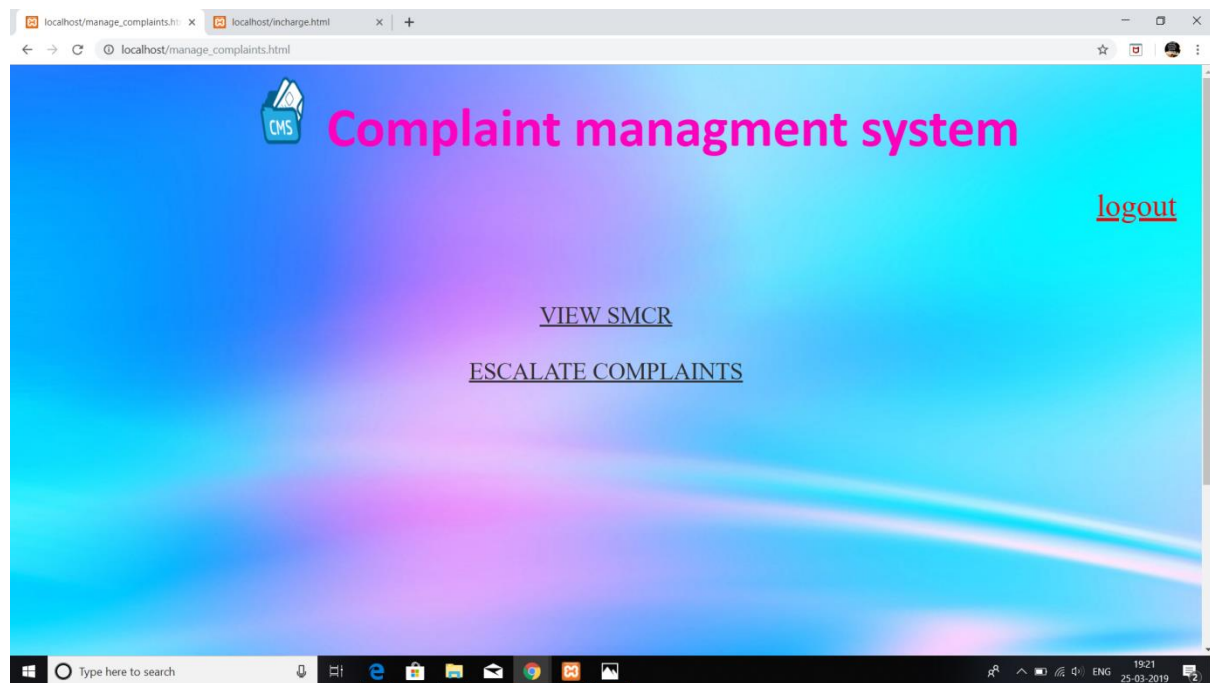
Complaint id:

Complaint status:

Update confirmation:



Managing complaints:



View SMCR:

The screenshot shows a web browser window with two tabs: 'localhost/view_smcr.php' and 'localhost/incharge.html'. The active page is 'localhost/view_smcr.php', which displays the 'Complaint management system' interface. The page has a blue header with a 'CMS' logo and a 'logout' link. Below the header, the title 'SELCO Master Complaints Register' is centered. A table with 7 columns (Complaint ID, username, Issue, Branch, Incharge, Date of registration, Status) and 10 rows of complaint data is displayed. The table data is as follows:

Complaint ID	username	Issue	Branch	Incharge	Date of registration	Status
2	sowji	Delievery or Collection Issue	Tamil Nadu	AGM	2019-03-13	pending
3	pooja	Product Warranty Issue	Maharastra	AGM	2019-03-13	pending
4	priya	Product Issue	Bihar	AGM	2019-03-11	pending
5	keerthi	Product Warranty Issue	Bihar	AGM	2019-03-15	pending
6	thanmay	Product Warranty Issue	Bihar	AGM	2019-03-17	pending
7	mounika	Payment Issue	Tamil Nadu	AGM	2019-03-14	pending
8	manisha	Product Warranty Issue	Karnataka	AGM	2019-03-17	pending
9	sample	Product Issue	Kerala	RCSE, BM	2019-03-22	pending
10	new_user	Payment Issue	Karnataka	OA, CSE/Sr CSE	2019-03-21	pending

The Windows taskbar at the bottom shows the search bar, task view button, and several application icons. The system clock indicates 19:20 on 25-03-2019.

Page displayed after escalation of complaints:

The screenshot shows the same web browser window with the 'localhost/escalate.html' page active. The page displays the 'Complaint management system' header with the 'CMS' logo and 'logout' link. The main content area shows the message 'complaints escalated successfully.' followed by a link '[click here](#) to view updated SMCR.' The Windows taskbar and system clock are identical to the previous screenshot.

Updated SMCR:

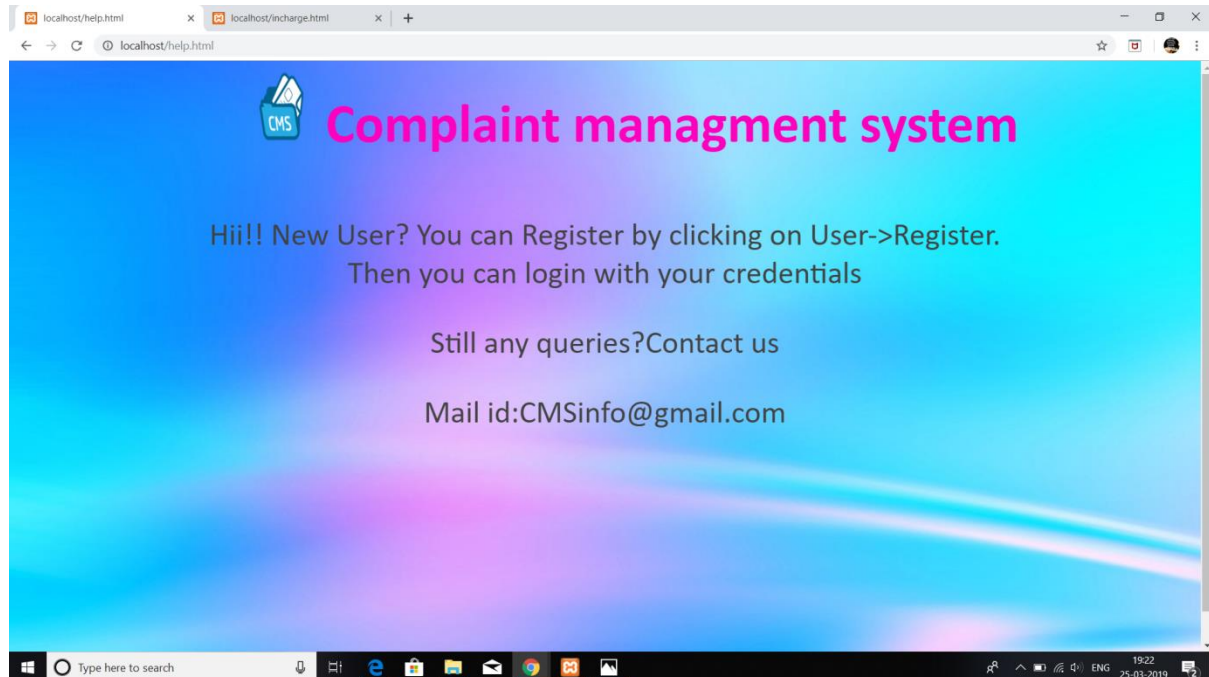


The screenshot shows a web browser window with two tabs: 'localhost/view_smcr.php' and 'localhost/incharge.html'. The active page is 'localhost/view_smcr.php', which displays the 'Complaint management system' interface. The page has a blue header with a 'CMS' logo and a 'logout' link. Below the header, the title 'SELCO Master Complaints Register' is centered. A table lists 10 complaints with columns for Complaint ID, username, Issue, Branch, Incharge, Date of registration, and Status. The table data is as follows:

Complaint ID	username	Issue	Branch	Incharge	Date of registration	Status
3	pooja	Product Warranty Issue	Maharashtra	AGM	2019-03-13	pending
4	priya	Product Issue	Bihar	AGM	2019-03-11	pending
5	keerthi	Product Warranty Issue	Bihar	AGM	2019-03-15	pending
6	thanmay	Product Warranty Issue	Bihar	AGM	2019-03-17	pending
7	mounika	Payment Issue	Tamil Nadu	AGM	2019-03-14	pending
8	manisha	Product Warranty Issue	Karnataka	AGM	2019-03-17	pending
9	sample	Product Issue	Kerala	RCSE, BM	2019-03-22	pending
10	new_user	Payment Issue	Karnataka	RCSE, BM	2019-03-21	pending

The browser's taskbar at the bottom shows the Windows search bar and various application icons. The system clock indicates the time is 19:21 on 25-03-2019.

Help page:



The screenshot shows the 'Help page' of the 'Complaint management system'. The page has a blue header with the 'CMS' logo. The main content area contains the following text:

Hiii!! New User? You can Register by clicking on User->Register.
Then you can login with your credentials

Still any queries?Contact us

Mail id:CMSinfo@gmail.com

The browser window shows the 'localhost/help.html' tab. The taskbar at the bottom is identical to the previous screenshot, showing the Windows search bar and application icons, with the system clock at 19:22 on 25-03-2019.

21.Conclusion:

The Complaint management System allows the users to register the complaints. The system has to maintain the details of users. Users have to login through valid username and password and register a complaint. After registering the complaint, they can check status of their complaint and logout.

The Administrator has full authority over user accounts. He maintains the list of registered users, their complaints and their information. He maintains the list of all complaints and their status information. He can change the status of a complaint if it is resolved and he can escalate complaints based on their date of registration. After checking all the information he logs out from his homepage.

The Complaint Management System has many advantages in following ways:

- Better monitoring of customer complaints.
- Better understanding of nature of problems.
- Resolves issues in a timely and cost effective way.
- Helps in improving the services offered to customers.

22.References

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

Web Links:

1. Complaint Managment System – SlideShare [<http://www.slideshare.net>]
2. www.w3schools.com/php
3. <https://stackoverflow.com>