PROBLEM SCENARIO

Abstract:-

In online voting system people can cast their vote through the internet.In order to prevent voter frauds we use two levels of security.A user id and password are used as the first level and the data entered by the user is verified with the contents of the database,if the data is correct then then the face of voter is captured by a web camera and sent to the database.The web page is designed using HTML.The HTML page is then connected to database using PHP.In second level of security the face of the person is verified with the face present in the database and validating using OPENCV.

Existing system:-

In the current voting system,the ballet machines were used in which the symbols of various political parties are displayed.When we press the button with the respective party'ssymbol the voting is done.The chance of fake person casting their vote is more in this existing system.And person has to travel long places to his consistituency to cast his vote.Therefore,we need an effective method to identify the fake voters during voting.So,our project is used for detecting the right person and also making the system to work in online,which helps voters to cast their vote from their place itself.

Fig.1 Exiting Voting Process scenario

Drawbacks of existing system:-

- Existing voting system has many defects such as lengthy process, time taking,no security level
- The chance of fake person casting their vote is more in this existing system.

Proposed system:-

In this proposed system we generate an idea of E-voting. By using two levels of Security.

Level 1:

- Unique id number (UID).
- At the time of voter registration system will request for the unique id from the voter.
- Unique id is verified from the database provide by the election commission.

Level 2:

- Face recognition with respective election commission id number

Advantages of proposed system:-

- The system is highly reliable and secure.
- In the long run the maintenance cost is very less when compared to the present systems.
- Illegal practices like rigging in elections can be checked for.
- It is possible to get instantaneous results and with accuracy.

Minimum Software Requirements:-

Operating system : Windows

Programming Language : HTML,PYTHON,PHP

Database : Oracle/Access

Server Deployment : XAMP

Client-side Scripting : HTML

Workbench : PYTHON

User Interface : HTML, CSS

Minimum Hardware Requirements:-

Processor	: core i5
Hard Disk	: 250GB minimum
Ram	: 256MB or above

ANALYSIS

Requirements Elicitation:

The following table represents categorization of requirements captured for secured online voting system using face recognition.

S.NO	REQUIREMENTS	TYPE	PRIORITY
1.	The ovs shall display list of parties for voter to cast vote	Functional	Must have
2.	The ovs shall support insertion of voter details to database	Functional	Must have
3.	The ovs shall support insertion of new user details	Functional	Must have
4.	The ovs shall support registration of voter	Functional	Must have
5.	The ovs shall allow to user to view and modify the personal details	Functional	Must have
6.	The ovs shall use voter Aadhar number as login id	Functional	Must have
7.	The ovs allows user to select the particular party to vote	Functional	Must have
8.	The ovs shall ask the voter to set a secret key which was used at the time of login	Functional	Must have

9.	The ovs collects voter details consisting of name,email,phone numbers etc	Functional	Must have
10.	The ovs allows voter to change the password	Functional	Must have
11.	The ovs automatically calculate the result of voting	Functional	Must have
12.	The ovs allow admin to check details of votes casted	Functional	Must have
13.	The ovs shall send OTP to voters registered mobile numbers	Functional	Must have
14.	The ovs provides with a link to email when voter want to reset password	Funtional	Must have
15.	The ovs raises an error when re-registration occurs	Functional	Must have
16.	The ovs provides notification to voter regarding which party voter has voted	Functional	Must have
17.	The ovs organize list of parties by party category	Functional	Must have
18.	The ovs checks the validation of password(OTP)	Functional	Must have
19.	The ovs allows the voter for casting vote to only one party	Functional	Must have

20.	The ovs checks whether the mobile number given by voter was 10 digits or not	Functional	Must have
21.	The ovs allows voter to change or reset passwords	Functional	Must have
22.	The ovs allows voter only once to cast their vote	Functional	Must have
23.	The ovs shall store the Aadhar number to voters who casted their vote in database	Functional	Must have
24.	The ovs checks for Aadhar verification	Functional	Must have
25.	The ovs does not allow the voter to recast their vote	Functional	Must have
26.	The ovs stores the updated data of voter	Functional	Must have
27.	The ovs allows user to logout from the interface	Functional	Must have
28.	In the case of user forgot password the ovs allows to send an OTP to registered mobile number	Functional	Could have
29.	The ovs shall login in a voter within 5 seconds	Non Functional	Could have
30.	The ovs shall support latest versions of Internet explorer and netscape browsers	Non Functional	Could have

31.	The ovs shall always support the previous version of Internet explorer and netscape browsers	Non Functional	Could have
32.	The ovs shall be written using php	Non Functional	Could have
33.	The ovs shall authenticate all users of the system who are not registered	Non Functional	Could have
34.	The ovs keeps a time limit for OTP	Non Functional	Could have
35.	Checks whether the password is strong or not	Functional	Must have

USECASE VIEW

IDENTIFICATION OF ACTORS

Actors represent system users. They are Not part of the system .They represent anyone or anything that interacts with 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,

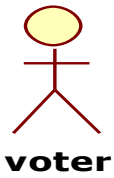
Actors identifiers are:-

1) voter

2) chief officer

3) preceding officer

Voter:-



- The person who register for voting can vote in the particular period of time

Chief Officer:-

- considers the voter profile.
- updates the voter details.
- Generates the edited reports of voters.

Preceding Officer:-

- Validates the user details
- Validates user face
- Collects the final votes

IDENTIFICATION OF USE-CASES OR SUB USE-CASES

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

A use-case diagram can contains:

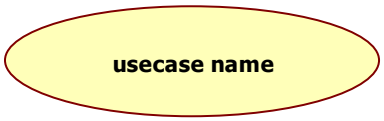
- Actors
- Use cases
- Relationships between actors and use cases in the system

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 simplest form, a use case can be described as a specific way of using the system from a user'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

The UML notation for use case is:-



Purpose of UseCases:-

- Well-structured use cases denote essential system or subsystem behaviors only, and are neither overly general nor too specific.
- A use case represents a functional requirement of the system as a whole
- A use case describes a set of sequences, in which each sequence represents the interaction of the things outside the system with the system itself.

Use cases identifier for secured online voting system using face recognition:-

1) Use case name: Register for vote

- This use case allows to register for vote to cast the vote.

2) Use case name: Apply for changes

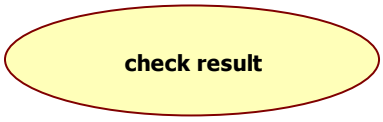
- This use case allows toapply for the changes

3) Use case name: Place an vote

- This use case allows to cast their vote.

4) Use case name: Check result

- This use case to check their result.



5) Use case name: Validate user details

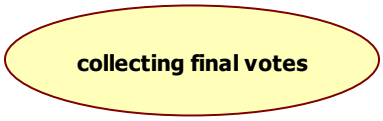
- This use case allows to check their profile.

6) Use case name: Validate user face

- This use case allows to check their face.

7) Use case name: Collecting final votes

- This use case allows to collect all the votes.

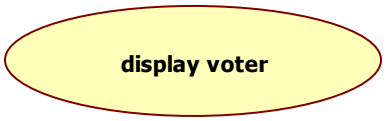


8) Use case name:Update voter

- This use case allows to update their details.

9) Use case name: Display voter

- This use case allows tester to display the voter details.



10) Use case name: Generates reports

- This usecase displays the result.

11) Use case name: Edit voter profile

- This use case modifies the voters details.

12) Use case name:Add new candidates

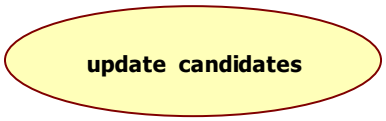
- This use case adds the new candidates to voter list.

13) Use case name:Delete candidates

- This use case deletes the voters from the voter list.

14) Use case name:Update candidates

- This use case updates the list of candidates.



BUILDING REQUIREMENTS MODEL THROUGH USE-CASE DIAGRAM

Definition:

Use-case diagrams graphically represent system behavior. These diagrams present a high level view of how the system is used as viewed from an outsider's perspective.

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:

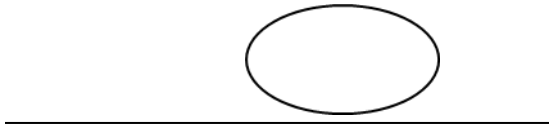
An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. If two objects are usually considered independently, the relationship is an association.

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

1.**Include relationship:**

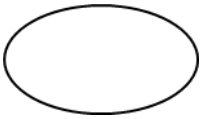
It is a stereotyped relationship that connects a base use case to an inclusion use case An include relationship specifies how the behavior in the inclusion use case is used by the base use case.



2.**Extend relationship:**

It is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case.

<<Extend>> is used when you wish to show that a use case provides additional functionality that may be required in another use case.



USECASE DIAGRAMS

Fig 1- USECASE DIAGRAM FOR VOTER

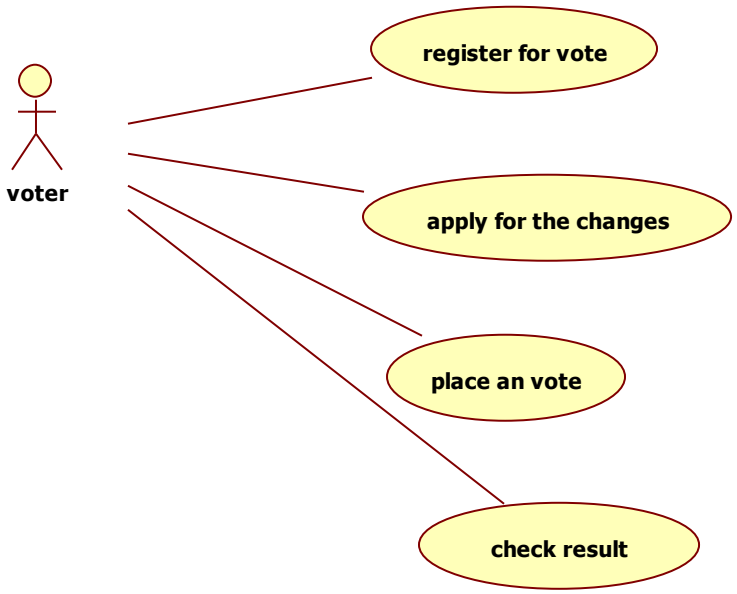


Fig 2- USECASE DIAGRAM FOR PRECEDING OFFICER

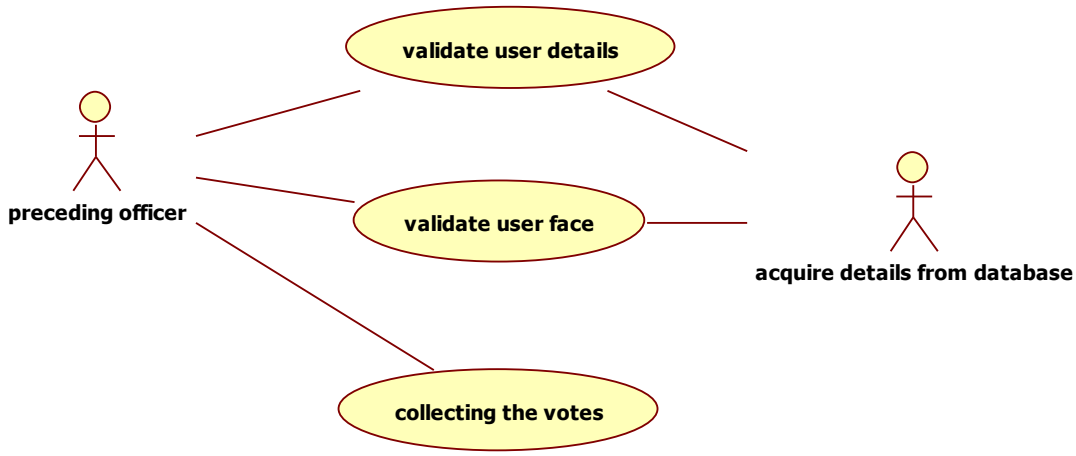
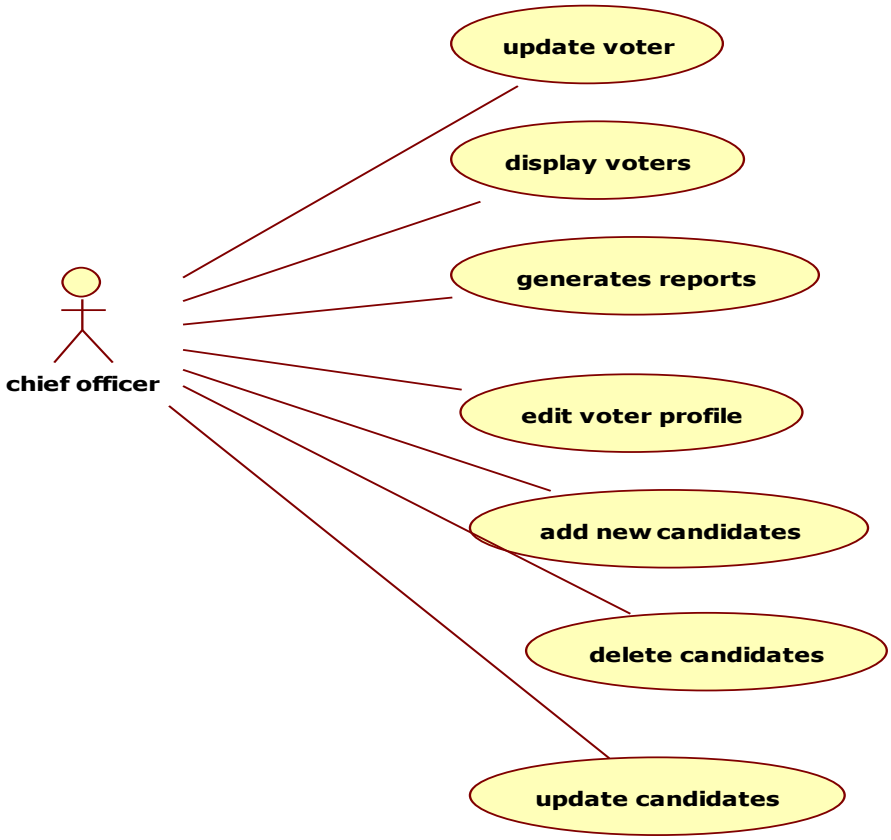


Fig 3- USECASE DIAGRAM FOR CHIEF OFFICER



FLOW OF EVENTS

A flow of events is a sequence of transactions 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 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. Includes the following

1. Use case name – Brief Description
2. Flow of events –
    - Basic flow
    - Alternate flow
    - Special requirements
    - Preconditions
    - Post conditions
    - Extension points

USE CASE SPECIFICATION FOR VOTER USE CASE

1.**Use-case Name : VOTER**

Brief Description:

Voter logins into voting website by entering his username and password.

2. **List of Actors:**

1. Voter
2. System

3**. Initial Conditions:**

This requires the user must be a valid user(above 18)and must be a registered user of the voting systemand must have valid internet connection.

4**. Normal Flow of Events:**

1. Opensvoting system login page.
2. App displays login page.
3. Displays login options.
4. Enter all details corresponding to all fields.
5. Validate user details with existing database.
6. Displays confirmation message.
7. Asks for user login.
8. Selects login to confirmation account confirmation.
9. View details ,

5**. Exceptional Flow of Events:**

1. User already exists.
2. Invalid user name or password.

3. Network failure.

4. Server error.

5. Insufficient details.

6**. Exit Conditions:**

1. Successfully casting the vote.

2. Closes the login page.

7. **Extension Points:**

No extension points.

USE CASE SPECIFICATION FOR VALIDATING USER DETAILS

1.**Use-case Name: Validating voter details**

Brief Description:

To test the details of voter are valid or not by preceding officer.

2. **List of Actors:**

1. Preceding officer
2. System

3**. Initial Conditions:**

This requires the user must be a valid user and must be a registered user of the voting system, must have valid internet connection.

4**. Normal Flow of Events:**

1. Opens voting system Login page
2. Enter password.
3. Validates password.
4. View details.
5. Find bugs in the details.
6. Record in database.
7. Notify user and manager.
8. get user information.
9. Find error information.
10. find answers in data base of bugs.
11. add report to bug tracking system
12. send notification to chief officer.

13.add bug information to voter.

14. App displays signup/login page.

5**. Exceptional Flow of Events:**

1. User already exists.

2. User is below 18 years.

3. Invalid user name or password.

4. Network failure.

5. Server error.

6**. Exit Conditions:**

1. Re apply for vote.

2.Closes App.

7. **Extension Points:**

No extension points

USE CASE SPECIFICATION FOR UPDATING VOTER DETAILS

1.**Use Case-Name: Updating voter details**

Brief Description:

To make changes to voter profile by chief officer.

2. **List of Actors:**

1. Chief officer

2. System

3**. Initial Conditions:**

This requires the user must be a valid user and must be a registered user of the votingsystem, must have valid internet connection and must have vote registered in the voting system to update them.

4**. Normal Flow of Events:**

1. Opens voting's login page

2. Prompt for password.

3. Enter password.

4. Validates password.

5. This use case is initiated by chief officer selecting assigning profile to update.

6.System displays the lists of voters.

7.The chief officer selects the name of voter.

8. System list the voters.

9.Chief officer selects the relevant person.

10. System displays a list of voters not already updated their profile.

11. Chief officer highlights the voter's profile to be updated.

12. System presents a message confirming that member have been updated.

5**. Exceptional Flow of Events:**

1.Voters age must be above 18

2.Voter already exists.

3. Invalid user name or password.

4. Network failure.

5. Server error.

6**. Exit Conditions:**

1. Updating already existing member.

2. Closes App

7. **Extension Points:**

No extension points

ACTIVITY DIAGRAMS

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.

The following symbols are used on the activity diagram toolbox to model activity diagrams:

Activities:-

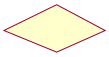
An activity represents the performance of some behavior in the workflow.

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.

Decision Points:-

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

Start state:

A start state explicitly shows 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

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.

End 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.

- Horizontal synchronization
- Vertical synchronization.

Fig 1- ACTIVITY DIAGRAM TO REGISTER FOR VOTE

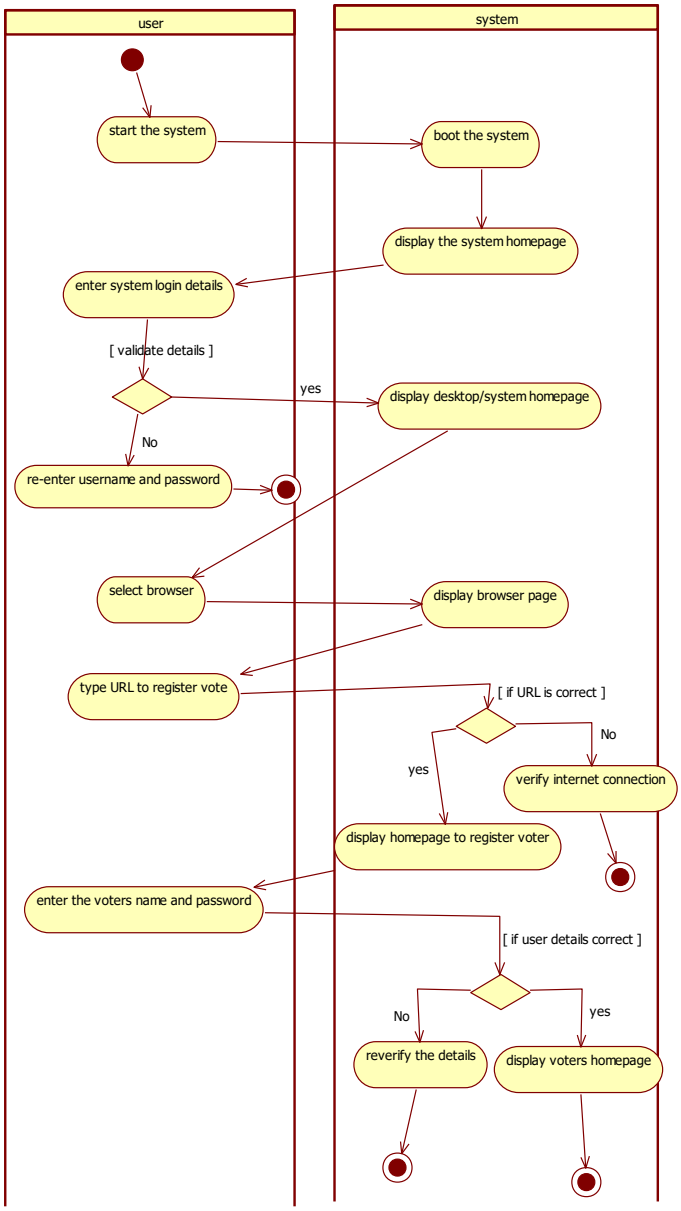


Fig 2- ACTIVITY DIAGRAM TO VALIDATE USER DETAILS

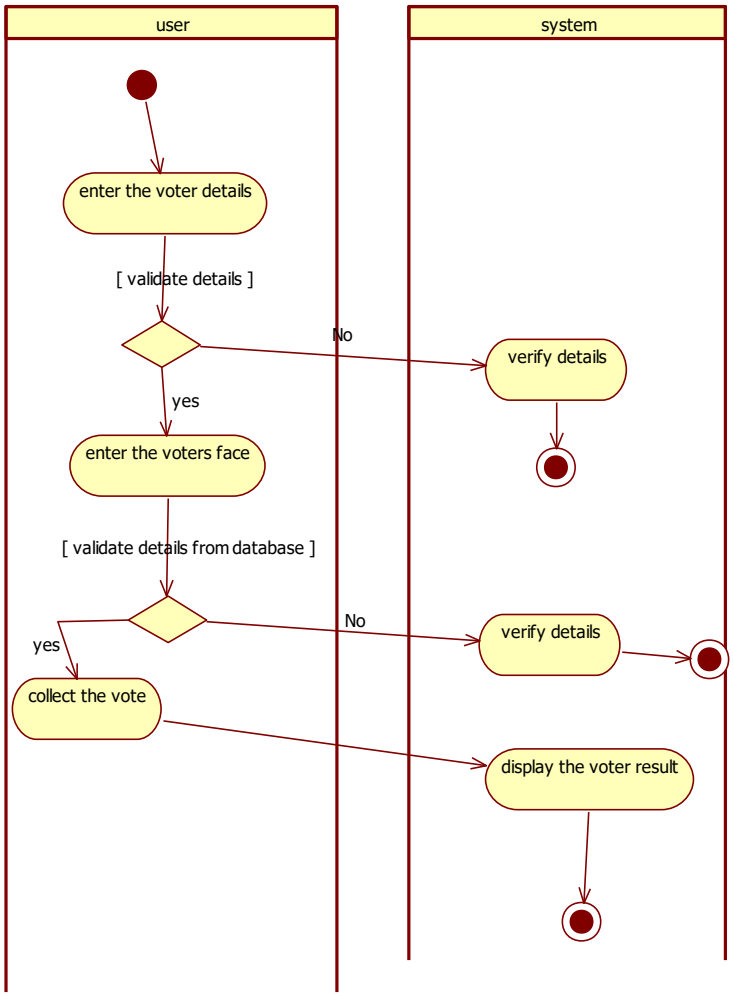
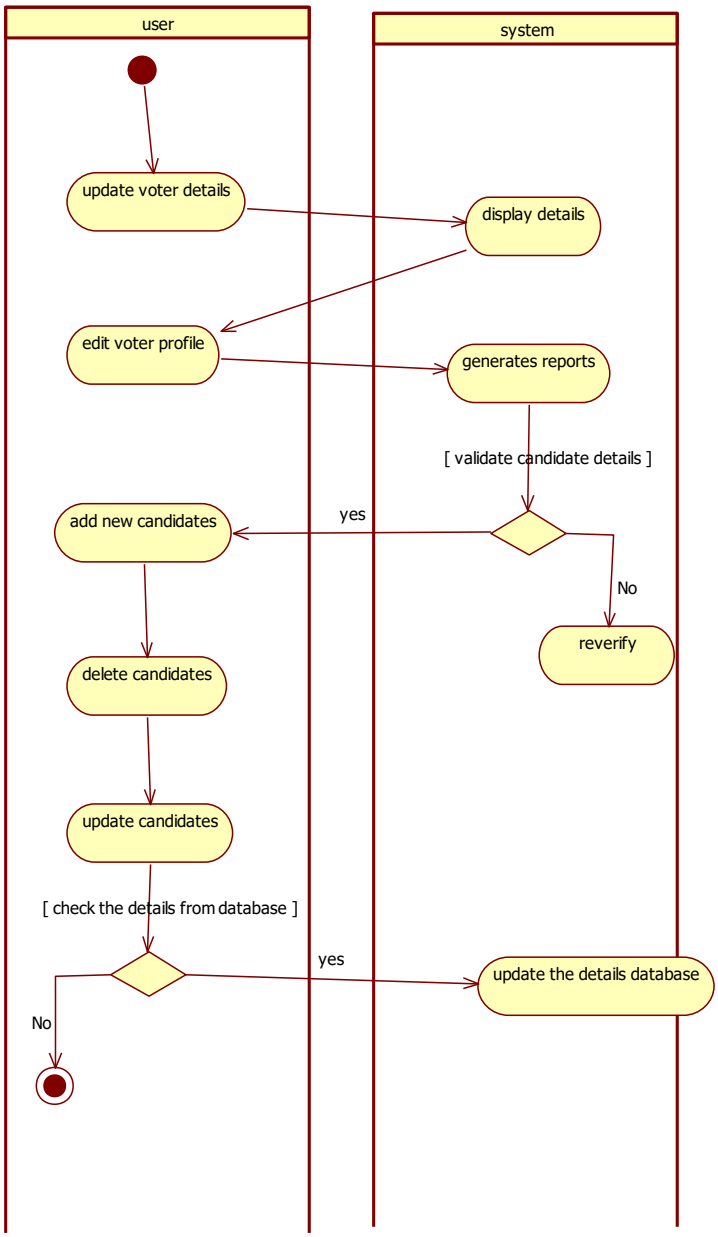


Fig 3- ACTIVITY DIAGRAM TO UPDATE THE VOTER DETAILS



LOGICAL VIEW

IDENTIFICATION OF ANALYSIS CLASSES

The class diagram is fundamental to object-oriented analysis. Through successive iterations, it provides both a high level basis for systems architecture, and a low-level basis for the allocation of data and behavior to individual classes and object instances, and ultimately for the design of the program code that implements the system. So, it is important to identify classes correctly. However, given the iterative nature of the object-oriented approach, it is not essential to get this right on the first attempt itself.

Approaches for identifying classes:

We have four alternative approaches for identifying classes:

1. The noun phrase approach;
2. The common class patterns approach;
3. The use- case driven to sequence/collaboration modeling approach;
4. Class Responsibility collaboration cards (CRC) approach.

1) NOUN PHRASE APPROACH:

In this method, analyst 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 .

It is safe to scrap the irrelevant classes, which either have no purpose or will be unnecessary. Candidate classes then are selected from the other two categories. Here identifying classes and developing a UML class diagram just like other activities is an iterative process.

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.
- All classes must make sense in the application domain; avoid computer implementation classes-defer them to the design stage.
- Carefully choose and define class names.

1. Selecting Classes from the Relevant and Fuzzy Categories:

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

a) Redundant classes.

Do not keep two classes that express the same information. If more than one word is being used to describe the same idea, select the one that is the most meaningful in the context of the system. This is part of building a common vocabulary for the system as a whole. Choose your vocabulary carefully; use the word that is being used by the user of the system.

b) Adjectives classes:

"Be wary of the use of adjectives. Adjectives can be used in many ways. An adjective can suggest a different kind of object, different use of the same object, or it could be utterly irrelevant. Does the object represented by the noun behave differently when the adjective is applied to it? If the use of the adjective signals that the behavior of the object is different, and then makes a new class".

For example : Single account holders behave differently than Joint account holders, so the two should be classified as different classes.

c) Attribute classes:

Tentative objects that are used only as values should be defined or restated as attributes and not as a class. For example, Client Status and Details of Client are not classes but attributes of the Client class.

2) COMMON CLASS PATTERNS 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:

a) Concept Class:

A concept is a particular idea or understanding that we have of our world. The concept class encompasses principles that are not tangible but used to organize or keep track of business activities or communications.

Example: Performance is an example of concept class object.

b) Events Class:

Events classes are points in time that must be recorded. Things happen, usually to something else at a given date and time or as a step in an ordered sequence. Associated with things remembered are attributes (after all, the things to remember are objects) such as who, what, when, where, how, or why.

Example: Landing, interrupt, request, and order are possible events.

c) Organization Class:

An organization class is a collection of people, resources, facilities, or groups to which the users belong; their capabilities have a defined mission, whose existence is largely independent of the individuals.

Example: An accounting department might be considered a potential class.

d) People class (also known as person, roles, and roles played class):

The people class represents the different roles users play in interacting with the application.

Example: Employee, client, teacher, and manager are examples of people.

3)USE-CASE DRIVEN APPROACH:

Identifying Classes And Their Behavior Through Sequence/Collaboration Modeling

One of the first steps in creating a class diagram is to derive from a use case, via a collaboration (or collaboration diagram), those classes that participate in realizing the use case. Through further analysis, a class diagram isdeveloped for each use case and the various use case class diagrams are then usually assembled into a larger analysis class diagram.

This can be drawn first for a single subsystem or increment, but class diagrams can be drawn at any scale that is appropriate, from a single use case instance to a large, complex system.

Identifying the objects involved in collaboration can be difficult at first, and takes some practice before the analyst can feel really comfortable with the process. Here collaboration (i.e. the set of classes that it comprises) can be identified directly for a use case, and that, once the classes are known, the next step is to consider the interaction among the classes and so build a collaboration diagram.

From collaboration diagram to classdiagram:

The next step in the development of a requirements model is usually to produce a class diagram that corresponds to each of the collaboration diagrams.

Collaboration diagrams are obtained by result of reasonably careful analysis, the transition is not usually too difficult.

The similarities & differences between Collaboration and class diagrams are :

First, consider the similarities:

Both show class or object symbols joined by connecting lines. **In** general, a class diagram has more or less the same structure as the corresponding collaboration diagram. **In** particular, both should show classes or objects of the same types.

Any of the three analysis stereotype notations for a class can be used on either diagram, and stereotype labels can also be omitted from individual classes, or from an entire diagram.

Next, the differences are:

1. The difference is that an actor is almost always shown on a collaboration diagram, but not usually shown on a class diagram. This is because the collaboration diagram represents a particular interaction and the actor is an important part of this interaction. However, a class diagram shows the more enduring structure of associations among the classes, and frequently supports a number of different interactions that may represent several different use cases.
2. A collaboration diagram usually contains only object instances, while a class diagram usually contains only classes.
3. The connections between the object symbols on a collaboration diagram symbolize links between objects, while on a class diagram the corresponding connections stand for associations between classes.
4. A collaboration diagram shows the dynamic interaction of a group of objects and thus every link needed for message passing is shown. The labeled arrows alongside the links represent messages between objects. On a class diagram, the associations themselves are usually labeled, but messages are not shown.
5. Finally, any of the three stereotype symbols can be used on either diagram, there are also differences in this notation.

4) 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.

IDENTIFICATION OF RESPONSIBILITIES OF CLASSESS

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 aid to group role-playing activity. Index cards are used in preference to pieces of paper due to their robustness and to limitations that their size (approx. 15cm x 8cm) imposes on number of responsibilities and collaborations that effectively allocated to each class.

A class name is entered at the top of each card and responsibilities and collaborations are listed underneath as they become apparent. For the sake of clarity, each collaboration is normally listed next to the corresponding responsibility.

From a UML perspective, use of CRC cards is in analyzing the object interaction that is 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.

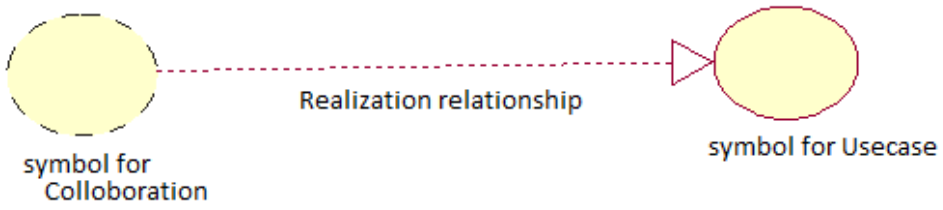
Before beginning a CRC session it is important that all team members are briefed on the organization of the session and a CRC session should be preceded by a separate exercise that identifies all the classes for that part of the application to be analyzed. The team members to whom these classes are allocated can then prepare for the role playing exercise by considering in advance a first-cut allocation of responsibilities and identification of collaborations. Here , it is important to ensure that the environment in which the sessions take place is free from interruptions and free for the flow of ideas among team members.

During a CRC card session, there must be an explicit strategy that helps to achieve an appropriate distribution of responsibilities among the classes. One simple but effective approach is to apply the rule that each object should be as lazy as possible, refusing to take on any additional responsibility unless instructed to do so by its fellow objects.

USE CASE REALIZATIONS

. A scenario is an instance of a use case it is one path through the flow of events for the use case. The use case diagram presents an outside view of the system. The functionality of the use case is captured in the flow of events. Scenarios are used to describe how use cases are realized as interactions among group of objectsScenarios are developed to help identify the objects, the classes, and the object interactions needed to carry out a piece of the functionality specified by the use case. They also provide an excellent communication medium to be used in the discussion of the system requirements with customers.

In the UML, use case realizations are drawn as dashed ovals and relationship symbols are as shown below.



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

SEQUENCE DIAGRAMS

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 life line 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.

There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other. A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

ELEMENTS OF SEQUENCE DIAGRAM

- Objects
- Links
- Messages
- Focus of control
- Object life line

Fig 1- SEQUENCE DIAGRAM FOR VALIDATING FACE

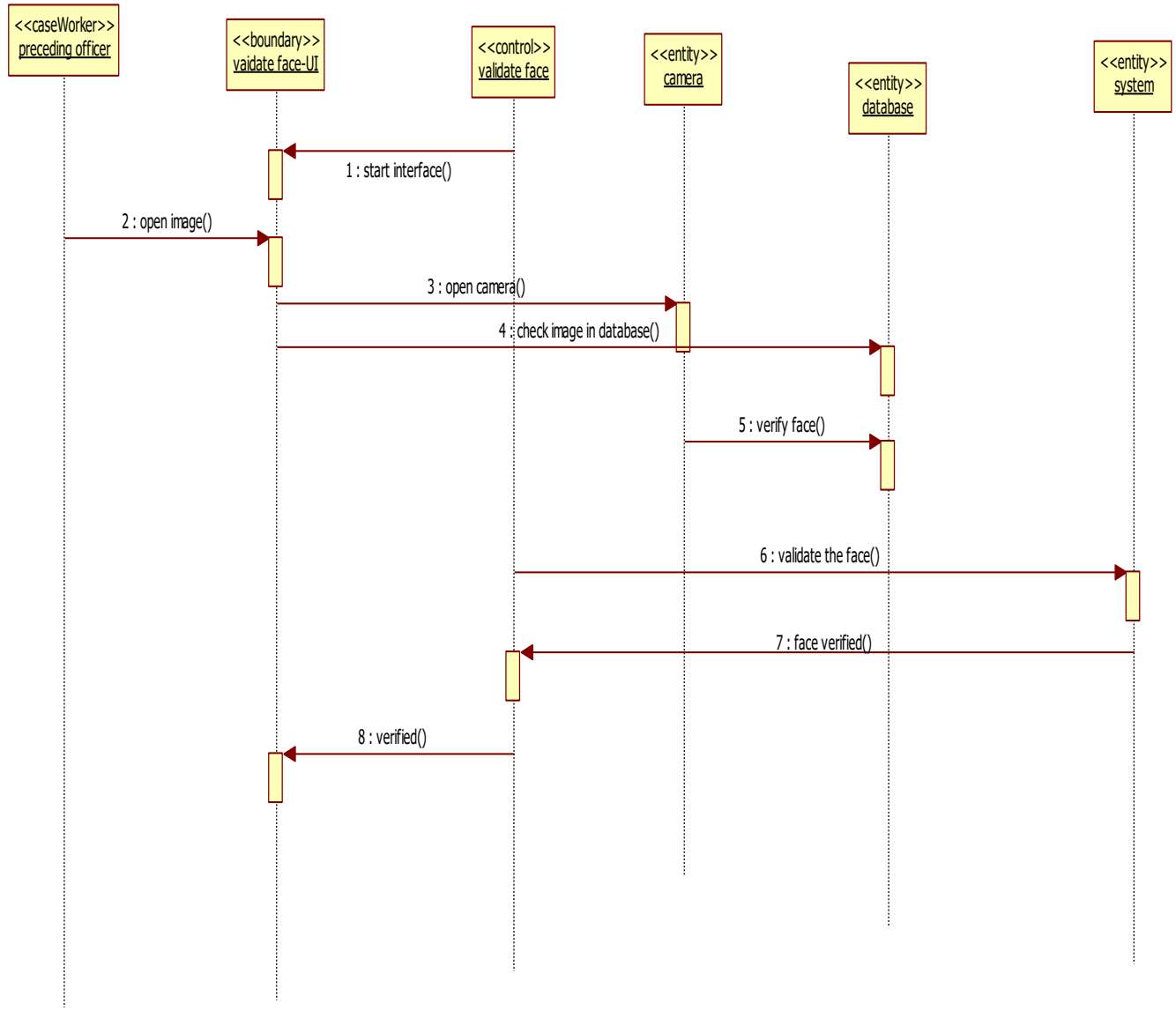


Fig 2- SEQUENCE DIAGRAM TO REGISTER FOR VOTE

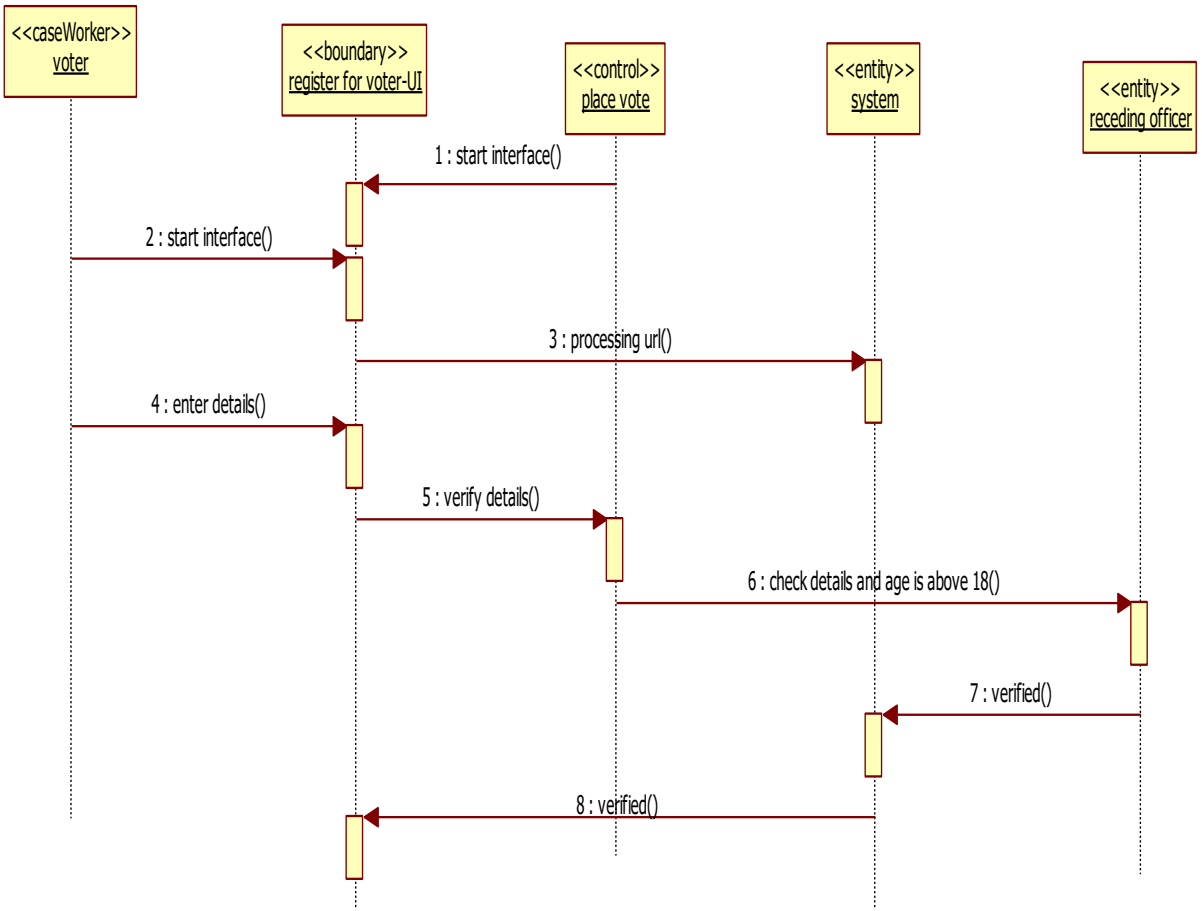
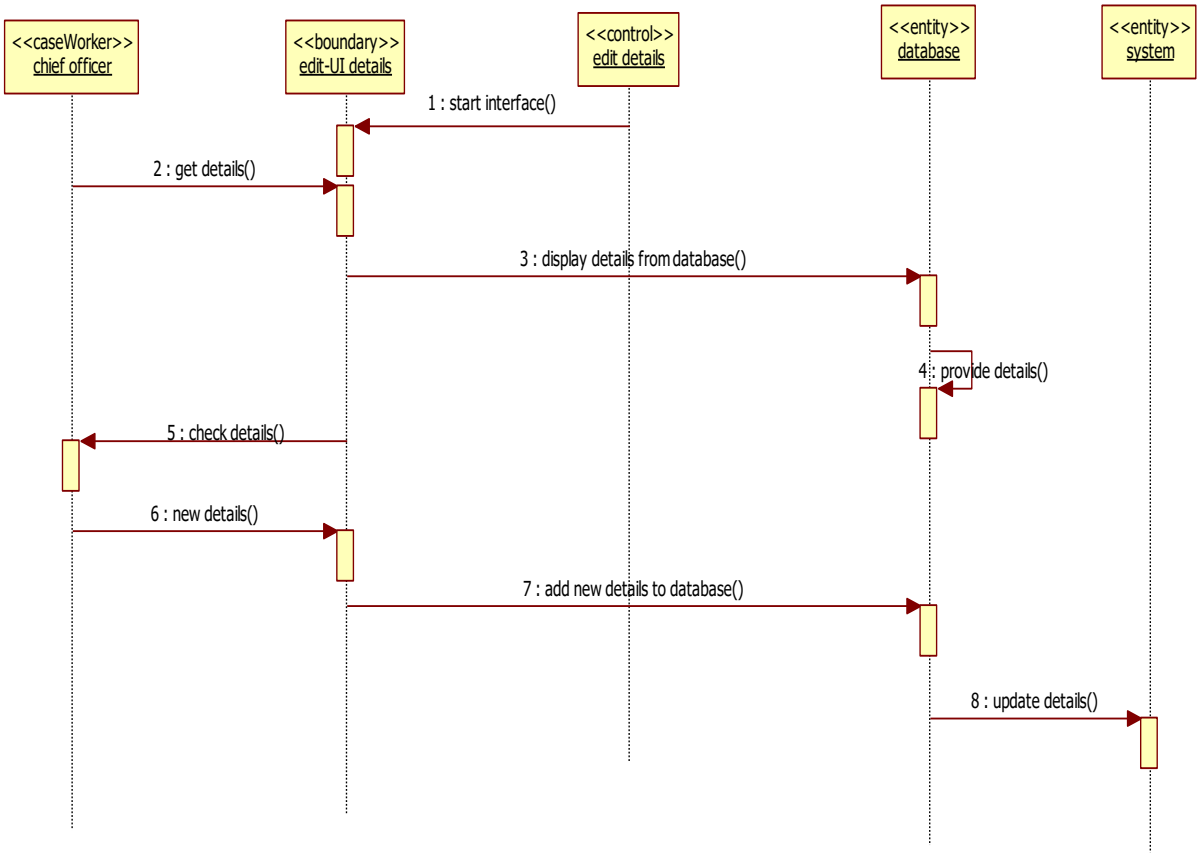


Fig 3- SEQUENCE DIAGRAM TO UPDATE VOTER PROFILE



COLLABORATION DIAGRAM

Collaboration diagrams are the second kind of interaction diagram in the UML diagrams. They are used to represent the collaboration that realizes a use case.The most significant difference between the two types of interaction diagram is that a collaboration diagram explicitly shows the links between the objects that participate in a collaboration, as in sequence diagrams, there is no explicit time dimension.

Message labels in collaboration diagrams:

Messages on a collaboration diagram are represented by a set of symbols that are the same as those used in a sequence diagram, but with some additional elements to show sequencing and recurrence as these cannot be inferred from the structure of the diagram. Each message label includes the message signature and also a sequence number that reflects call nesting, iteration, branching, concurrency and synchronization within the interaction.

The formal message label syntax is as follows:

[predecessor] [guard-condition] sequence-expression [return-value ':='] message-name' (' [argument-list] ')'

A predecessoris a list of sequence numbers of the messages that must occur before the current message can be enabled. This permits the detailed specification of branching pathways. The message with the immediately preceding sequence number is assumed to be the predecessor by default, so if an interaction has no alternative pathways the predecessor list may be omitted without any ambiguity. The syntax for a predecessor is as follows:

sequence-number { ',' sequence-number} 'I'

The 'I'at the end of this expression indicates the end of the list and is only included when an explicit predecessor is shown.

Guard conditionsare written in Object Constraint Language (OCL),and are only shown where the enabling of a message is subject to the defined condition. A guard condition may be used to represent the synchronization of different threads of control.

A sequence-expressionis a list of integers separated by dots ('.') optionally followed by a *name* (a single letter), optionally followed by a *recurrence* term and terminated by a colon. A sequence-expression has the following syntax:

integer { '.' integer } [name] [recurrence] ':'

In this expression *integer* represents the sequential order of the message. This may be nested within a loop or a branch construct, so that, for example, message 5.1 occurs after message 5.2 and both are contained within the activation of message 5.

The *name* of a sequence-expression is used to differentiate two concurrent messages since these are given the same sequence number. For example, messages 3.2.1a and 3.2.1b are concurrent within the activation of message 3.2.

Recurrence reflects either iterative or conditional execution and its syntax is as follows:

Branching*:* '['condition-clause‘] ,

Iteration*:* ‘ * ‘ ‘ [‘ iteration-clause ‘] '

Elements:

- Objects
- Links
- Messages
- Path

Fig 1- COLLABORATION DIAGRAM FOR VALIDATING FACE

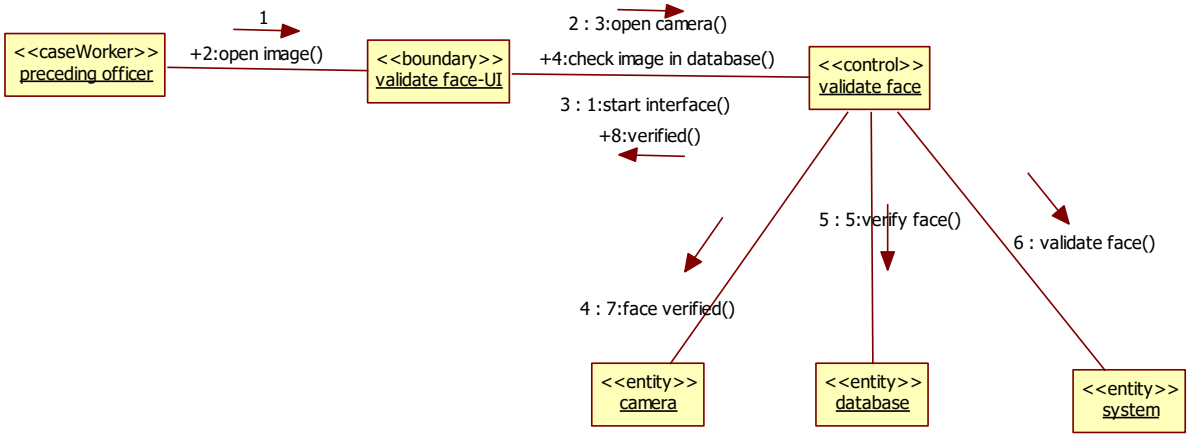


Fig 2- COLLABORATION DIAGRAM TO REGISTER FOR VOTE

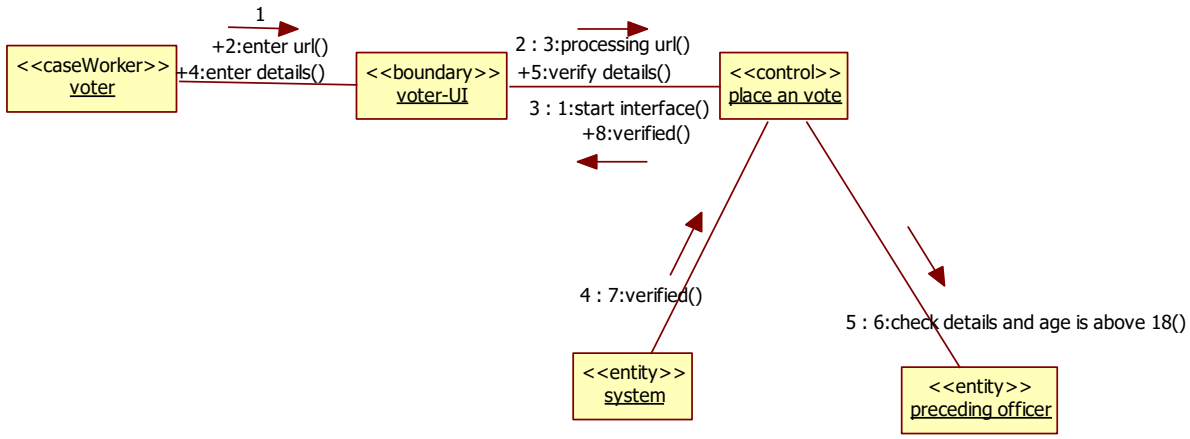
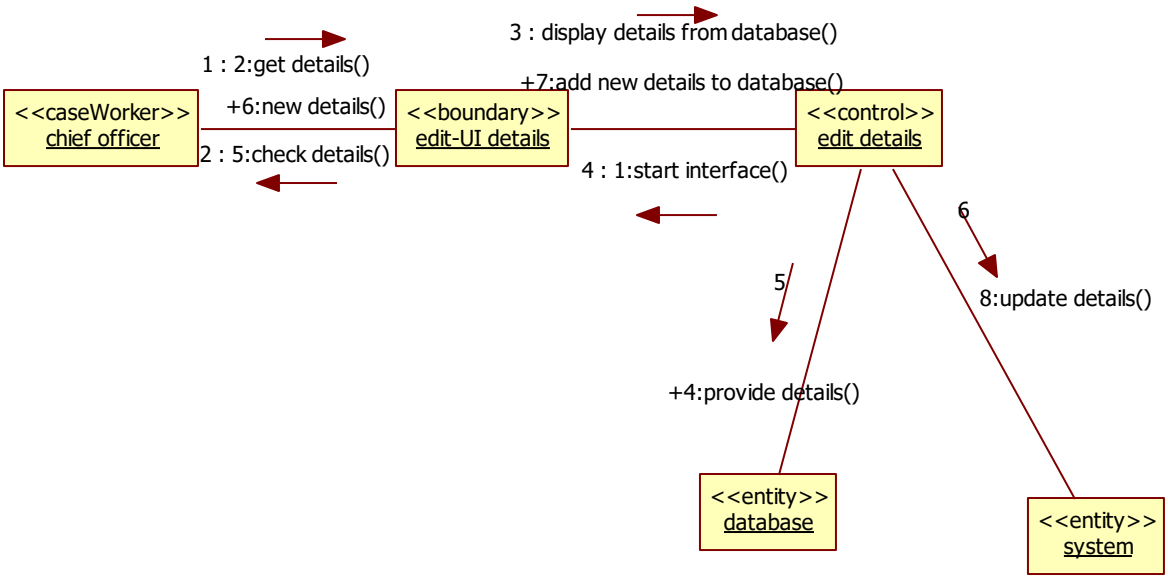


Fig 3- COLLABORATION DIAGRAM TO UPDATE VOTER PROFILE



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
- Keep the class simple; state only enough attribute to defineobject state.
- Attributes are less likely to be fully described in the problem statement.
- Omit derived attributes.
- Do not carry discovery attributes to excess.

The attributes identified in our system are:

- Attributes for User class: name, password
- Attributes for account holder details class: name, balance, password
- Attributes for bill details class: bill no, total amount
- Attributes for cheque book details class: cheque book number

The responsibilities identified in our system are:

- Methods for User class: Successful login/failure, display user statements
- Methods for account holder details class: request for details, get transaction details.
- Methods for cheque services class: get details, stop services.
- Methods for bill details class: bill no, total amount.

IDENTIFICATION OF RELATIONSHIPS AMONG CLASSES

NEED FOR RELATIONSHIPS AMONG CLASSES:

All systems are made up of many classes and objects. System behavioris achieved through the collaborations of the objects in the system.

Two types of relationships in Class diagram are:

1. Association Relationship
2. Aggregation Relationship

1. Association Relationship:

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.

2. Aggregation Relationship:

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).

3. Super-sub structure (Generalization Hierarchy):

These allow objects to be build from other objects. The super-sub class hierarchy is a relationship between classes, where one class is the parent class of another class.

NAMING RELATIONSHIP:

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 (e.g., Buses are allotted to Routes). 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 INDICATORS:

- 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
- 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)

UML CLASS DIAGRAM

- 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.

Class diagrams contain icons representing classes, packages, interfaces, and their relationships, you can create one or more class diagrams to depict the classes at the top level of the current model; such diagrams are themselves contained by the top level of the current model.

OBJECT:

- An object is a representation of an entity, either real-world or conceptual.
- An object is a concept, abstraction, or thing with well defined boundaries and meaning for an application.
- Each object in a system has three characteristics: state, behaviour, identity

STATE:

The state of an object is one of the possible conditions in which it may exist. The state of an object typically changes over time, and is defined by a set of properties, with the values of the properties, plus the relationships the object may have with other objects.

BEHAVIOR:

- Behavior determines how an object responds to request from other objects.
- Behavior is implemented by the set of operations for the object.

IDENTITY:

- Identity means that each object is unique even if its state is identical to that of another object.

CLASS:

A class is a description of a group of objects with common properties common behaviour, 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.
- The bottom compartment contains the behavior of the class.

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. Some common stereotypes for a class are entity, boundary and control class.

Entity classes:

Entity classes are used to model 'information and associated behavior of some phenomenon or concept such as an individual, a real-life object, or a real-life event'. As a general rule, entity classes represent something within the application domain, but external to the software system, about which the system must store some information.

Instances of an entity class will often require persistent storage of information about the things that they represent. This can sometimes help to decide whether an entity class is the appropriate modeling construct.

Boundary classes:

Boundary classes, it is a 'model interaction between the system and its actors'. Since they are part of the requirements model, boundary classes are relatively abstract. They do not directly represent all the different sorts of interface that will be used in the implementation language. The design model may well do this later, but from an analysis perspective we are interested only in identifying the main logical interfaces with users and other systems.

Control classes:

Control classes 'represent coordination, sequencing, transactions and control of other objects' .In the USDP, as in the earlier methodology objectory, it is generally recommended that there should be a control class for each use case.

In a sense, then, the control class represents the calculation and scheduling aspects of the logic of the use case at any rate, those parts that are not specific to the behavior of a particular entity class, and that are specific to the use case. Meanwhile the boundary class represents interaction with the user and the entity classes represent the behavior of things in the application domain and storage of information that is directly associated with those things

Fig 1-CLASS DIAGRAM TO REGISTER FOR VOTE:

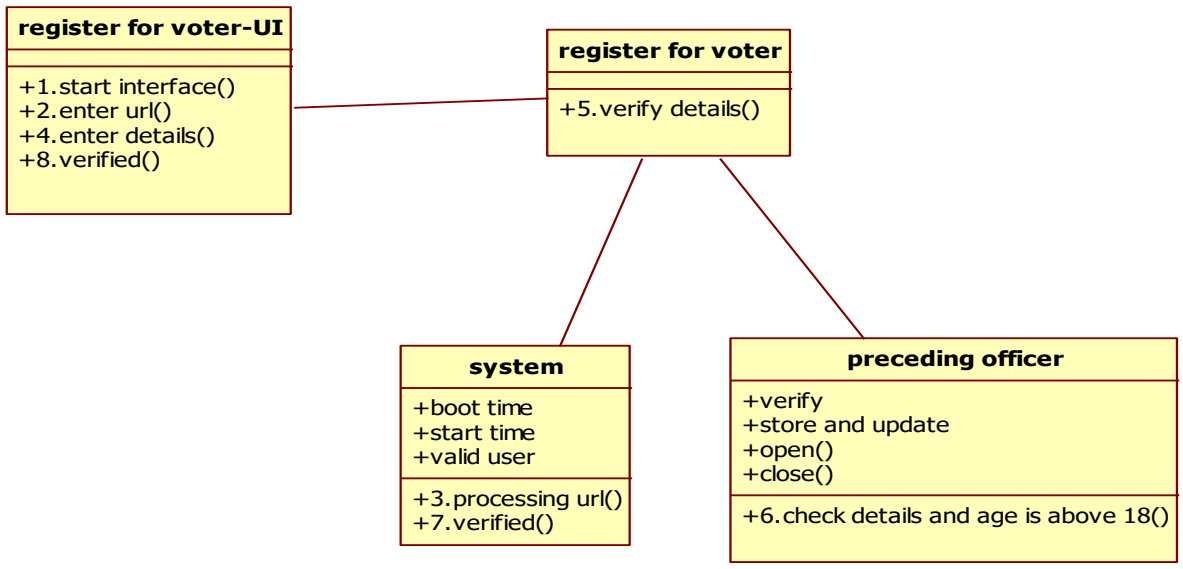


Fig 2-CLASS DIAGRAM TO VALIDATE FACE:

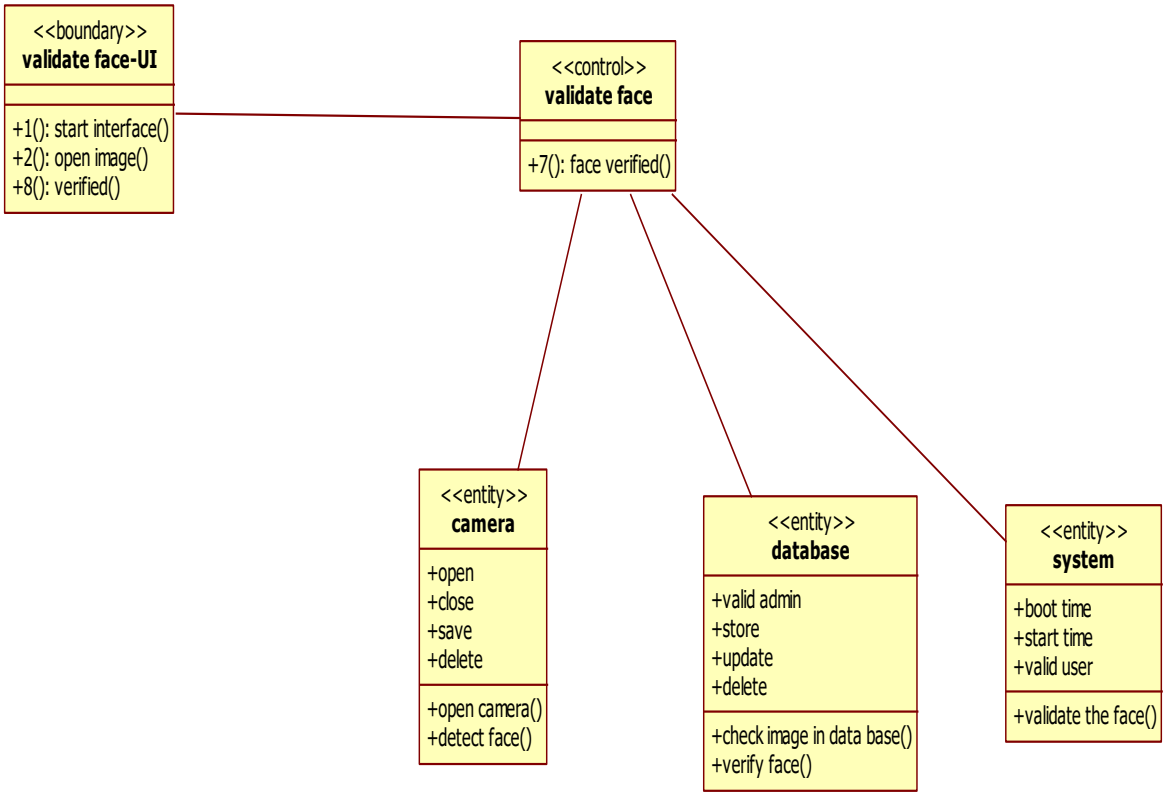
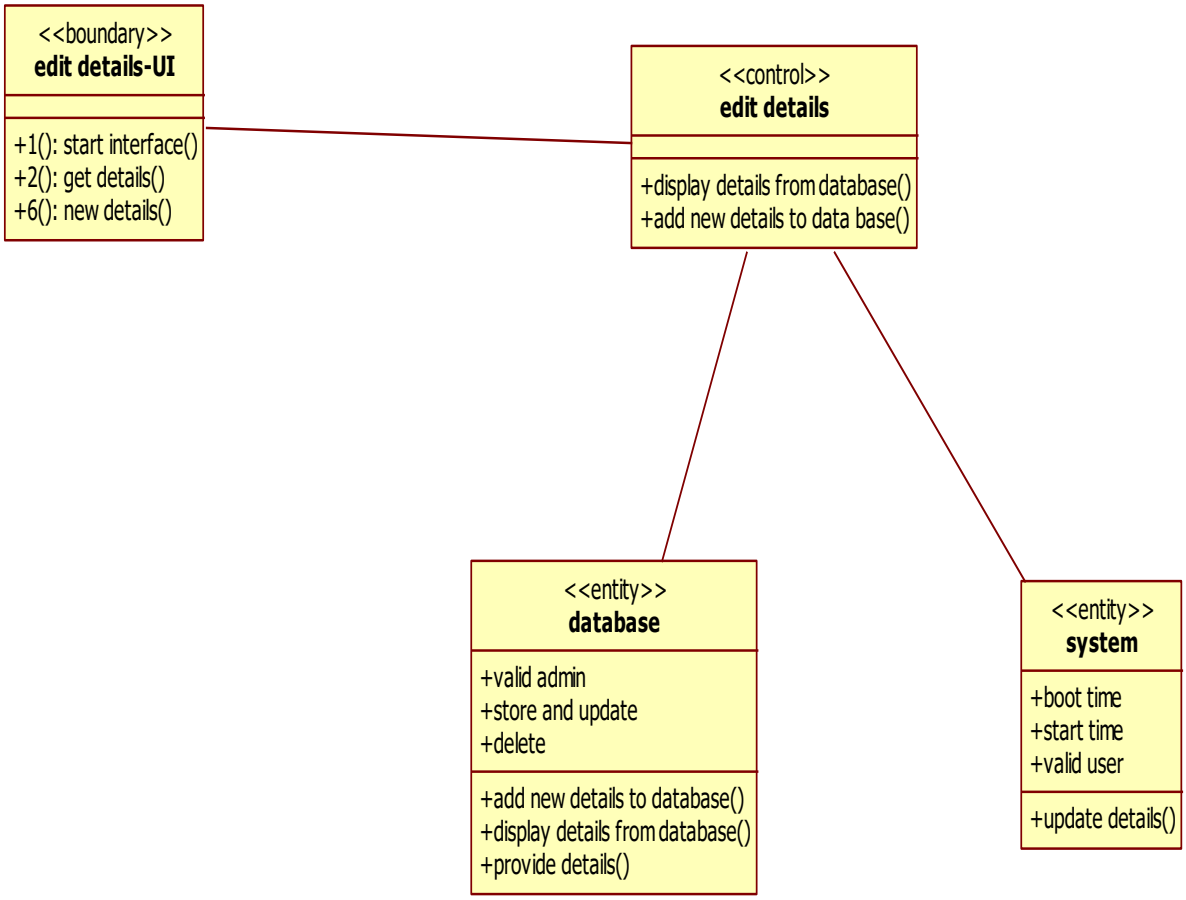


Fig 3- CLASS DIAGRAM TO UPDATE VOTER PROFILE



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. Sometimes it is necessary to consider inside behavior of an object.

A state chart diagram shows the **states** of a single object, the events or messages that cause a **transition** from one state to another and the **actions** that result from a state change. As in 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 class objects with significant behavior.

State chart diagrams are closely related to activity diagrams. The main difference between the two diagrams is state chart diagrams are state centric, while activity diagrams are activity centric. A state chart diagram is typically used to model the discrete stages of an object's lifetime, whereas an activity diagram is better suited to model the sequence of activities in a process.

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 it's better to concentrate on sequence diagram. In an ESU the object for Course Offering 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.

Actions:

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. You 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.

Transitions are labeled with the following syntax:

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

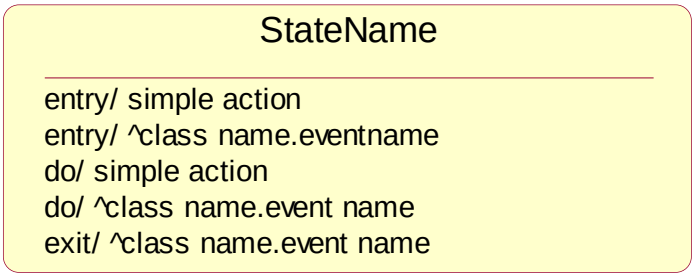
Only one event is allowed per transition, and one action per event.

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 diagram:

- State chart diagrams are used to model dynamic view of a system.
- State chart diagrams are used to modeling lifetime of an object.
- State chart diagrams are used to focus on the changing state of a system driven by events.
- It will also be used when showing the behavior of a class over several use cases.

ELEMENTS OF STATE CHART DIAGRAMS

1. **State:-**

   It is a condition or situation during the life of an object during which it satisfies some conditions, performs some activity, or waits for some event

2. **Event:-**

   It is the specification of significant occurrence that has a location in time and space.

3. **Transition:-**

   It is a relation between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and conditions are satisfied.

4. **Action state:-**

   An action state is shorthand for a state with an entry action and at least one outgoing transition involving the implicit event of completing the entry action.

5. **Initial state:-**

   A pseudo state to establish the start of the event into an actual state.

6. **Final state:-**

   The final state symbol represents the completion of the activity

7. **Concurrent sub state:-**

   A concurrent state is divided into two or more sub states. It is a state that contains other state vertices. Any state enclosed with in a composite state is called a sub state of that concurrent state.

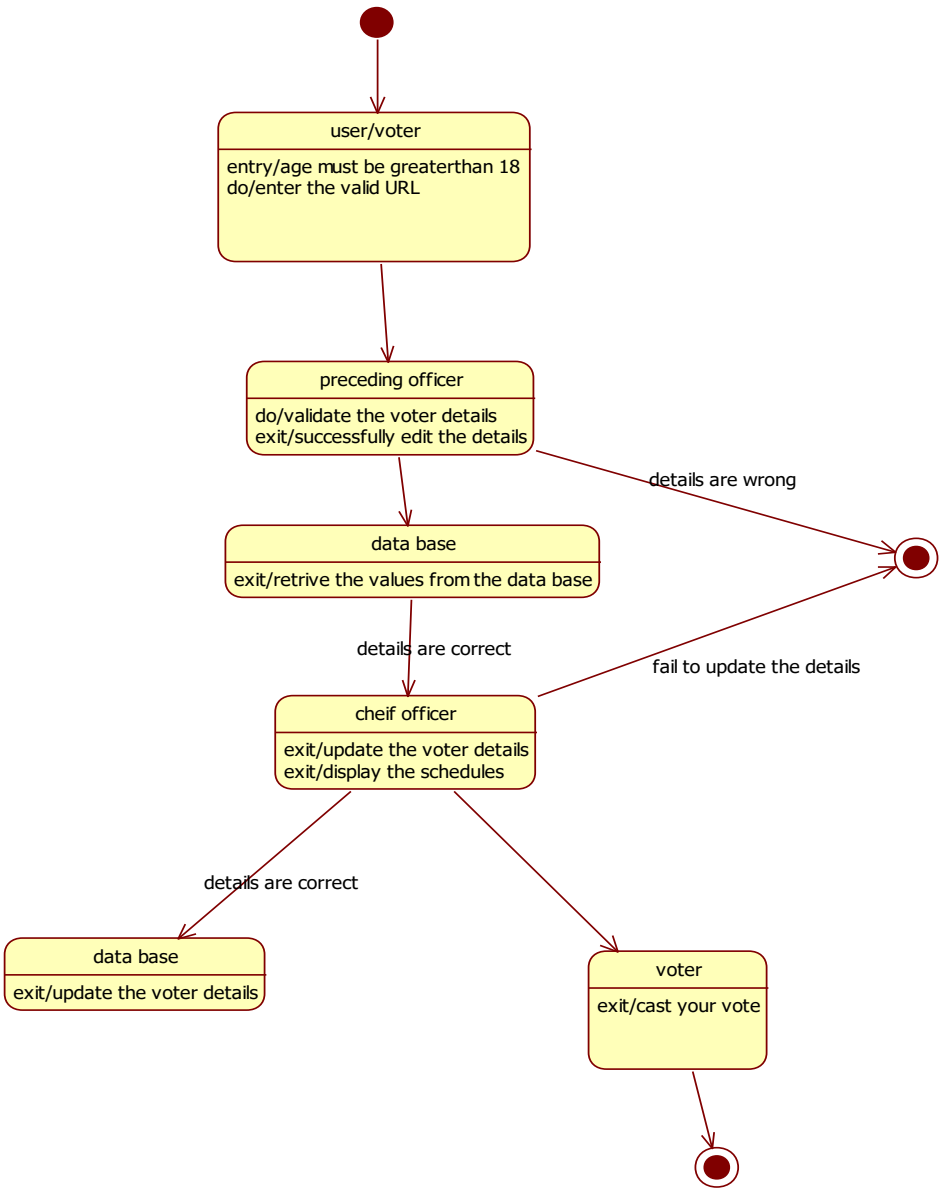
8. **Guard conditions:-**

   Activity and state diagrams express a decision when conditions are used to indicate different possible transitions that depend on Boolean conditions of container object. UML calls those conditions as guard conditions.

9. **Forks and joins:-**

   A fork construct is used to model a single flow of control that divides into two or more separate, but simultaneous flows. A join consists of two or more flows of control that unite into a single flow of control.

STATE CHART DIAGRAM FOR VOTING SYSTEM



DESIGN

DESIGNING CLASSES BY APPLYING DESIGN AXIOMS

Coupling and cohesion :

The factors coupling and cohesion are important factors for good design.

Coupling describes the degree of interconnectedness between design components and is reflected by the number of links an object has and by the degree of interaction the object has with other objects.

Cohesion is a measure of the degree to which an element contributes to a single purpose. The concepts of coupling and cohesion are not mutually exclusive but actually support each other. This criterion can be used within object-orientation as described below.

Interaction Coupling:

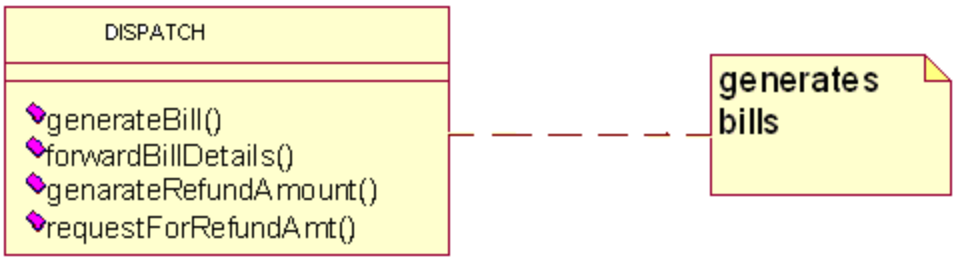
It is a measure of the number of message types an object sends to other objects and the number of parameters passed with these message types. Interaction coupling should be kept to a minimum to reduce the possibility of changes rippling through the interfaces and to make reuse easier. When an object is reused in another application it will still need to send these messages and hence needs objects in the new application that provide these services. This complicates the reuse process as it requires groups of classes to be reused rather than individual classes.

Inheritance Coupling:

describes the degree to which a subclass actually needs the features it inherits from its base class.

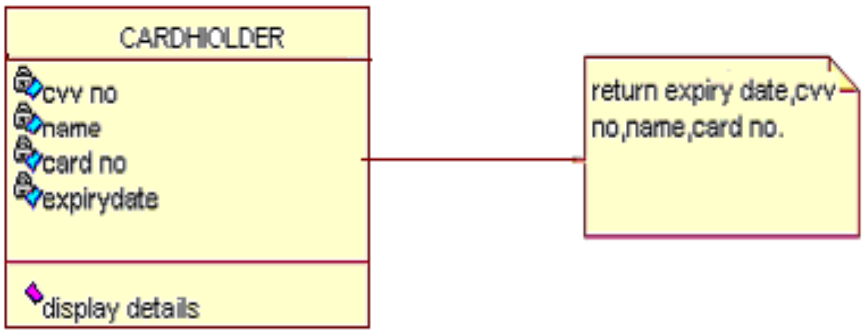
Operation Cohesion:

It measures the degree to which an operation focuses on a single functional requirement. Good design produces highly cohesive operations, each of which deals with a single functional requirement. For example in the following figure, the operation generate Bills() is highly cohesive.



Class Cohesion:

It reflects the degree to which a class is focused on a single requirement. The class Lecturer in the figure below exhibits low levels of cohesion as it has three attributes (roomNumber, roomLength and roomWidth and one operation calculate RoomSpace ()) that would be more appropriate in a class Room. The class Lecturer should only have attributes that describe a Lecturer object (e.g. lecturerName and lecturerAddress) and operations that use them.



Specialization Cohesion:-

It addresses the semantic cohesion of inheritance hierarchies. For example in the following figure all the attributes and operations of the Address base class are used by the m derived classes - this hierarchy has high inheritance coupling. However, it is neither true that a person is a kind of address nor that a company is a kind of address. The example is only using inheritance as a syntactic structure for sharing attributes and operations. This structure has low specialization cohesion and is poor design. It does not reflect meaningful inheritance in the problem domain.

REFINING ATTRIBUTES, METHODS & 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 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.

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

The name is the attribute name, the type-expression is its data type, the initial value is 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**

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.

IMPLEMENTATION DIAGRAMS

COMPONENT DIAGRAMS

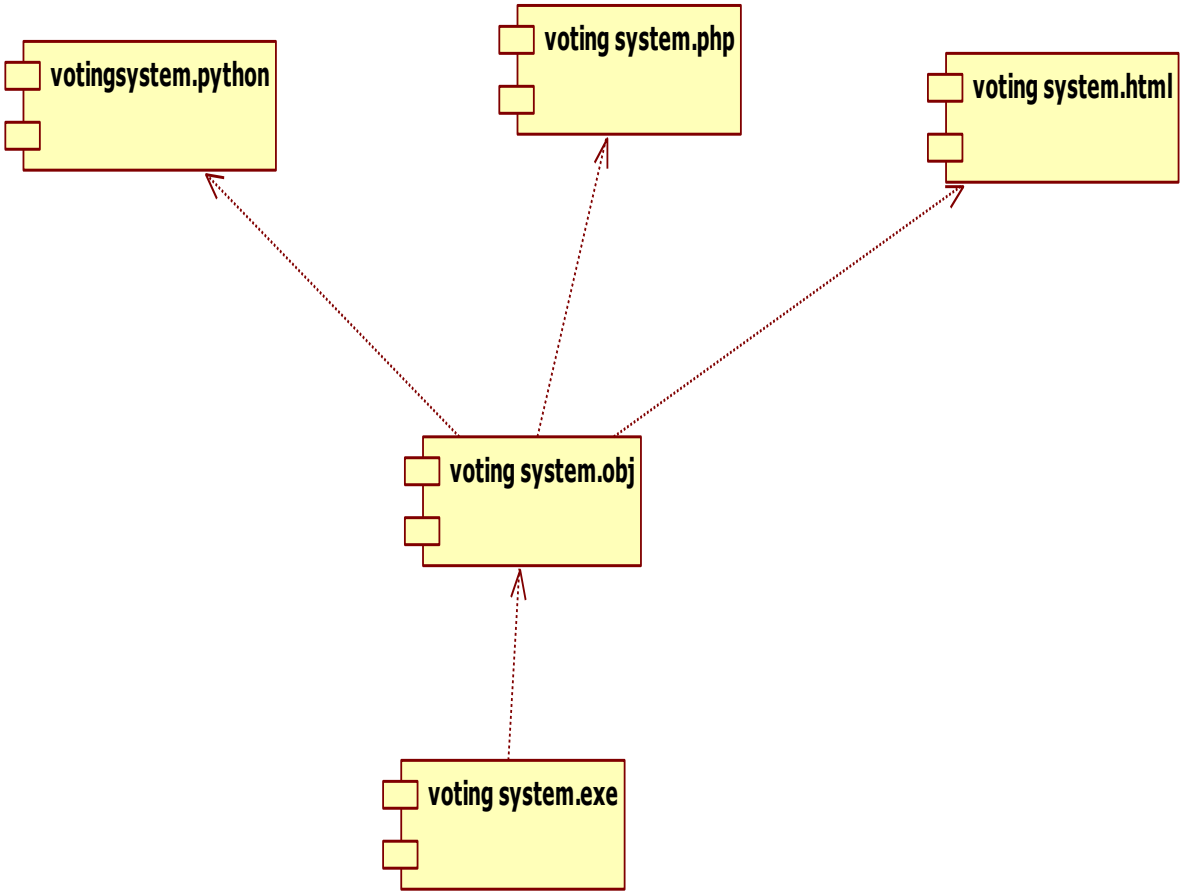
In a large project there will be many files that make up the system. These files will have dependencies on one another. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time, at link-time or at run-time. There are also dependencies 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.

Fig 1- COMPONENT DIAGRAM FOR VOTING SYSTEM



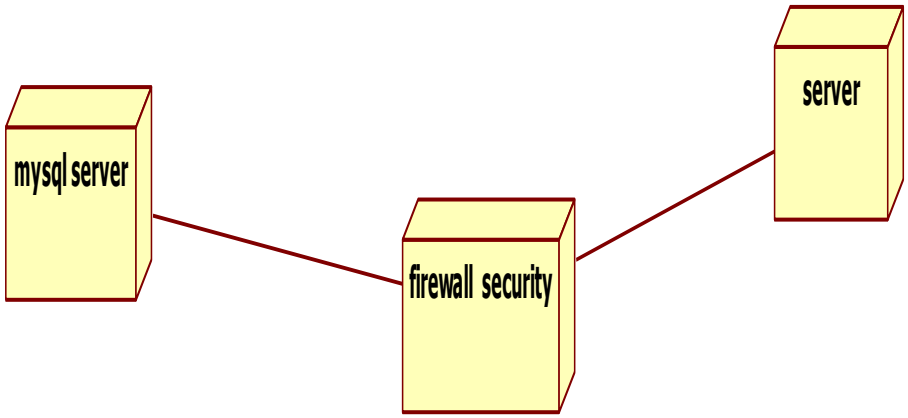
DEPLOYMENT DIAGRAM

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.

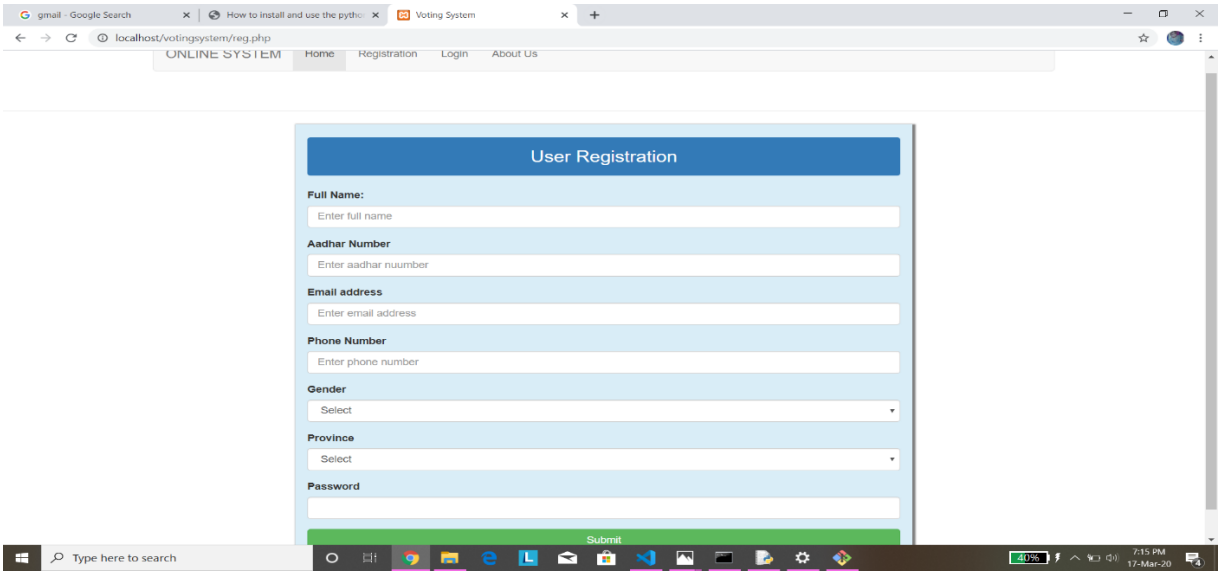
Nodes are drawn as 3D views of cubes or rectangular prisms, and the following figure shows a simplest deployment diagram where the nodes connected by communication associations.

DEPLOYMENT DIAGRAM FOR SERVER IN VOTING SYSTEM

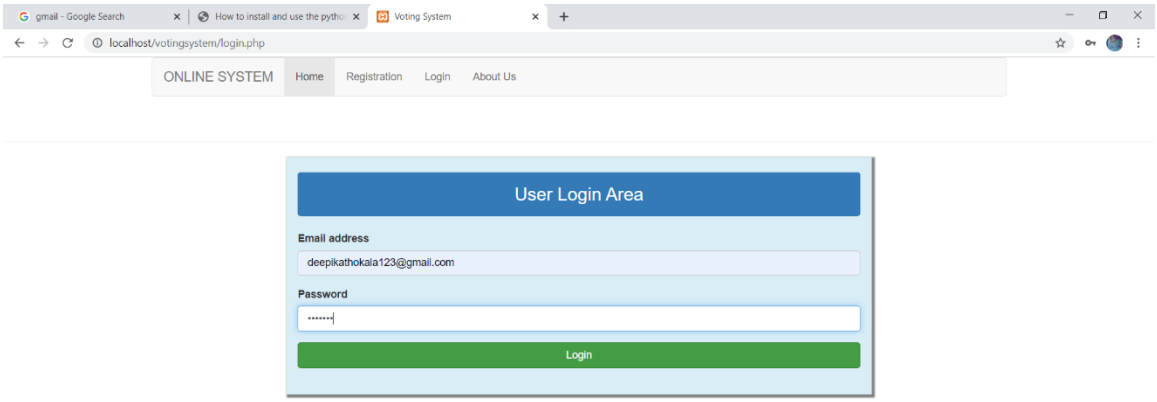


GETTING INKED IS THE SYMBOL OF CASTING THE VOTE

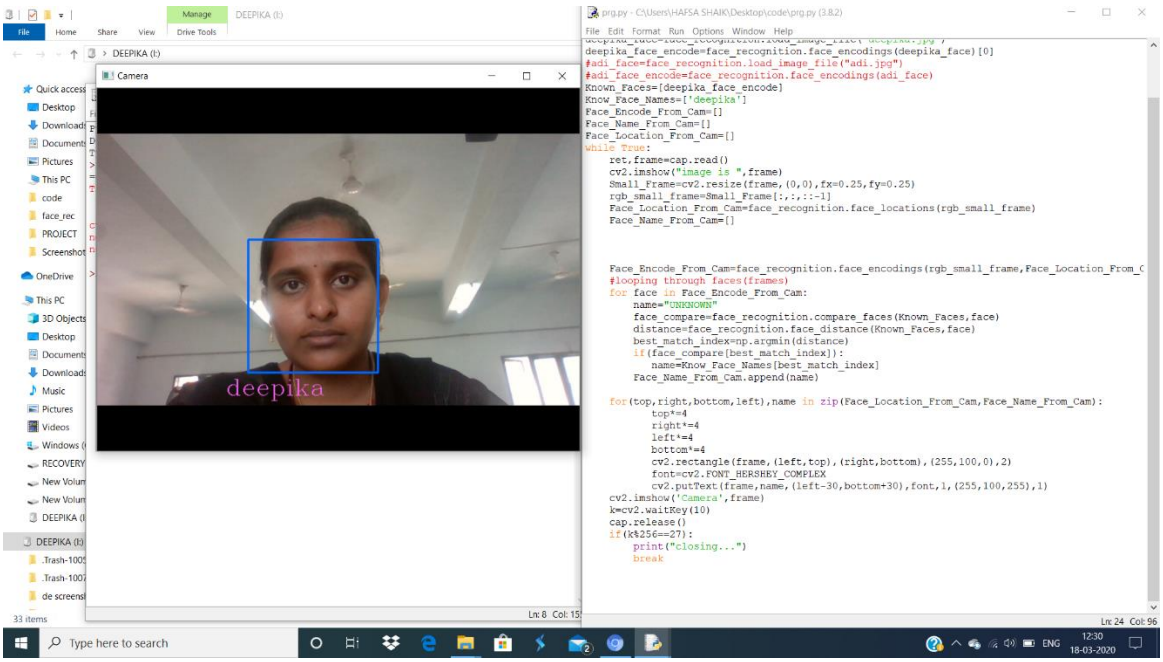
REGISTRATION FORM TO REGISTER FOR VOTE



OPENING THE WEB PAGE TO VOTE

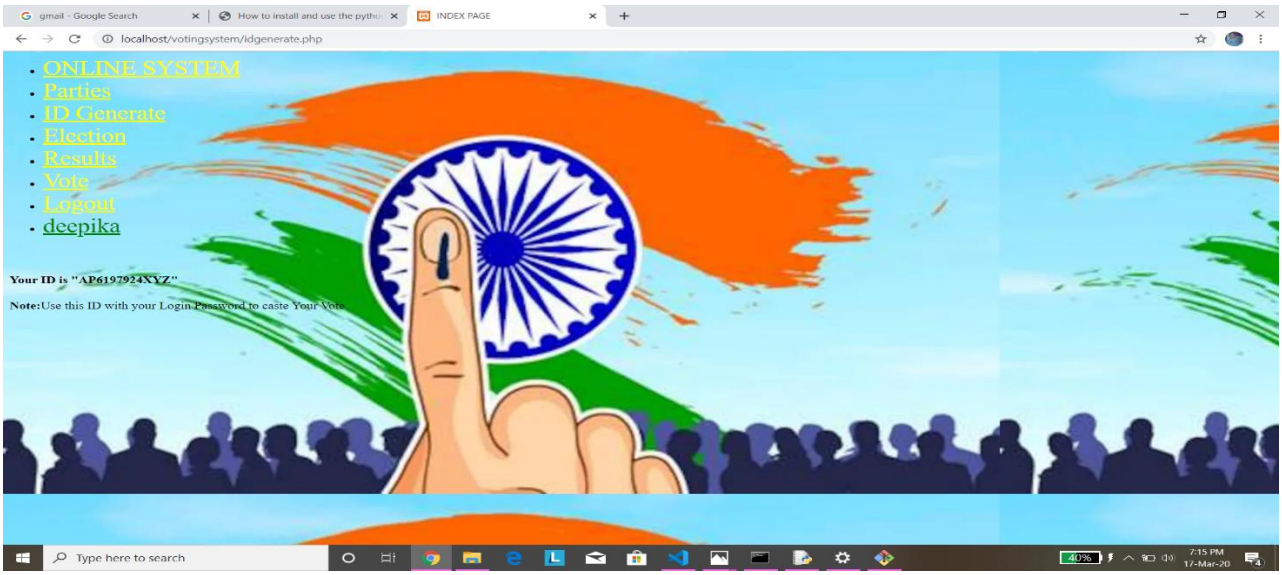


AFTER SUCCESSFUL FACE RECOGNITION

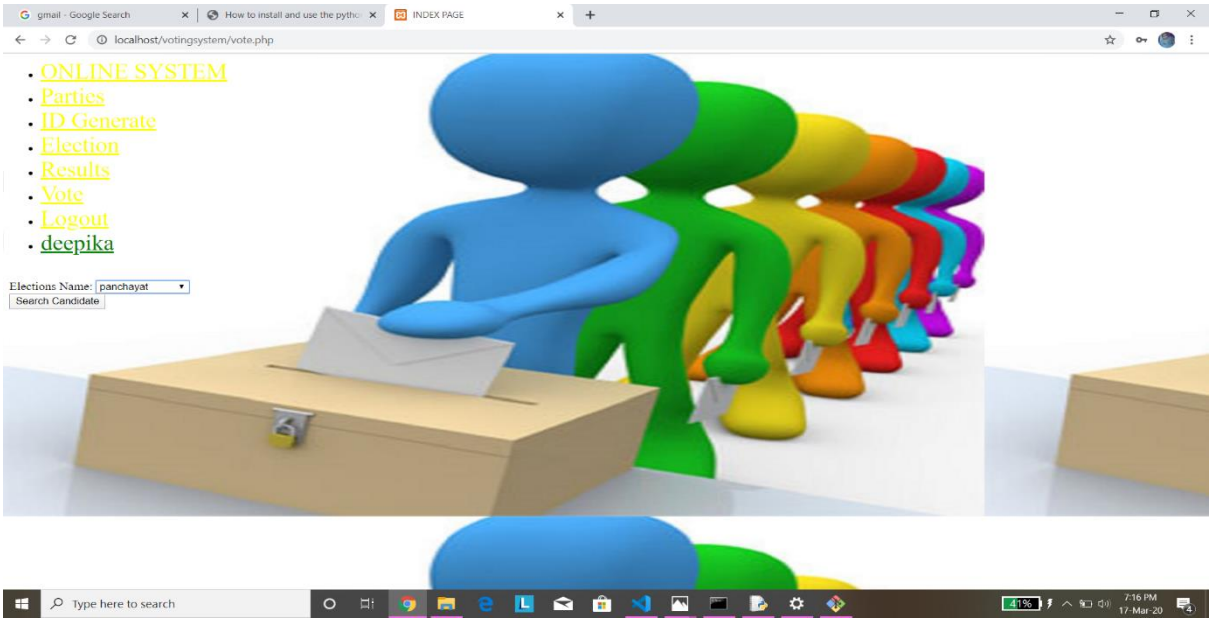


DETAILS OF VOTING FORM DISPLAYED AFTER FACE IS RECOGNIZED

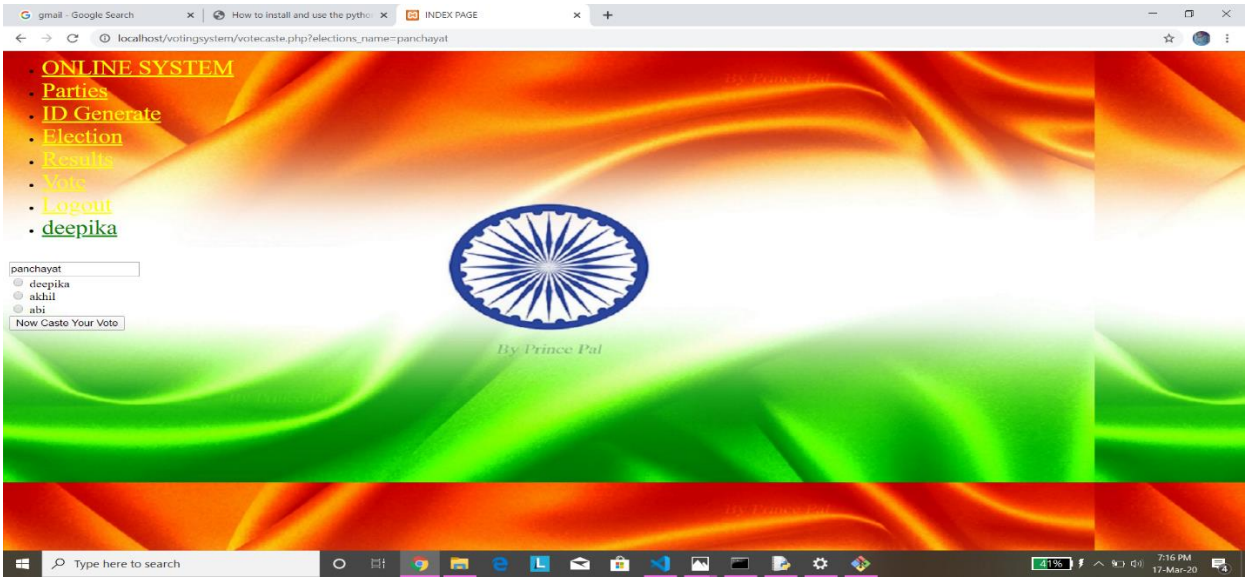
USE GIVEN VOTER ID TO CAST THE VOTE



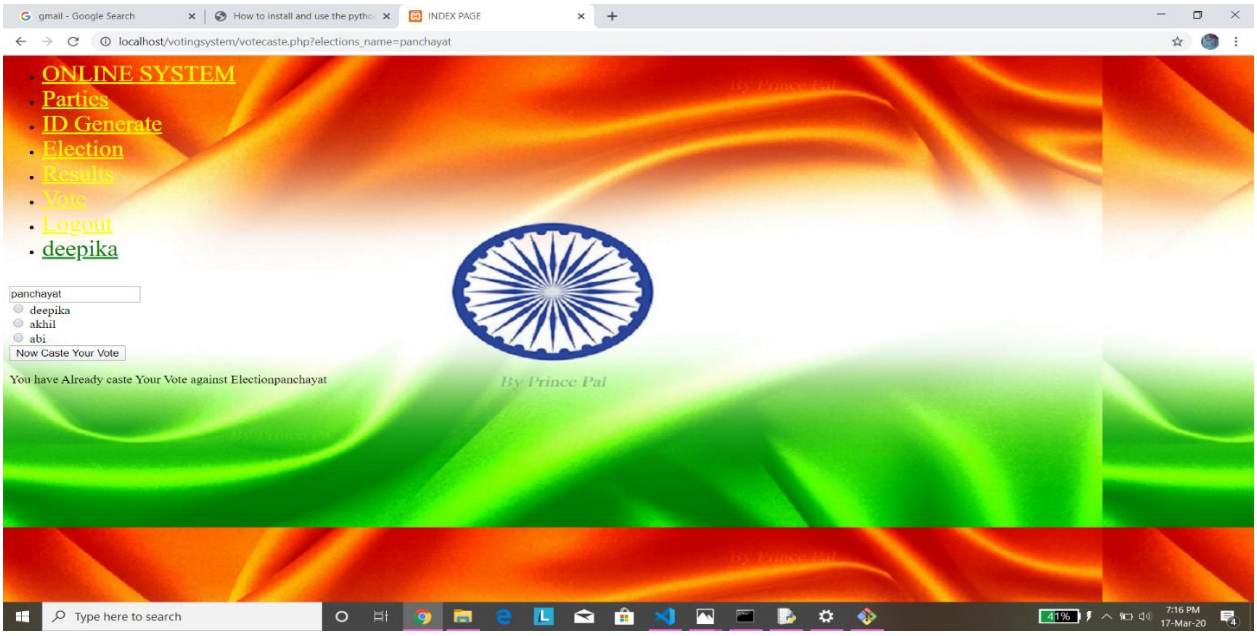
SEARCH FOR THE CANDIDATE YOU WANT TO VOTE



CAST YOUR VOTE HERE

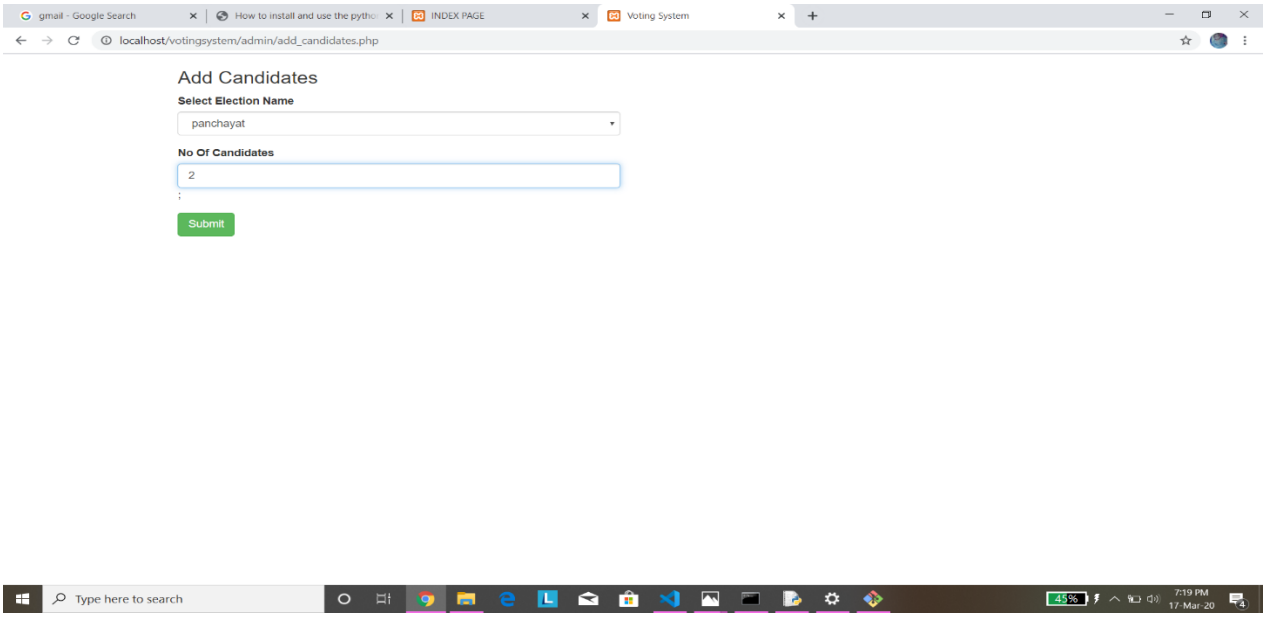


IT CONSIDERS ONLY UNIQUE ID

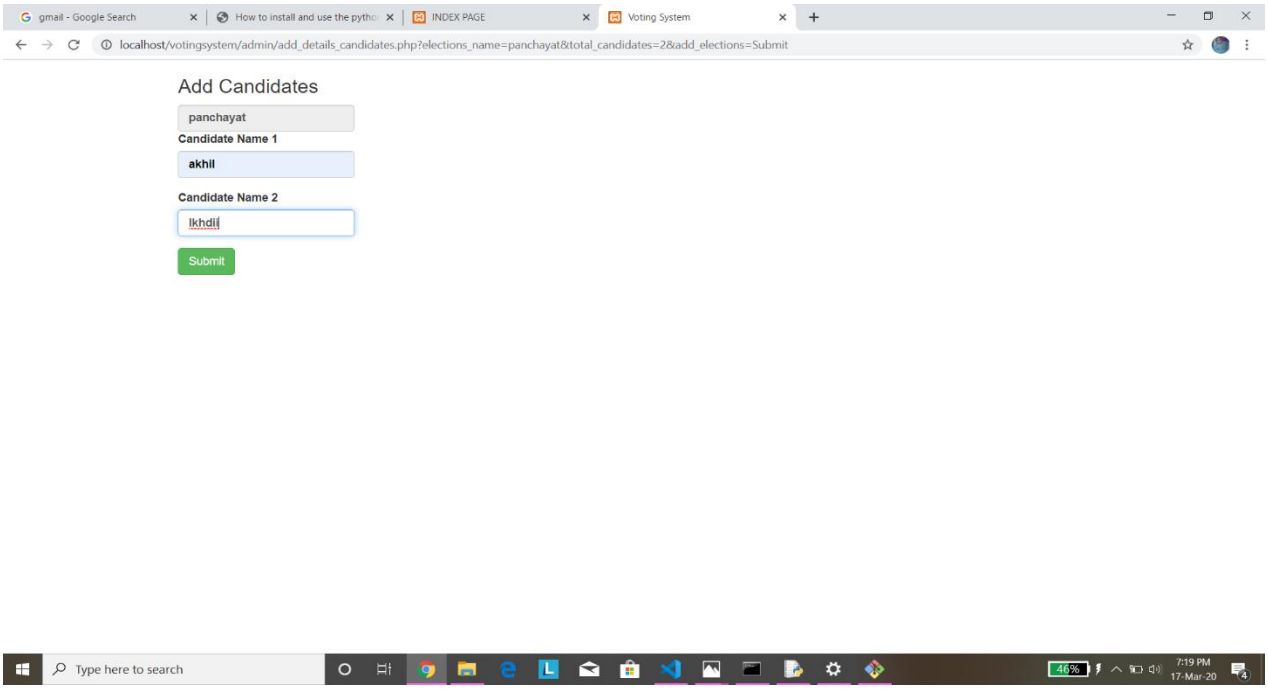


LOGIN USING NEW ID

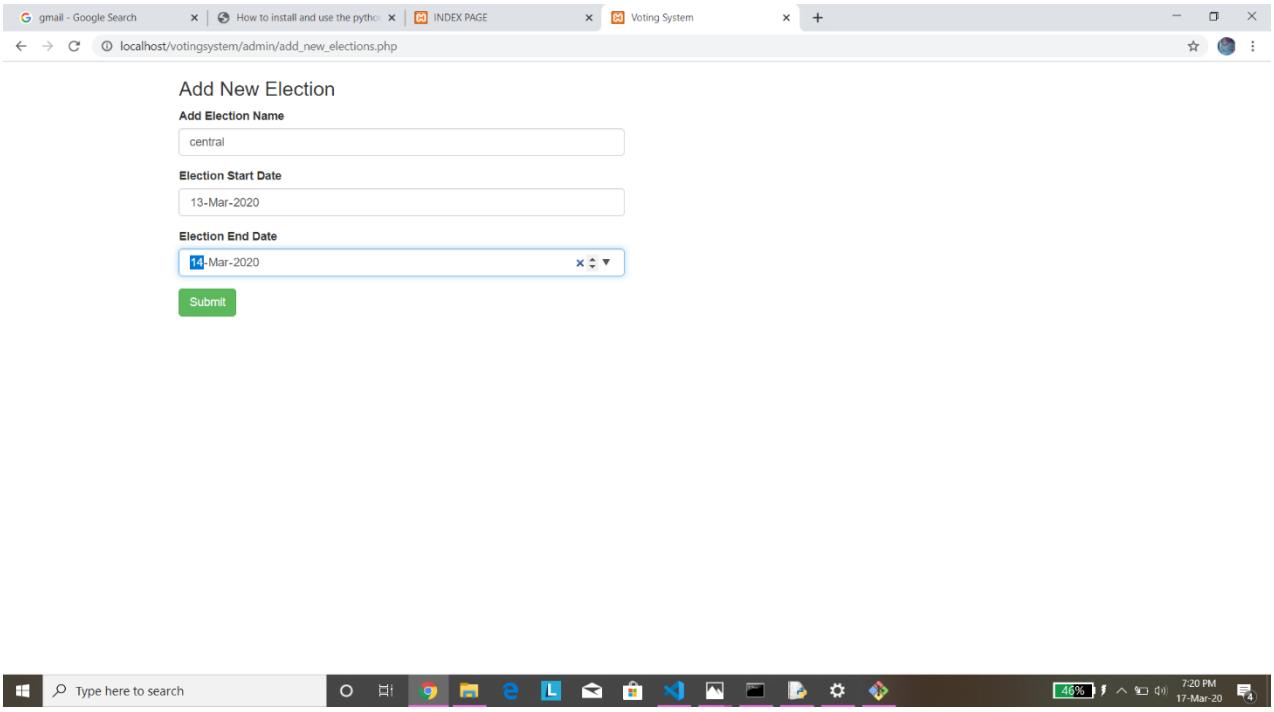
SELECT THE NO OF CANDIDATES



CAST VOTE SEPERATELY FOR EACH CANDIDATE



SUBMIT THE VOTE



CONCLUSION

At present our government is spending more than 125 crores for conducting a Lok Sabha election. This money is spent on issues such as security, electoral ballots etc. The average percentage of voting is a less than 60% .Moreover voting fraud can be easily done in the present system. Also the percentage of literates coming to vote is very less. But with our system the money spent on election can be reduced to less than 10 crores.Also there is no chance of voter frauds and the money spent on security can be drastically decreased. Persons who have an internet connection at home with a web camera can vote without taking the strain to come to voting booths.

REFERENCES

1. Simon Benett, SteeveMc Robb, Ray Farmer, “Object Oriented Analysis and Design using UML”, Tata Mc Hill Publishers, 2008.

2. Terryy Quatrani, “Visual Modelling with Rational Rose 2002”, Prentice Hall Publishers, 1998.

3. GradyBooch,”object –oriented analysis and design with applications ,3rd edition”.addision weseley,2007

4. Rebecawirfs-brocks ,Brian Wilkerson,laurenweiner,”Design Object oriented Software”.prenticeahll

5. Martin flower .”Analysis Patterns :Reusable Object Models “.Addison-wesley.1997.

6. Head First php& My sql ,head first labs. 2013.