Laboratory Manual

Software Architectures
(CS-701)

For

Final Year Students CSE
Dept: Computer Science & Engineering

Submitted To:-
Mr. Ratan Deep Singh
(Assistant Professor)
CSE, ITM

Submitted By:
Riya Jain
0905CS201144
Section : CS-C

LIST OF PROGRAMS

Program-01

ATM APPLICATION SYSTEM

1. ABSTRACT
2. FEATURES
3. ADVANTAGES AND DISADVANTAGES
4. REQUIREMENTS
   - 4.1 FUNCTIONAL REQUIREMENTS
   - 4.2 NON-FUNCTIONAL REQUIREMENTS
   - 4.3 TECHNICAL REQUIREMENTS
5. DIAGRAMS
   - 5.1 USE CASE DIAGRAM
   - 5.2 ACTIVITY DIAGRAM
   - 5.3 SEQUENCE DIAGRAM
   - 5.4 COLLABORATION DIAGRAM
   - 5.5 COMPONENT DIAGRAM
   - 5.6 DEPLOYMENT DIAGRAM
   - 5.7 STATE CHART DIAGRAM

ABSTRACT

An automated teller machine (ATM) or cash machine is an electronic telecommunications device that enables customers of financial institutions to perform financial transactions, such as cash withdrawals, deposits, funds transfers, or account information inquiries, at any time and without the need for direct interaction with bank staff.

Using an ATM, customers can access their bank deposit or credit accounts in order to make a variety of financial transactions, most notably cash withdrawals and balance checking, as well as transferring credit to and from mobile phones. ATMs can also be used to withdraw cash in a foreign country. If the currency being withdrawn from the ATM is different from that in which the bank account is denominated, the money will be converted at the financial institution's exchange rate.

It is an electronic device that is used by only bank customers to process account transactions. The users access their accounts through a special type of plastic card that is encoded with user information on a magnetic strip. The strip contains an identification code that is transmitted to the bank's central computer by modem.

The automated teller machine consists of mainly two input devices and four output devices that are:

Input Devices

- Card reader
- Keypad

Output Devices

- Speaker
- Display Screen
- Receipt Printer
- Cash Depositor

FEATURES

- Transfer funds between linked bank accounts.
- Receive account balance.
- Prints recent transactions list.
- Change your pin.
- Deposit your cash.
- Prepaid mobile recharge.
- Bill payments.
- Cash withdrawal.

Advantages of ATM Application

- The ATM provides 24 hours service
- The ATM provides privacy in banking communications
- The ATMs reduce the workload banks staff
- The ATM may give customer new currency notes
- The ATMs are convenient for banks customers
- The ATM is very beneficial for travelers
- The ATM provides services without any error

Disadvantages of ATM Application

- Fraud. Criminals can fit skimming devices and small cameras to **ATMs**. ...
- Fees. Banks and machine owners draw a huge source of revenue from **ATM** fees. ...
- Theft Risk. If you go to a bank, you're likely walking into a secured area watched by multiple cameras or a life guard. ...
- Card Retention. **ATMs** give, but they can also take.

- Cannot be provided in rural areas: In a country like India, where banks are having large number of rural and non-computerized branches, ATM services cannot be provided.
- Limitation of cash withdrawals: Again, there is a limitation of cash withdrawals from ATM. For example, many banks do not permit withdrawal of more than 25,000 at a time.
- Possibility of misusing ATM card**:** ATM card, if misplaced, lost or stolen, may be misused. There are number of such reported incidences now a day.

REQUIREMENTS:

Functional requirements

1. Secure registration and profile management facilities for Customers
2. Secured mechanism for Payment
3. Account management

Non-functional requirements

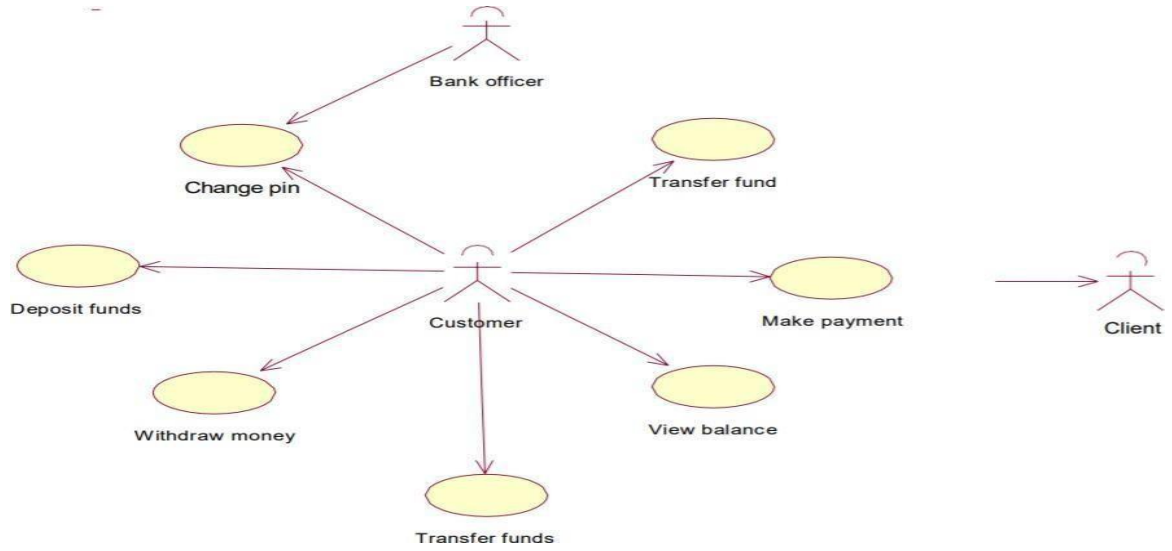
1. Performance
2. Quality
3. Secure access of confidential data (user's details).
4. 24 X 7availability

Technical requirements

1. Browser
2. Server
3. My SQL
4. PHP
5. Java script
6. HTML &CSS

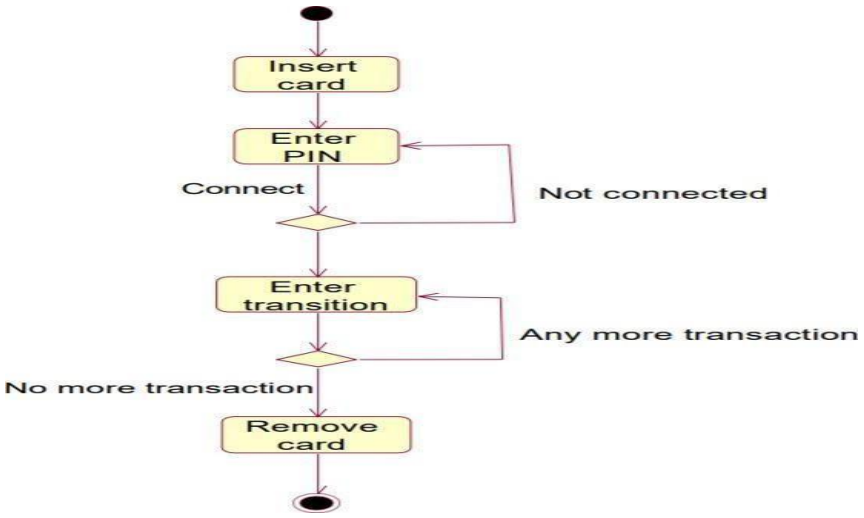
ATM Scenario Use Case Diagram:

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.



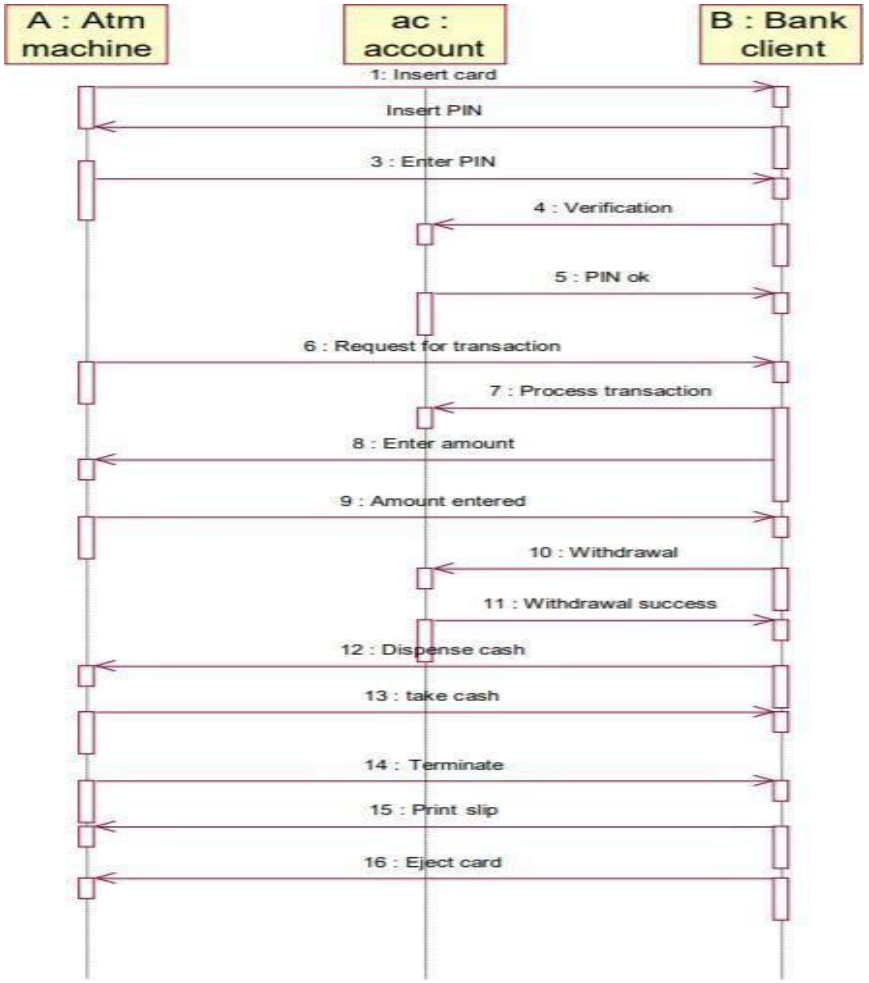
ATM Scenario Activity Diagram:

Activity diagrams are probably the most important UML diagrams for doing business process modeling. In software development, it is generally used to describe the flow of different activities and actions. These can be both sequential and in parallel. They describe the objects used, consumed or produced by an activity and the relationship between the different activities. All the above is essential in business process modeling.



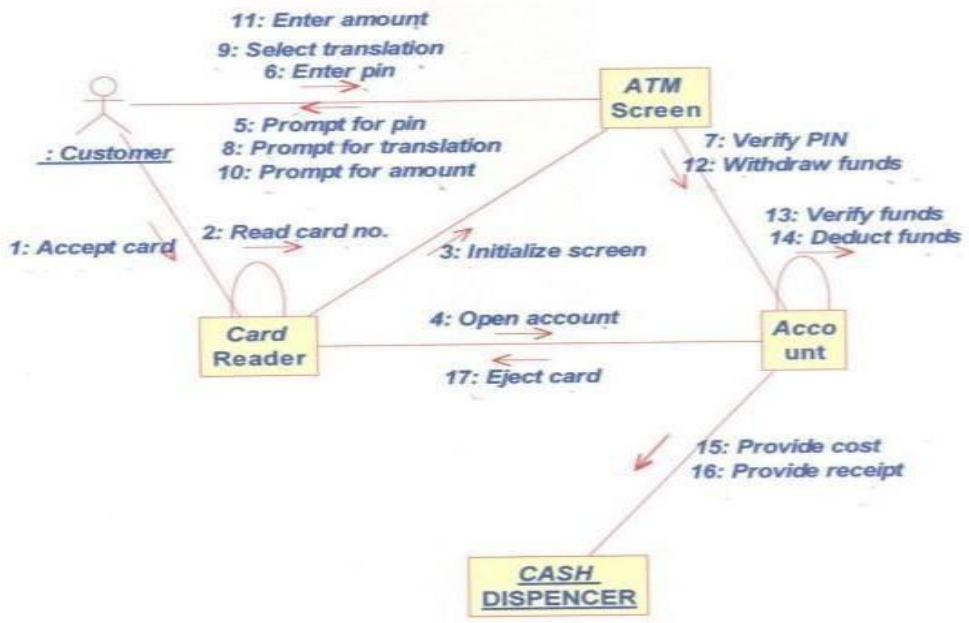
ATM Scenario Sequence Diagram:

A sequence diagram shows a set of messages arranged in time sequence. Each classifier role is shown as a lifeline—that is, a vertical line that represents the role over time through the entire interaction. Messages are shown as arrows between lifelines.



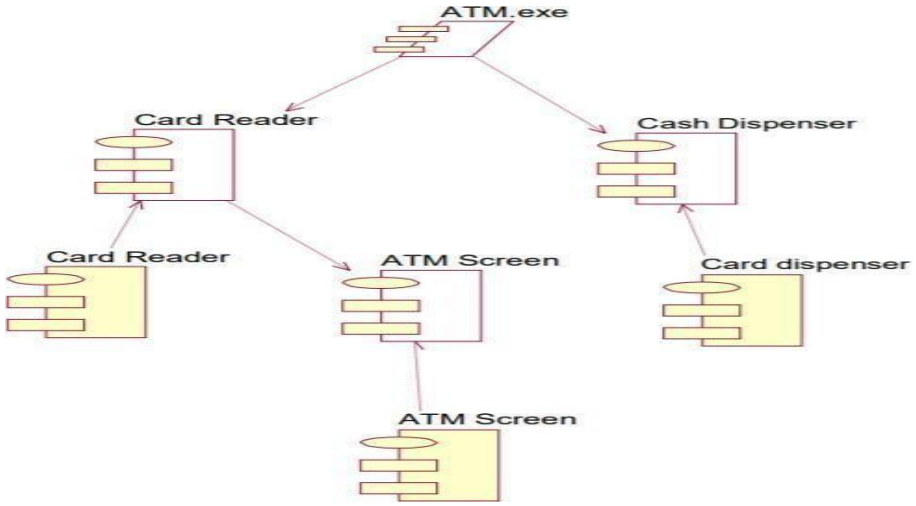
ATM Scenario Collaboration Diagram:

A collaboration is a description of a collection of objects that interact to implement some behavior within a context. It describes a society of cooperating objects assembled to carry out some purpose.



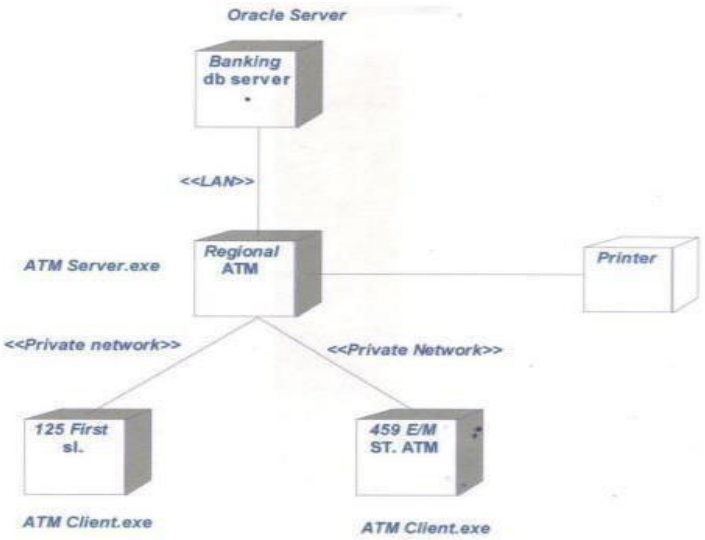
ATM Scenario Component Diagram:

Component UML diagrams can help break down the system into smaller components. Sometimes it is hard to depict the architecture of a system because it might encompass several departments or it might employ different technologies.



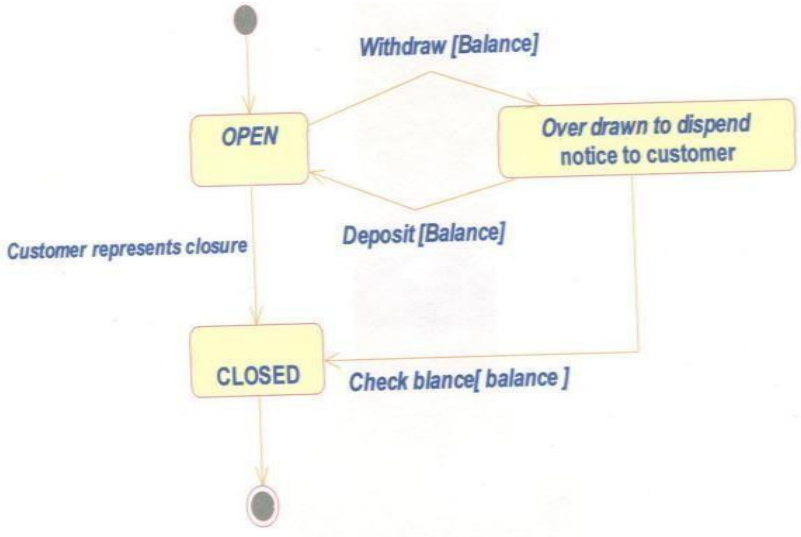
ATM Scenario Deployment Diagram:

Deployment diagrams are a set of nodes and their relationships. These nodes are physical entities where the components are deployed. The deployment view represents the arrangement of runtime component instances on node instances.



ATM Scenario State Chart Diagram:

State chart diagrams, are used to describe the different states of a component within a system. It takes the name state machine because the diagram is essentially a machine that describes the several states of an object and how it changes based on internal and external events.



Program-02

WEATHER MAPPING SYSTEM

1. ABSTRACT
2. FEATURES
3. ADVANTAGES AND DISADVANTAGES
4. REQUIREMENTS
   - 4.1 FUNCTIONAL REQUIREMENTS
   - 4.2 NON-FUNCTIONAL REQUIREMENTS
   - 4.3 TECHNICAL REQUIREMENTS
5. DIAGRAMS
   - 5.1 USE CASE DIAGRAM
   - 5.2 ACTIVITY DIAGRAM
   - 5.3 SEQUENCE DIAGRAM
   - 5.4 SEQUENCE DIAGRAM FOR LOGIN
   - 5.5 SEQUENCE DIAGRAM FOR WEATHER STATION USER
   - 5.6 OBJECT ORIENTED DIAGRAM OF THE SYSTEM

ABSTRACT

A **weather map**, also known as **synoptic weather chart**, displays various meteorological features across a particular area at a particular point in time and has various symbols which all have specific meanings. Such maps have been in use since the mid-19th century and are used for research and weather forecasting purposes.

A weather mapping system (WMS) is required to generate weather maps on a regular basis using data collected from remote, unattended weather stations and other data sources such as weather observers, balloons and satellite.

Weather stations transmit their data to the area computer in response to a request from that machine. The area computer system validates the collected data and integrates the data from different sources. The integrated data is archived and, using data from this archive and a digitized map database, a set of local weather maps is created.

Typically, the weather data collection system establishes a modem link with the weather station and requests transmission of the data.

The data sent to the weather data collection system are the maximum, minimum and average ground temperatures, the maximum, minimum and average air pressures, the maximum, minimum, and average wind speeds.

Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and be modified in the future.

FEATURES

WMS provides a range of map processing operations (e.g., two-dimensional and three-dimensional mappings), weather data storage for the maps, and network access to the maps for remote viewers. Its key marketing features are the following:

- A user-friendly operator environment
- The throughput of weather data acquisition is 50% higher than previous product
- Maps display can be as fast as the maximum hardware speed
- Is designed for easy upgrade to new platforms
- Has open platform connectivity

Advantages

- High Quality of Data
- Reliable weather forecasts
- More accurate results
- Research
- Locate Precipitation
- Weather Surveillance
- Flood Forecasting
- Hail Detection

Disadvantages

- Cannot Detect Fog
- Cannot Detect Wind Independently
- Not Entirely Reliable
- Relies on Intense Datasets
- The Estimates can be wrong
- Weather changes all the time
- Radar Technology keeps growing

REQUIREMENTS:

Functional requirements

1. Secure registration and profile management facilities
2. Secured mechanism
3. Weather Management

Non-functional requirements

1. Performance
2. Quality
3. Secure access of confidential data (user's details).
4. 24 X 7availability

Technical requirements

1. Browser
2. Server
3. My SQL
4. PHP
5. Java script
6. HTML &CSS

Use-Case Diagram of The System

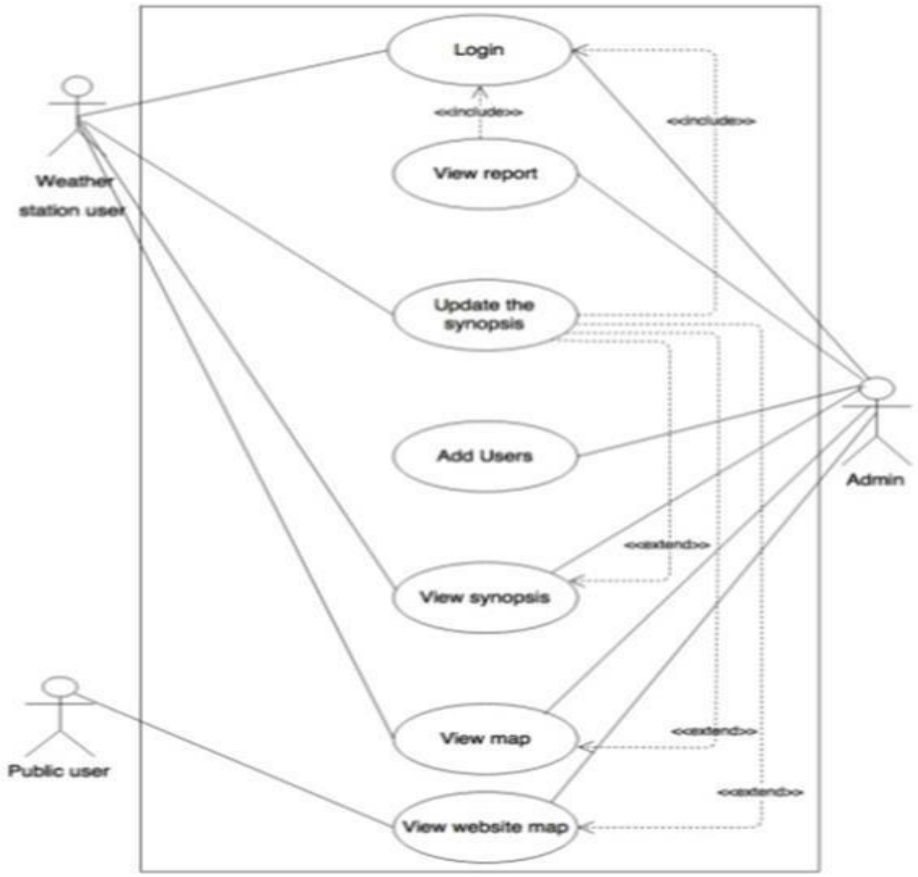
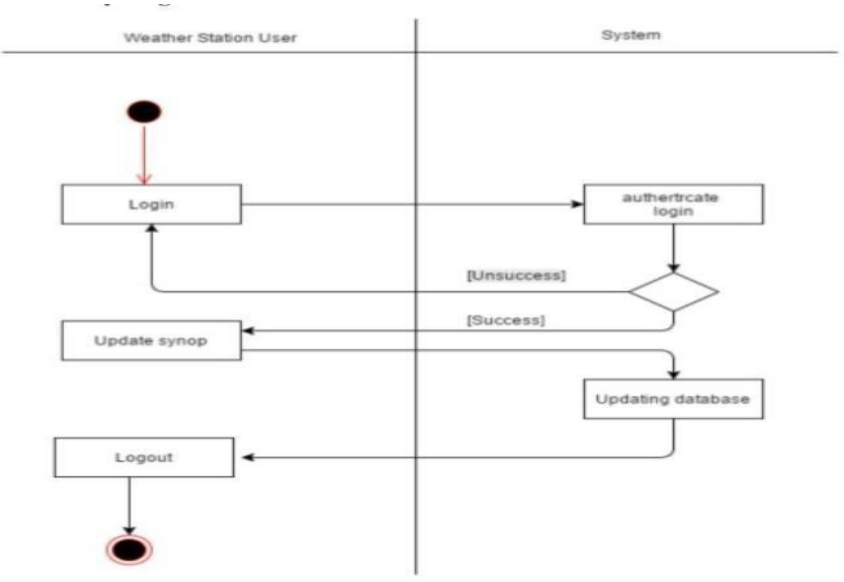
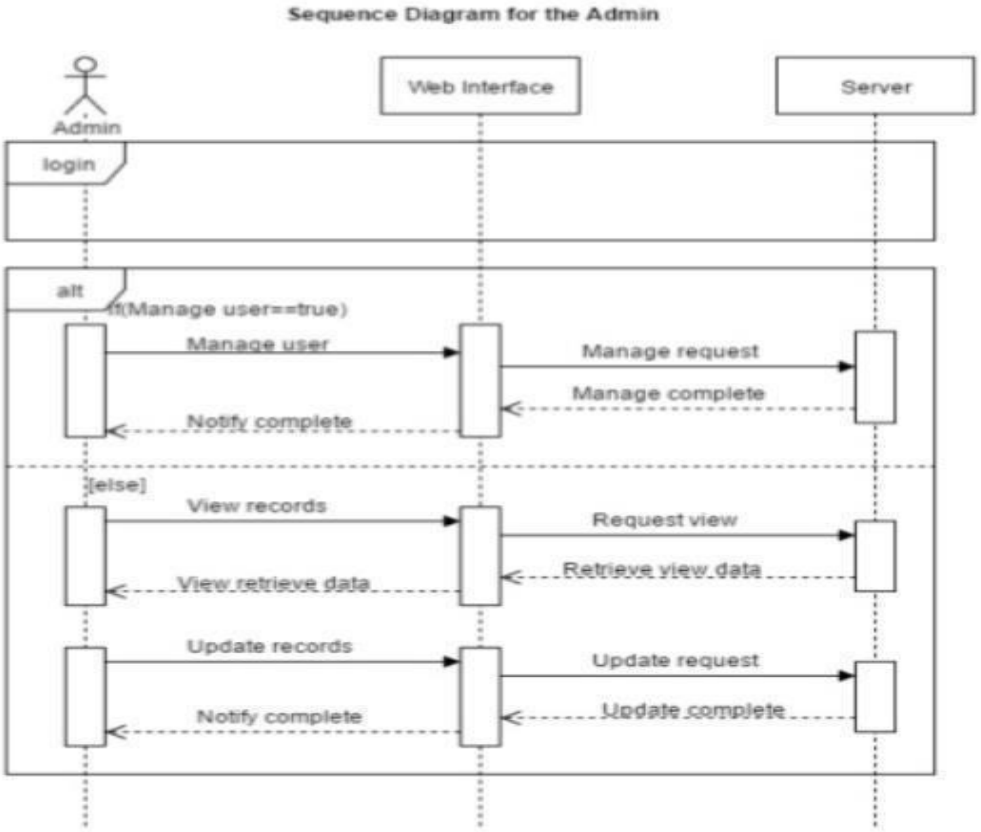


Figure 5.3: Use Case for Weather Stations

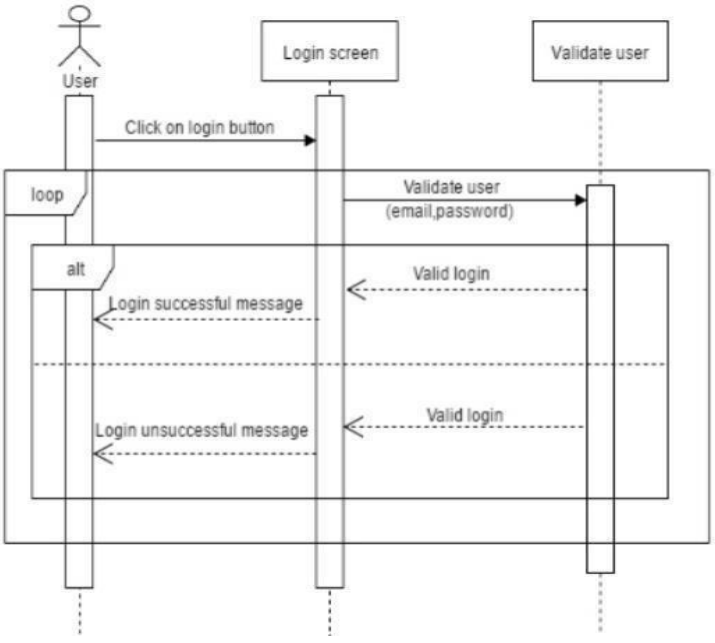
Activity Diagram of the System



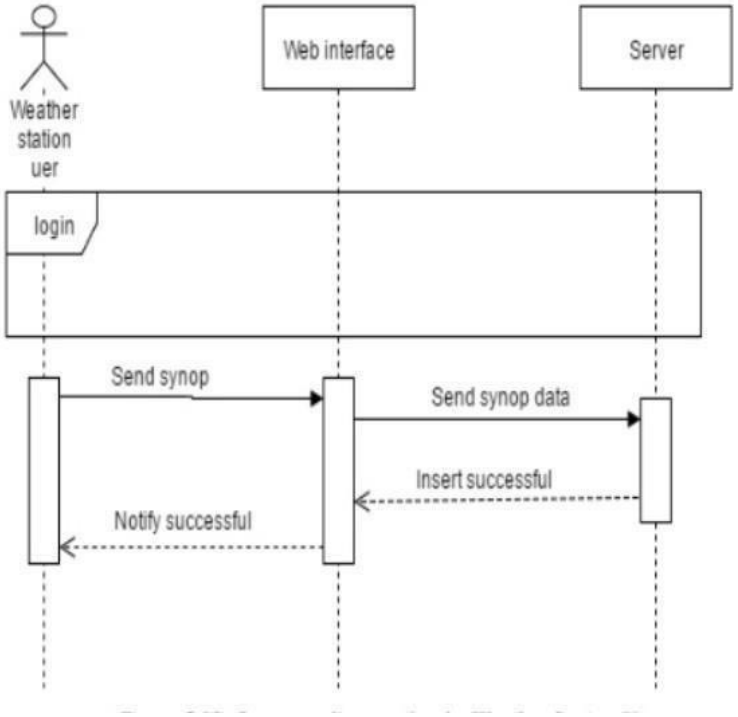
Sequence Diagram of This System



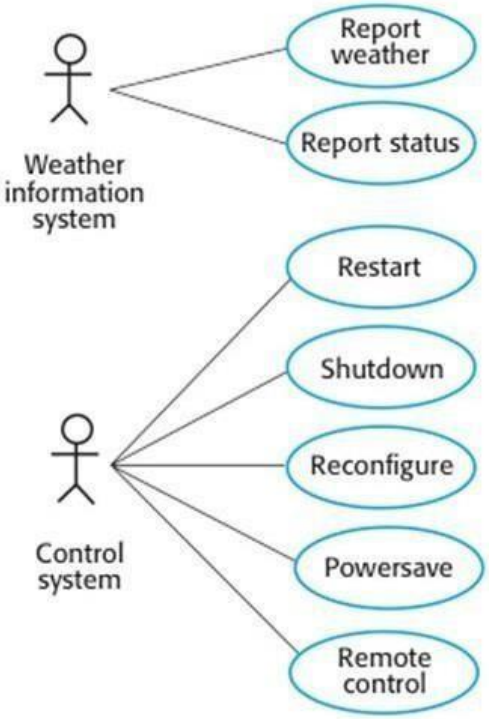
Sequence Diagram for Login



Sequence Diagram for Weather Station User



Object Oriented Design of the System



Program-03

LIBRARY MANAGEMENT SYSTEM

1. ABSTRACT
2. ADVANTAGES AND DISADVANTAGES
3. FEATURES
4. REQUIREMENTS

4.1 FUNCTIONAL REQUIREMENTS

4.3 NON-FUNCTIONAL REQUIREMENTS

4.3 TECHNICAL REQUIREMENTS 5.

DIAGRAMS

5.1 CLASS DIAGRAM

5.2 USECASE DIAGRAM

5.3 ACTIVITY DIAGRAM

5.4 STATE CHART DIAGRAM

5.5 SEQUENCE DIAGRAM

5.6 COLLABORATION DIAGRAM

5.7 DEPLOYMENT DIAGRAM

ABSTRACT

A Library Management System is a software built to handle the primary housekeeping functions of a library. Libraries rely on library management systems to manage asset collections as well as relationships with their members. Library management systems help libraries keep track of the books and their checkouts, as well as members' subscriptions and profiles.

Library management systems also involve maintaining the database for entering new books and recording books that have been borrowed with their respective due dates. It provides familiar and attractive interface with inserting and reporting capabilities.

Library management system is a project which aims in developing a computerized system to maintain all the daily work of library

. It is a term for computer-based **system** that manage the catalogue of a **library**. The main **purpose** of this **system** is to manage **library** daily operation efficiently

Advantages

1. Simple & Easy to Use

The Library Management Software is simple, user-friendly, and can be easily integrated with your existing system. The library management system benefits provide online and offline storage, automated backups, and easy upgrades to simplify and enhance the learning process.

2. Increased Library Engagement

Avoid frustration and tediousness by providing students with 24/7 access to library resources from anywhere, anytime. Library Management Software allows the librarian to maintain all types of books, eBooks, journals, photos, videos, and create events.

3.Efficient Cloud Data Management

Automate, simplify and deploy library database seamlessly to make it easy for your institution to benefit from secure cloud services. Improve efficiency with the automation of various library tasks including acquisition, cataloging, serials management, circulation and reference.

4. Highly Secure, Scalable & Reliable

College libraries benefit from scalable infrastructure, role-based secure access, high performance and reliable to ensure seamless access to library database.

5. Mobile Access

The library management system provides mobile access to search the library catalog, schedules, books and resources from anywhere, at any given time via smartphones and tablets.

7. Error-free

The automated library software is user-friendly, powerful and developed for easy entry of data, makes library operations free from errors.

8. Innovation

Students can search, write articles, upload photos and videos, manage email, send messages, but also help them to keep up with the librarian and other students via chat, discussion forums, and social media.

9. Cost-effective

Embracing sophisticated technologies is cost-effective and a viable choice for education institutions. Using cloud, mobile and digital libraries eliminates paper-based processes and maintenance overheads, improves productivity, reduces operation costs and saves time.

Disadvantages

1. The data stored is prone to cyber hacks. Opting for a reliable online system eliminates the risk
2. Costly and Expensive
3. Complicated to operate
4. Online Systems require high-speed internet connectivity
5. Risk of computer virus
6. The automation feature is not available in offline/ open-source systems thus, requires manual action to perform operations

Features

- A modern integrated **library management system** (LMS).
- Can be scalable to Windows, Linux and Mac OS platform.
- Print your own barcodes.
- Full catalog, circulation and acquisitions **system** for **library** stock **management**.
- Web based OPAC (Online Public Access Catalog) **system**.
- Simple, clear search interface for all users.

REQUIREMENTS:

Functional Requirements

1. A new user which is not registered in the system. Registration form must be available. Details must be correct.
2. Member must be registered against unique ID
3. Books must be available
4. We must have librarian account which manages the whole system.

Non-functional requirements

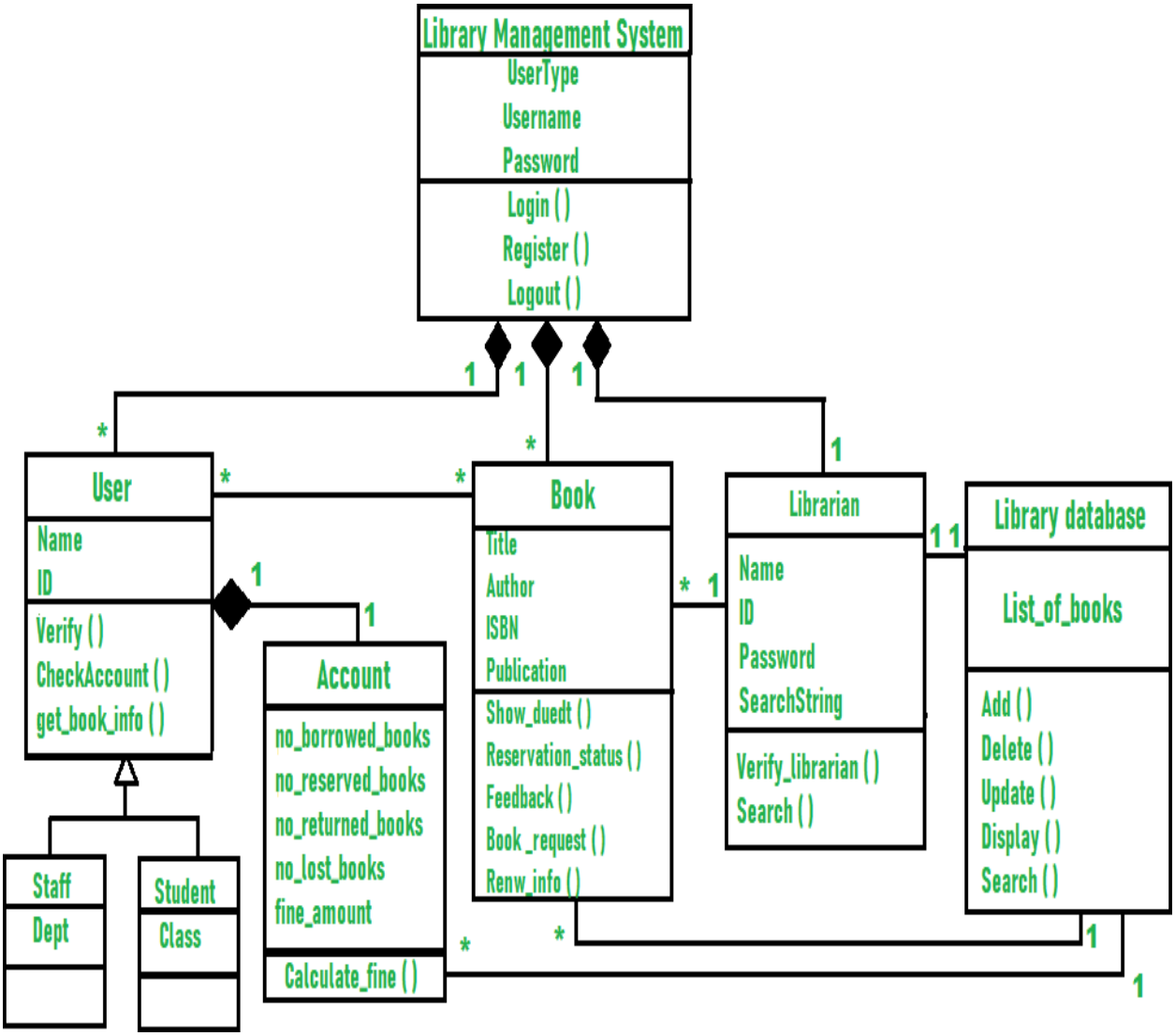
1. Performance
2. Quality
3. Secure access of confidential data (user's details).
4. 24 X 7availability

Technical requirements

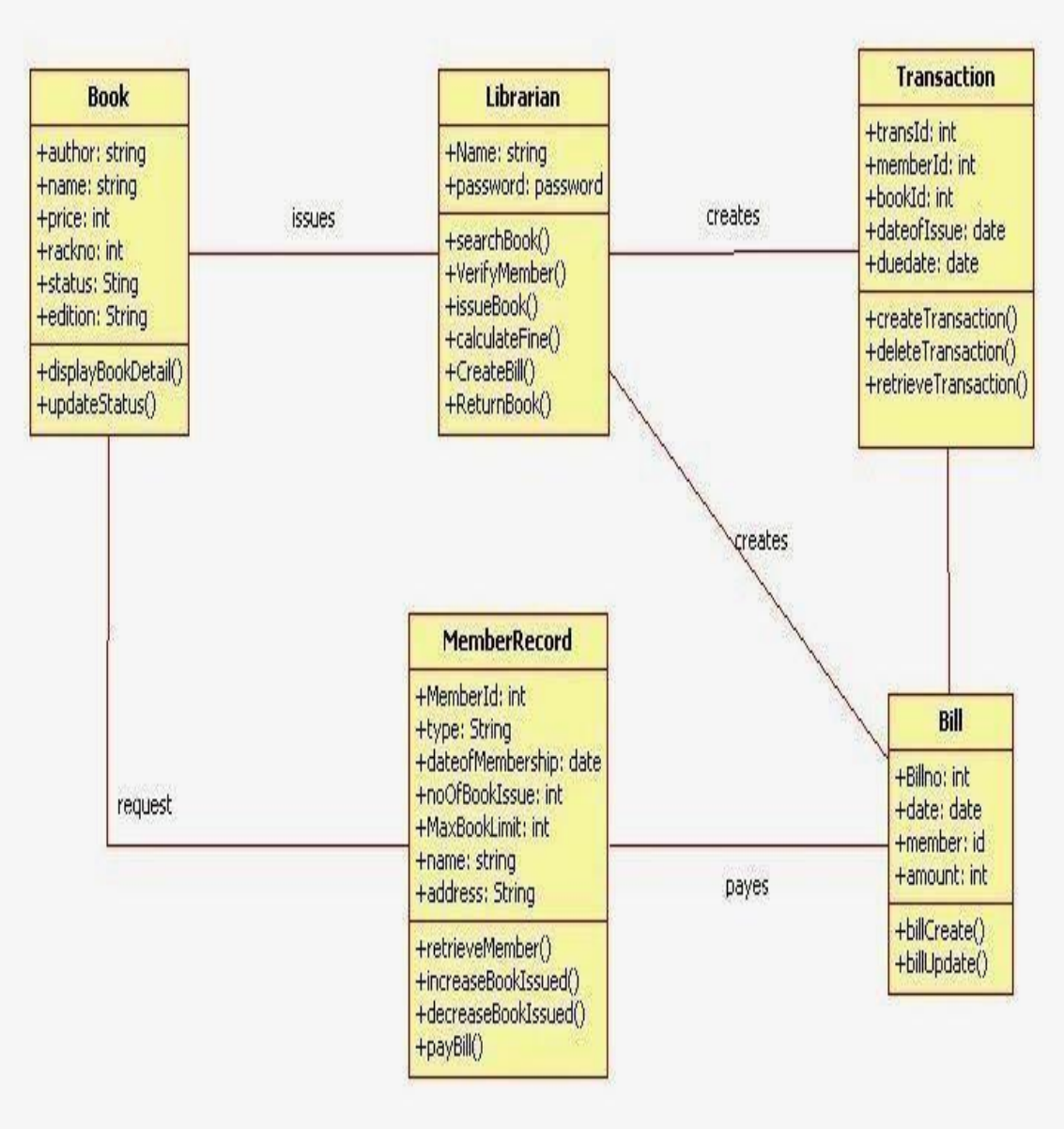
1. Browser
2. Server
3. My SQL
4. PHP
5. Java script
6. HTML &CSS

Diagram

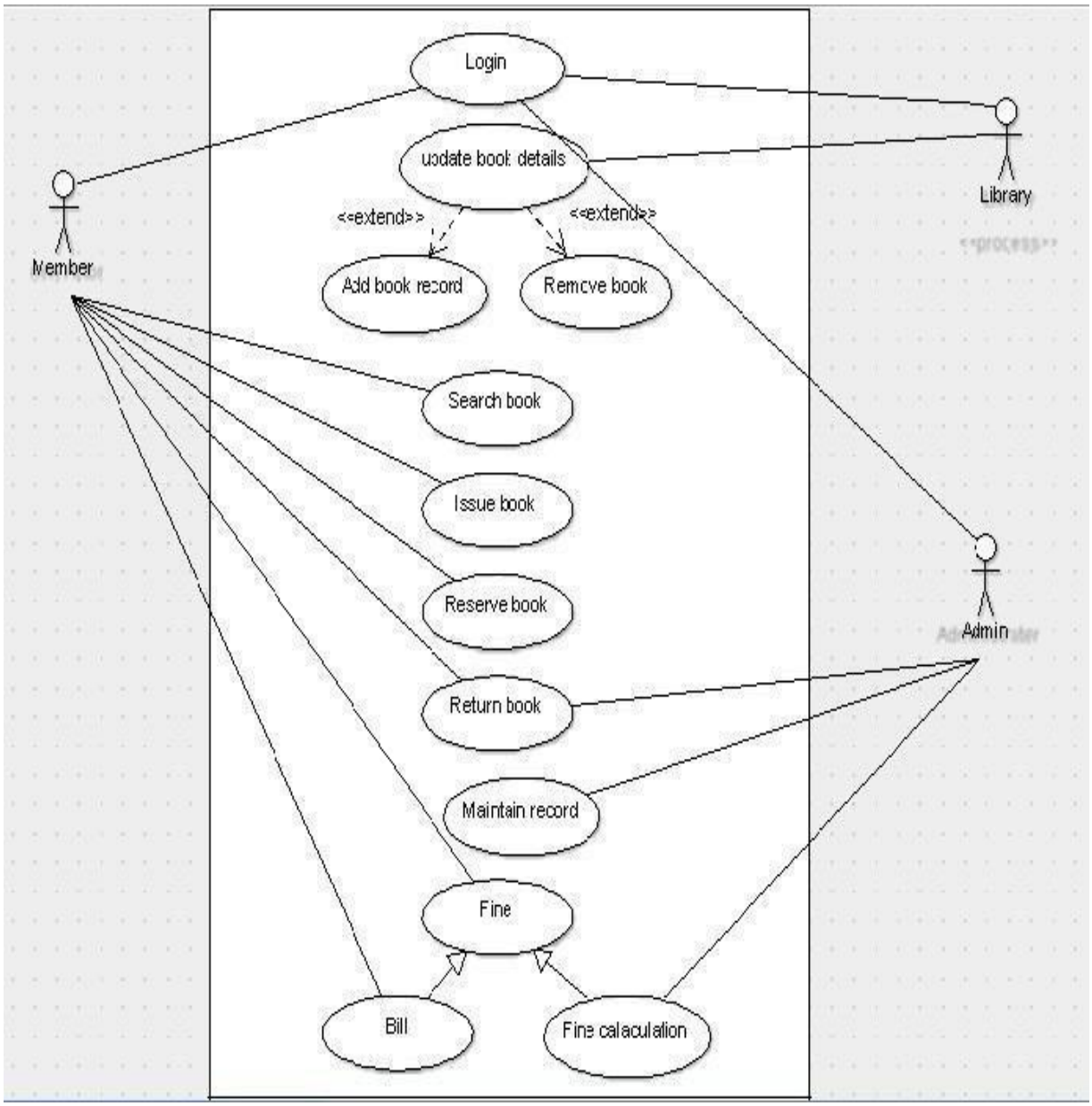
Class Diagram



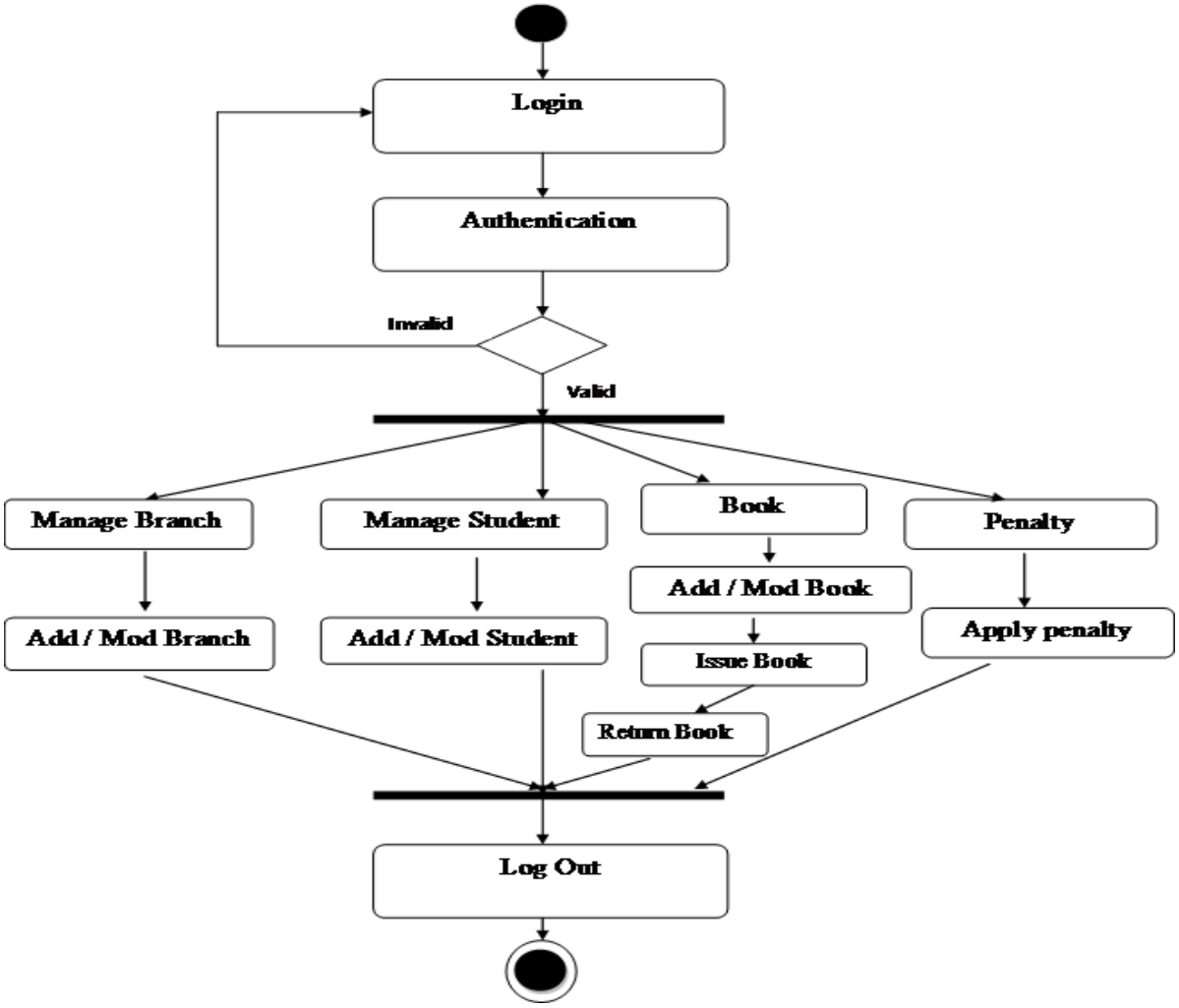
Object Diagram



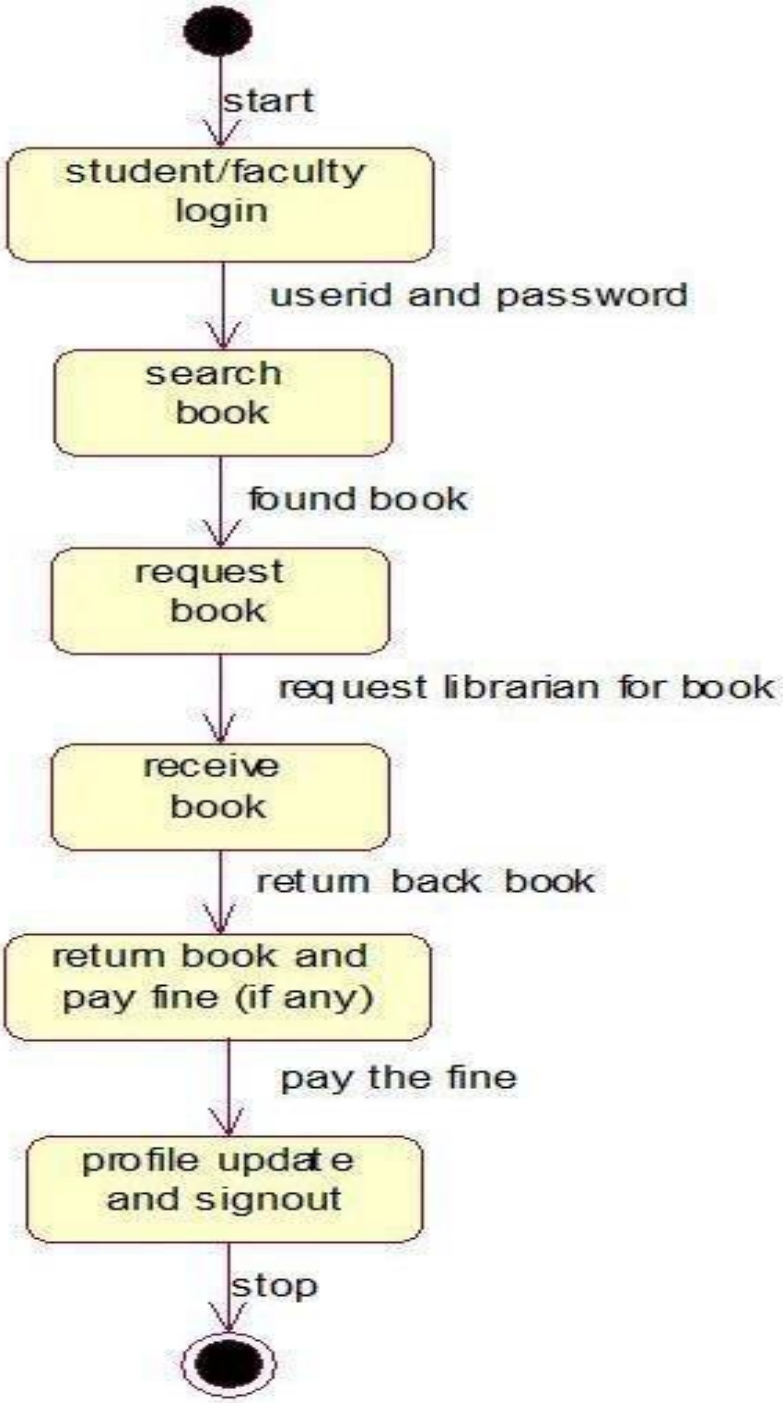
USECASE DIAGRAM



ACTIVITY DIAGRAM

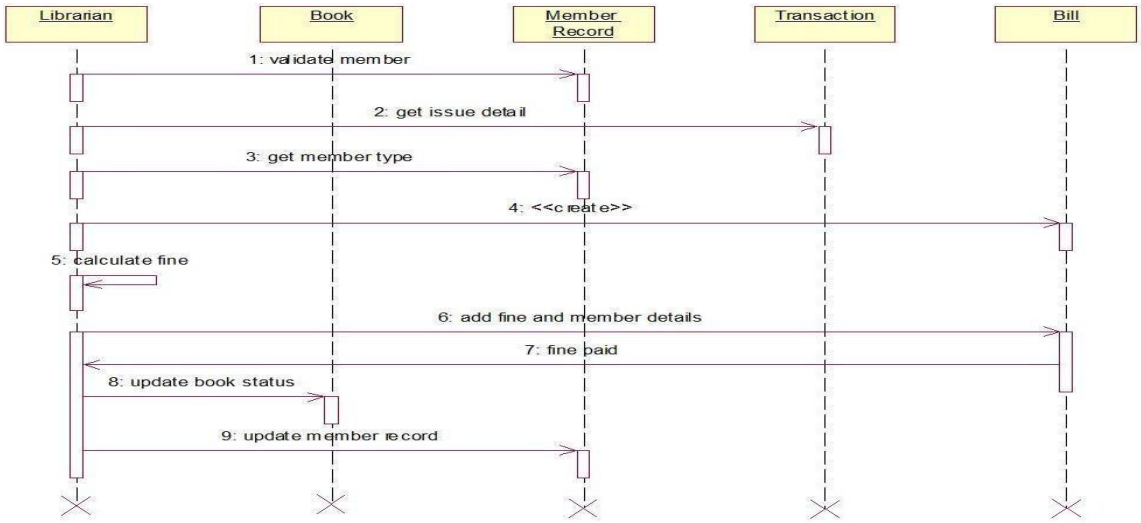


State Chart Diagram



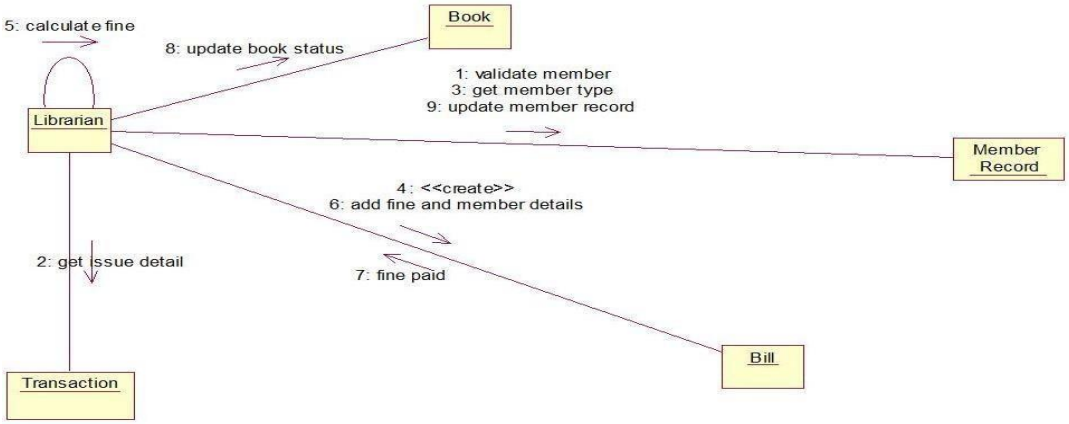
Sequence Diagram

A sequence diagram shows a set of messages arranged in time sequence. Each classifier role is shown as a lifeline—that is, a vertical line that represents the role over time through the entire interaction. Messages are shown as arrows between lifelines.



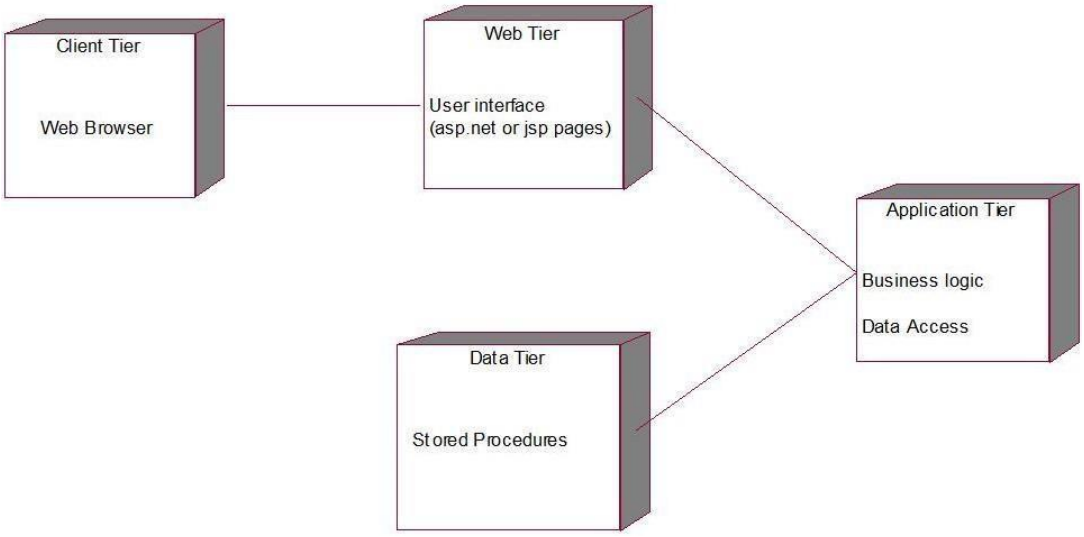
COLLABORATION DIAGRAM

A collaboration is a description of a collection of objects that interact to implement some behavior within a context. It describes a society of cooperating objects assembled to carry out some purpose. A collaboration contains slots that are filled by objects and links at run time. A collaboration slot is called a role because it describes the purpose of an object or link within the collaboration.



DEPLOYMENT DIAGRAM

Deployment diagrams are a set of nodes and their relationships. These nodes are physical entities where the components are deployed. The deployment view represents the arrangement of runtime component instances on node instances.



Program -04

Online Book Shop

1. ABSTRACT
2. REQURIMENT :
    - 2.1 FUNCTIONAL REQUIREMENTS
    - 2.2 NON-FUNCTIONAL REQUIREMENTS
    - 2.3 TECHNICAL REQUIREMENTS
3. DIAGRAMS
    - 3.1 CLASS DIAGRAM
    - 3.2 USECASE DIAGRAM
    - 3.3 ACTIVITY DIAGRAM
    - 3.4 STATE CHART DIAGRAM
    - 3.5 SEQUENCE DIAGRAM
    - 3.6 COMPOMENT DIAGRAM
    - 3.7 DEPLOYMENT DIAGRAM

Abstract

The main objective of the project is to create an online book store that allows users to search and purchase a book online based on title, author and subject. The selected books are displayed in a tabular format and the user can order their books online through credit card payment. Using this Website, the user can purchase a book online instead of going out to a book store and wasting time.

Online Book store is an online web application where the customer can purchase books online. Through a web browser the customers can search for a book by its title or author, later can add to the shopping cart and finally purchase using credit card transaction. The user can login using his account details or new customers can set up an account very quickly.

The Online Book Store Website provides customers with online shopping through a web browser. A customer can, create, sign in to his account, place items into a shopping cart and purchase using his credit card details.

The Administrator will have additional functionalities when compared to the common user. He can add, delete and update the book details, book categories, member information and also confirm a placed order.

The main objective of the project is to create an online book store that allows users to search and purchase a book based on title, author and subject. The selected books are displayed in a tabular format and the user can order their books online through credit card payment. The Administrator will have additional functionalities when compared to the common user.

The motivation to create this project has many sources

Interest to develop a good user-friendly website with many online transactions using a database.

To increase my knowledge horizon in technologies like .NET, SQL, CSS, HTML.

To gain good experience in .NET before joining in a full-time job.

To gain expertise using Data Grid, Data Set, Data Table, Data Adapter and Data Readers.

REQUIREMENTS:

Functional Requirements

1. A new user which is not registered in the system. Registration form must be available. Details must be correct.
2. Member must be registered against unique ID
3. Books must be available
4. We must have librarian account which manages the whole system.

Non-functional Requirements

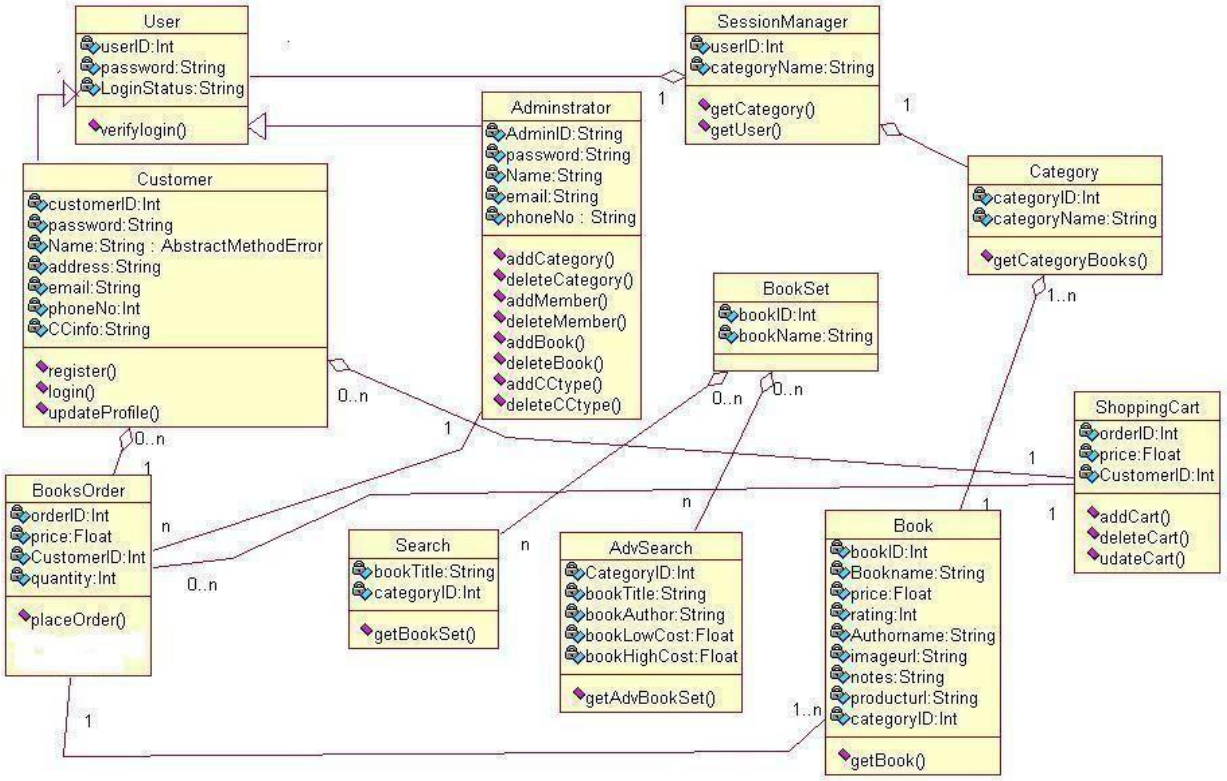
1. Performance
2. Quality
3. Secure access of confidential data (user's details).
4. 24 X 7availability

Technical Requirements

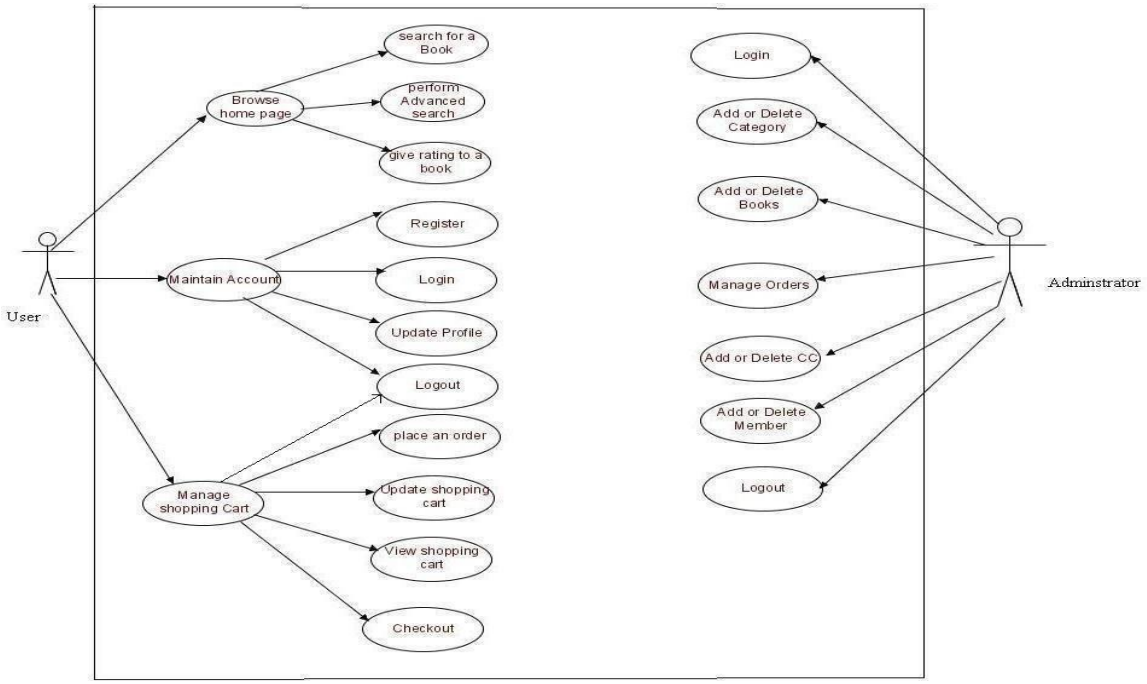
1. Browser
2. Server
3. My SQL
4. PHP
5. Java script
6. HTML &CSS

DIAGRAMS:

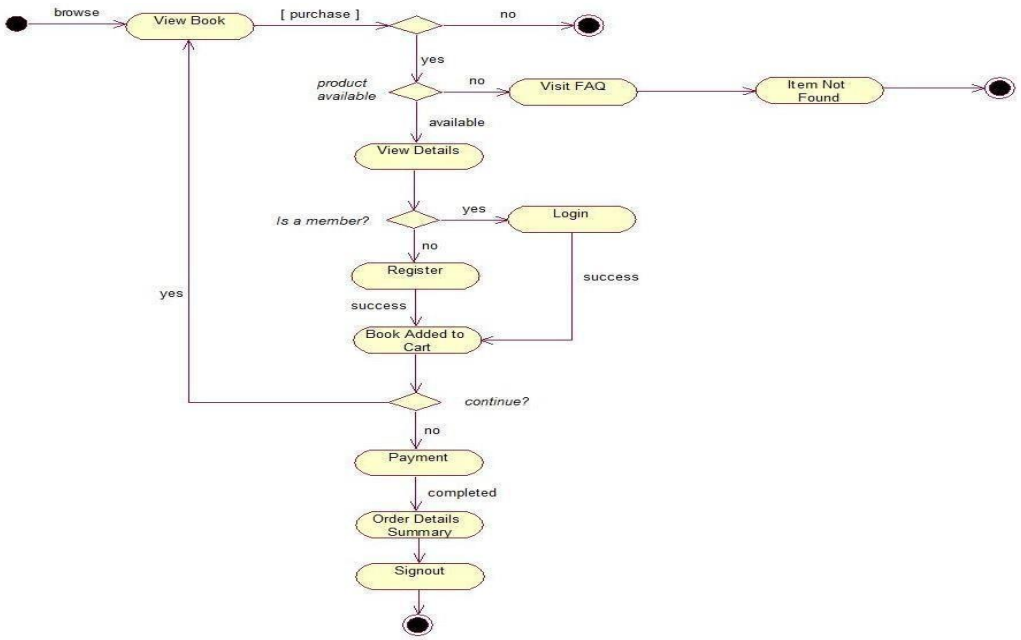
CLASS DIAGRAM



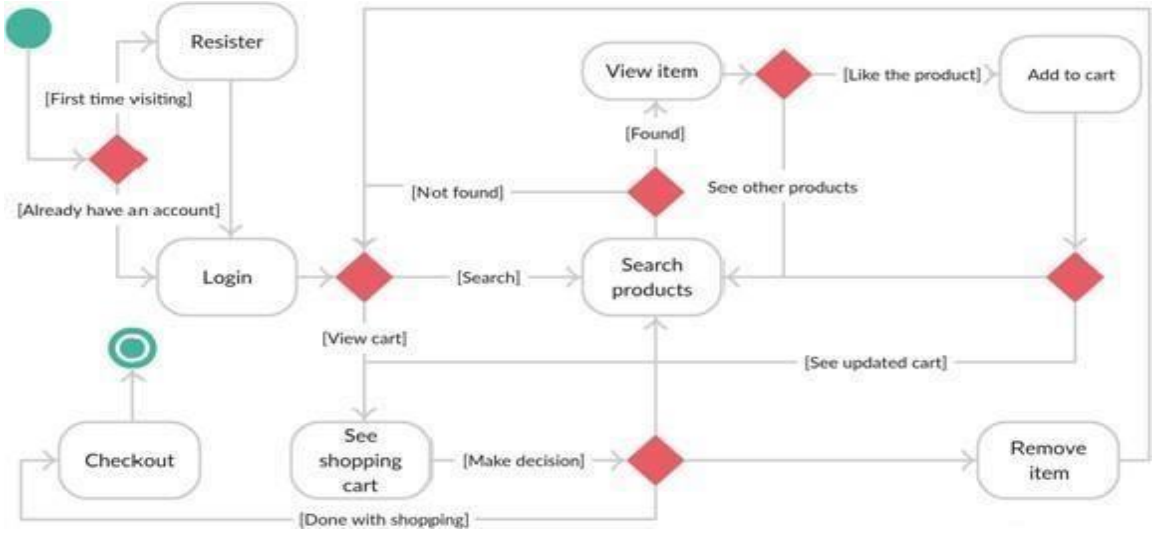
Use Case Diagram



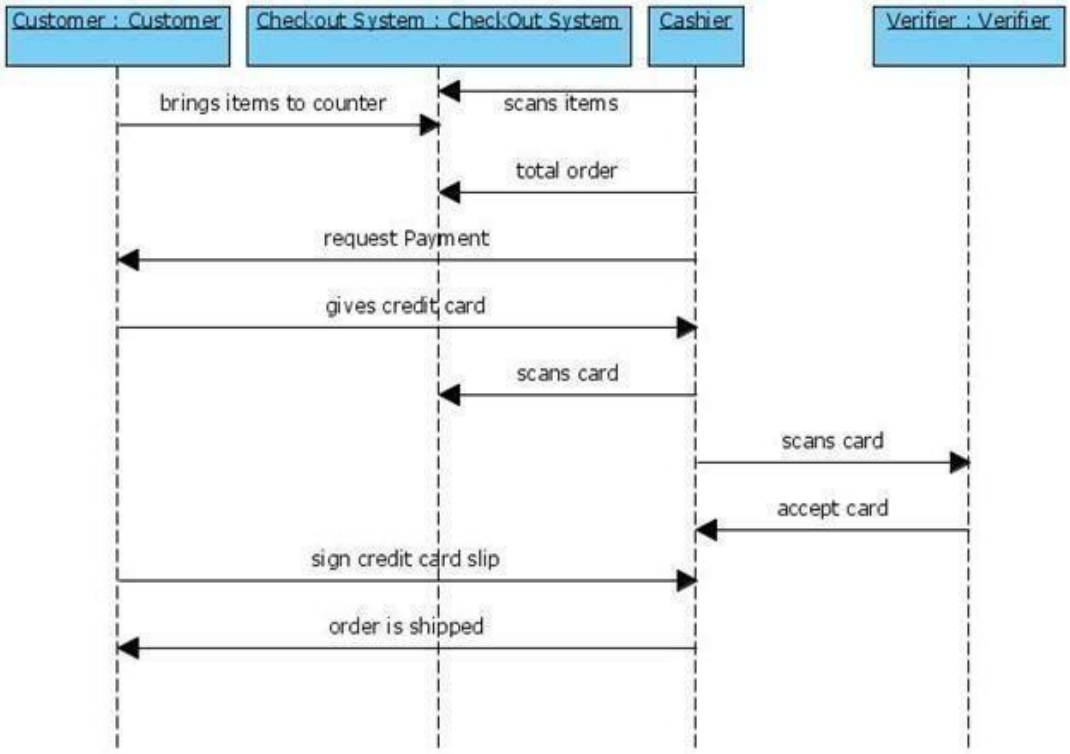
ACTIVITY DIAGRAM



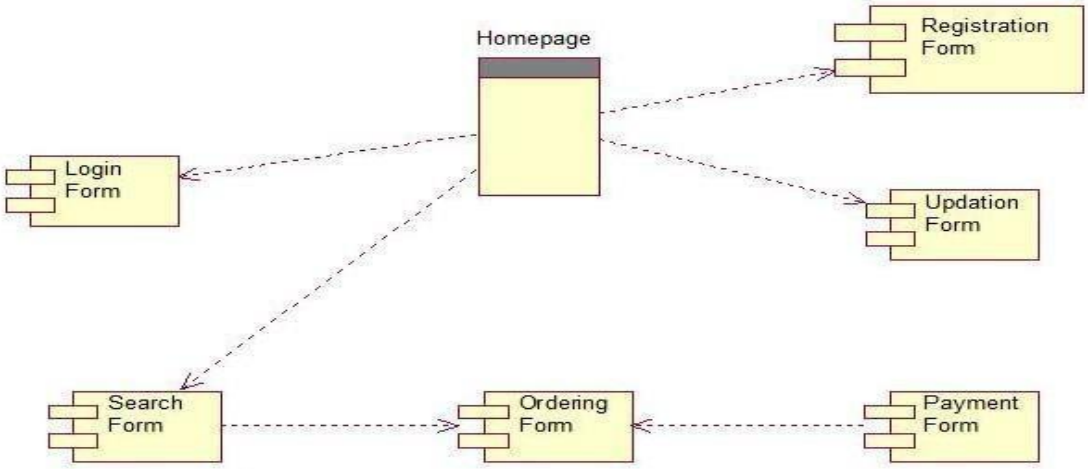
STATE CHART DIAGRAM



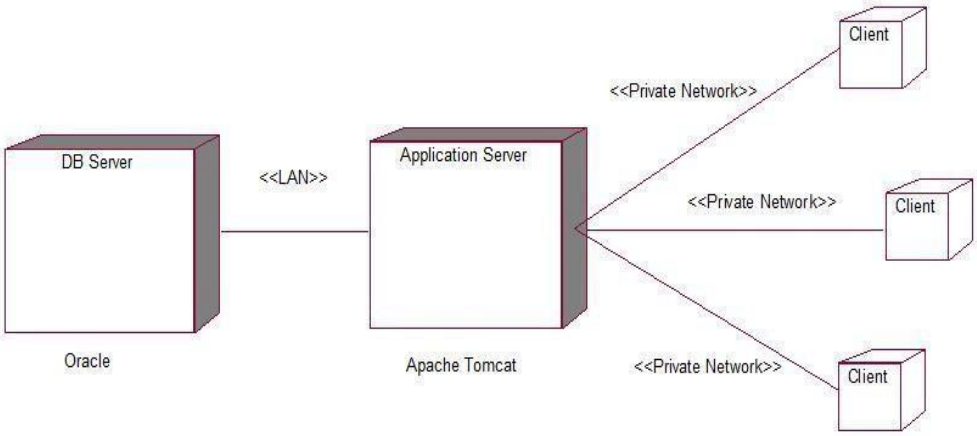
Sequence Diagram



Component Diagram



Deployment Diagram



Program-05

RAILWAY RESERVATION SYSTEM

1 ABSTRACT

2 REQUIREMENTS:

1. FUNCTIONAL REQUIREMENTS
2. NON-FUNCTIONAL REQUIREMENTS
3. TECHNICAL REQUIREMENTS

3 DIAGRAMS:

1. CLASS DIAGRAM
2. OBJECT DIGRAM
3. USE CASE DIAGRAM
4. ACTIVITY DIAGRAM
5. STATECHART DIAGRAM
6. SEQUENCE DIAGRAM
7. COLLABORATION DIAGRAM
8. DEPLOYMENT DIAGRAM

ABSTRACT

The railway reservation system is software for the purpose of reserving train seats at any time and from anywhere. This application provides us complete information about a train between specified source and destination.

This application gives current status of reservations of particular train, fares for different classes of train and also waiting status. For this application, a visitor must register to avail the service. In this system, train records are maintained and retrieved. Administrator monitors all users and their transactions.

Administrator has complete access to database and can add train or cancel train or add station for particular train or skip a station for train. There are several payment options for user like credit card, debit card and a user can also cancel ticket. This application also provides current position of train i.e., name of stations between which the train is currently running.

It provides familiar and attractive interface with inserting and reporting capabilities.

REQUIREMENTS:

Functional requirements

1. Secure registration and profile management facilities for Customers
2. Secured mechanism for Payment
3. 3 Account management

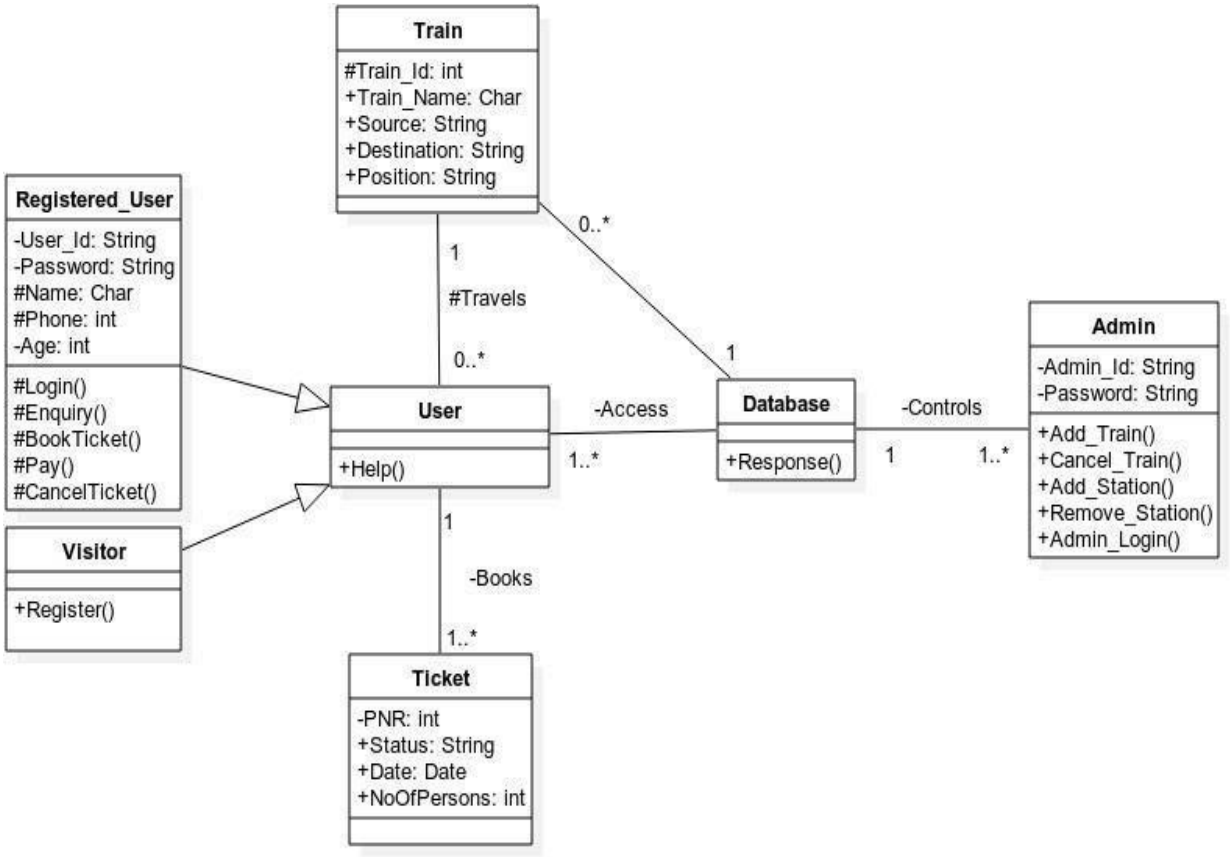
Non-functional Requirements

1. Performance
2. Quality
3. Secure access of confidential data (user's details).
4. 24 X 7availability

Technical Requirements

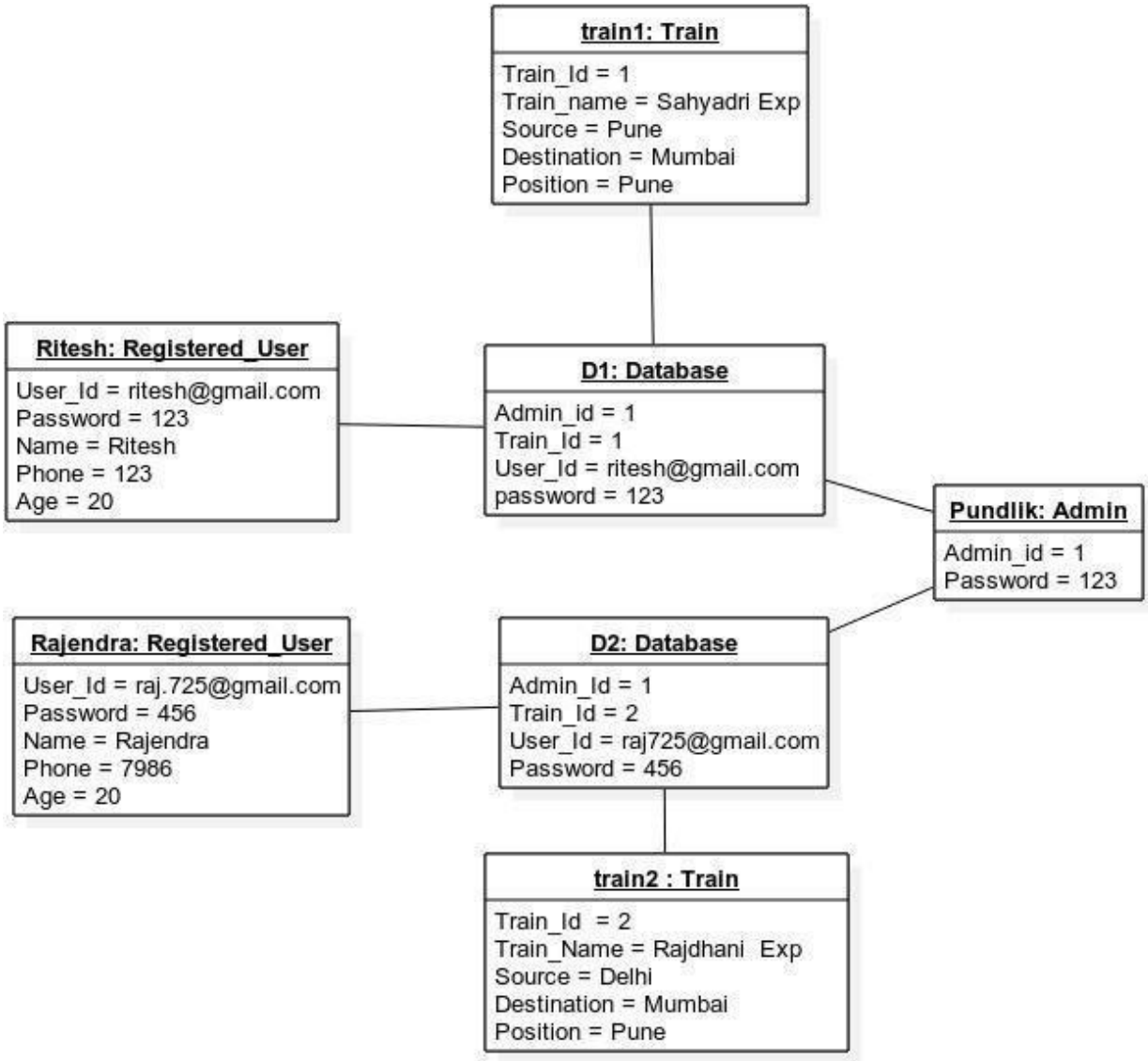
1. Browser
2. Server
3. My SQL
4. PHP
5. Java script
6. HTML &CSS

CLASS DIAGRAM:



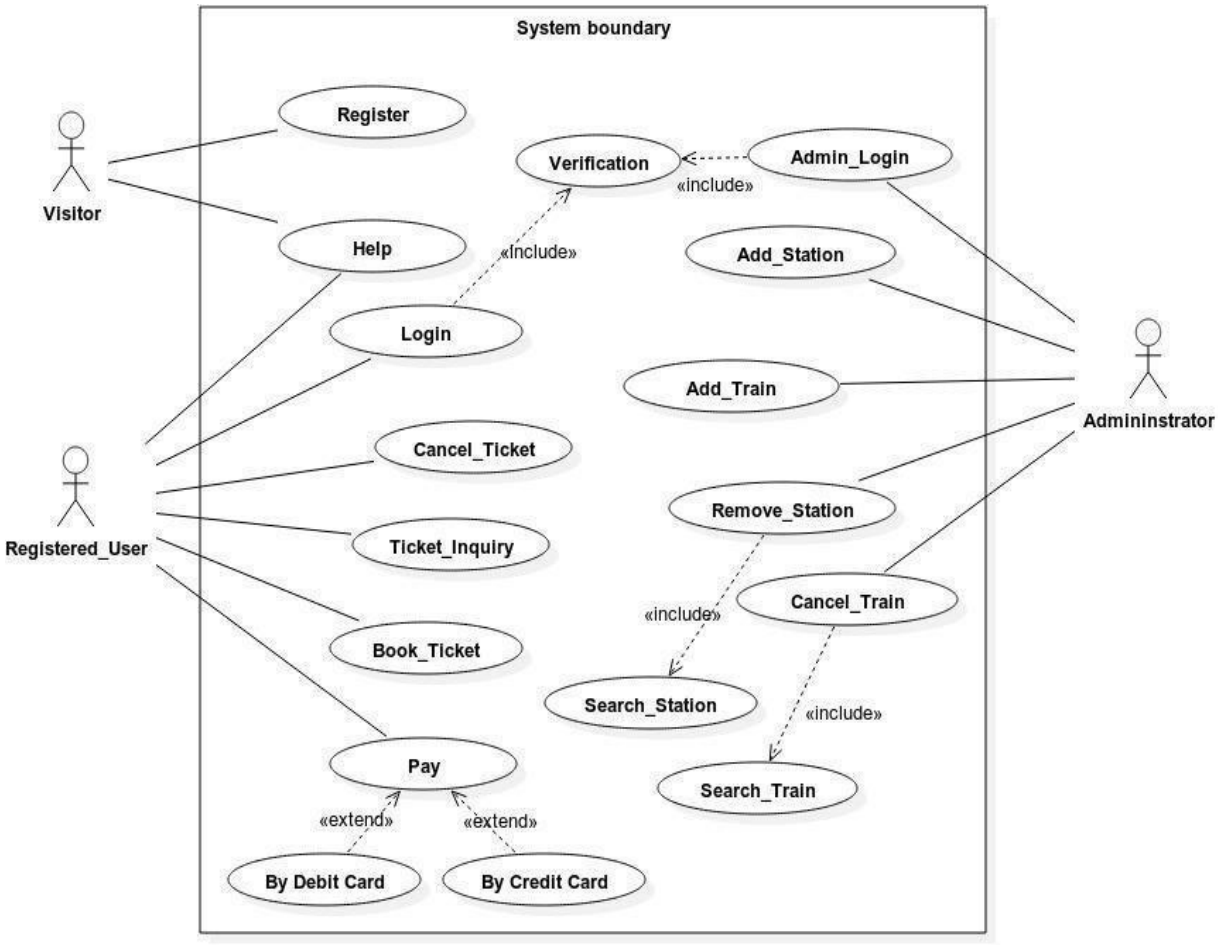
A class diagram is description of set of objects that share same attributes, operations, methods, relationships and semantics.

OBJECT DIAGRAM:



Object diagram describes the static structure of the system at any instant. In our case a registered users “Rites” and “Rajendra” access the database to see the details of book ticket. Rites books a ticket from pine to Mumbai by Sahyadri express. Rajendra books a ticket from Delhi to Mumbai by Rajdhani express. Pondlike is administrator with Admin_id = 1.

USE CASE DIAGRAM:



A use case describes an interaction with actors as a sequence of messages between the system and one or more actors. The term actor includes humans, as well as other computer systems and processes.

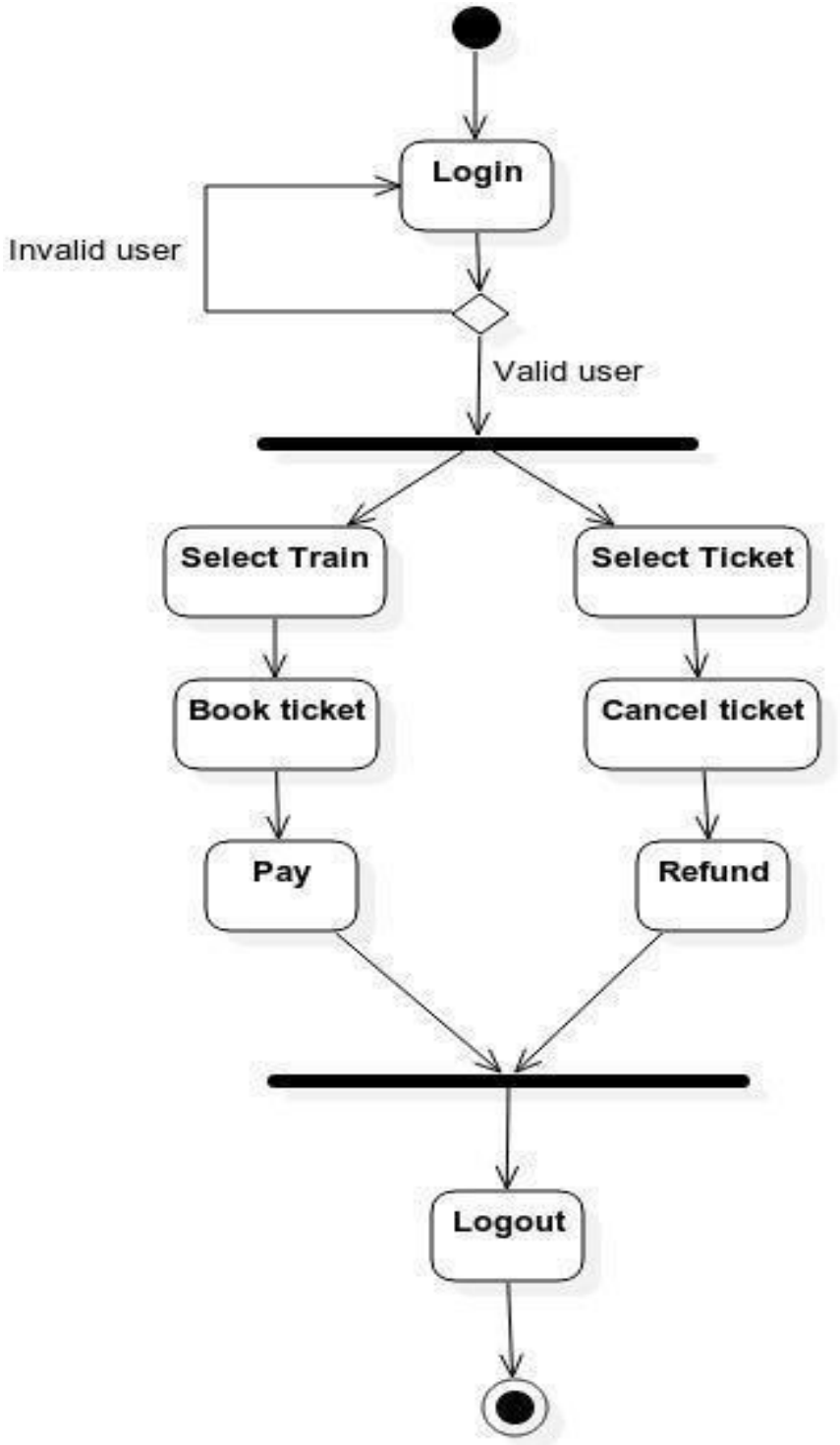
The use case diagram is having following three actors

1. Visitor: - Visitor must register to use the service.

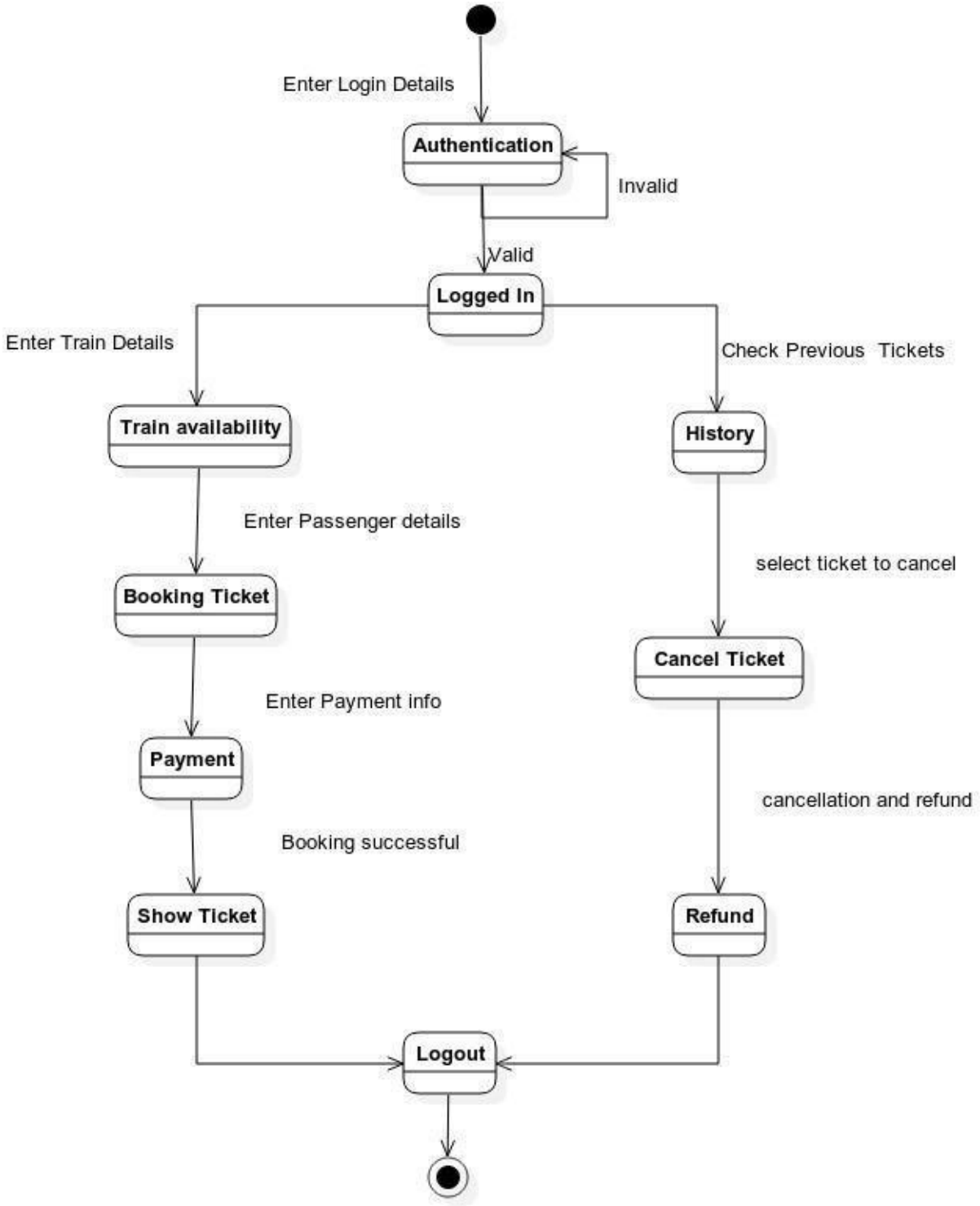
2. Registered user: - A registered user can Login, book ticket, cancel ticket, pay and inquire ticket.

3. Administrator: - Administrator maintains the database. Administrator can add train or station, remove train or station from database.

Activity Diagram



STATECHART DIAGRAM



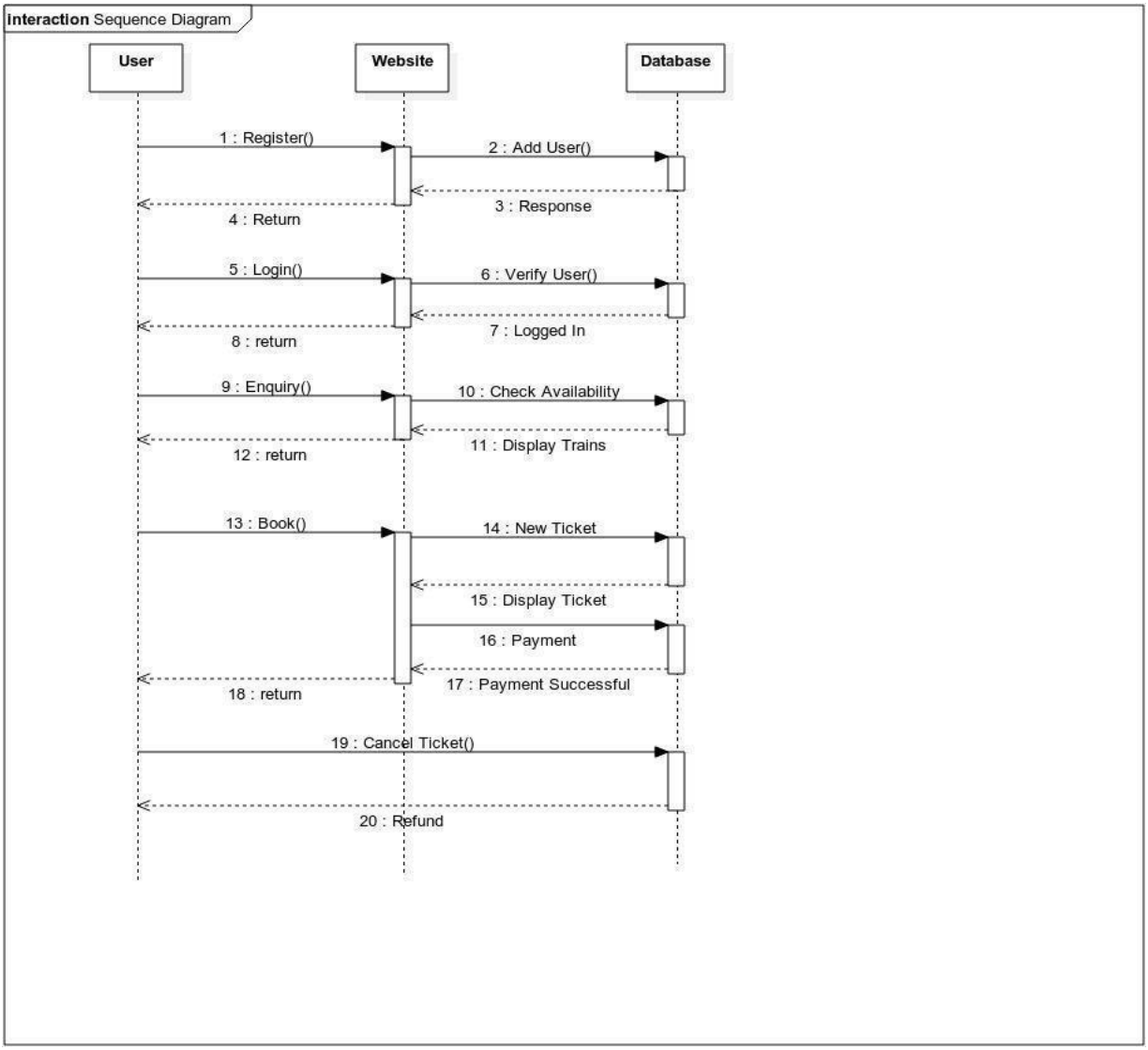
STATES:

1. Authentication and logged in
2. Train availability
3. Book ticket
4. Payment
5. Show ticket
6. History
7. Cancel ticket
8. Refund
9. Logout_

TRANSITIONS:

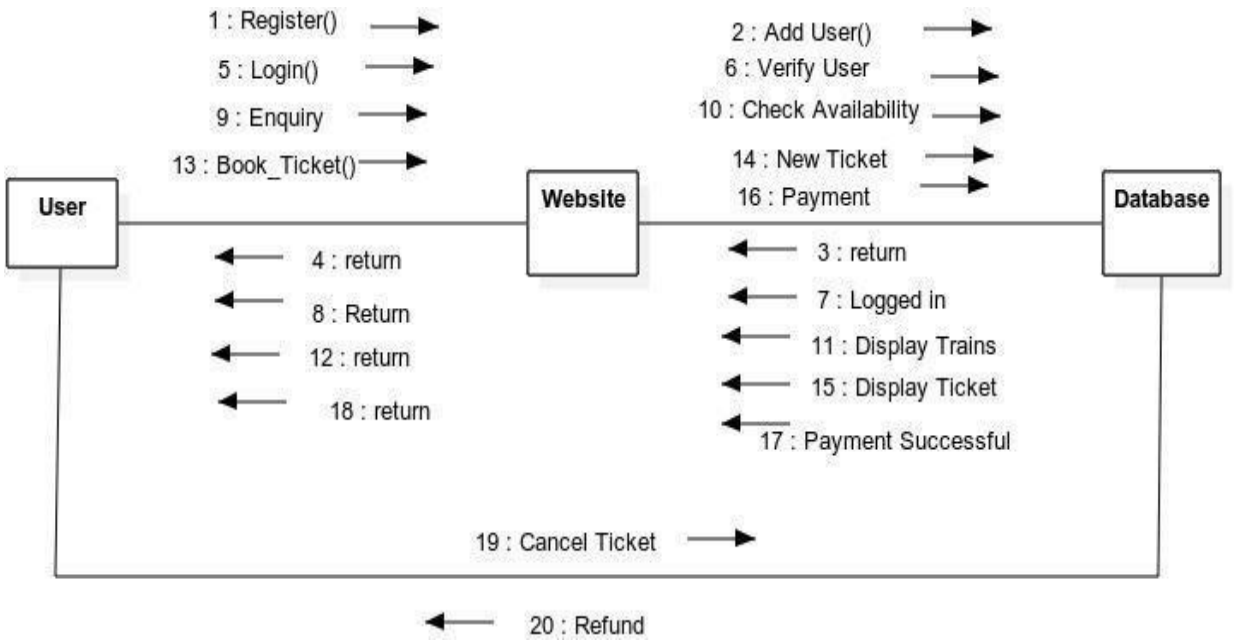
1. Authenticate - Logged in
2. Logged in – Enter train details – Train availability – Enter passenger details – Booking ticket – Enter payment info – Pay – Show ticket - Logout
3. Logged in – Check previous tickets – Cancel ticket – Refund - Logout

SEQUENCE DIAGRAM:



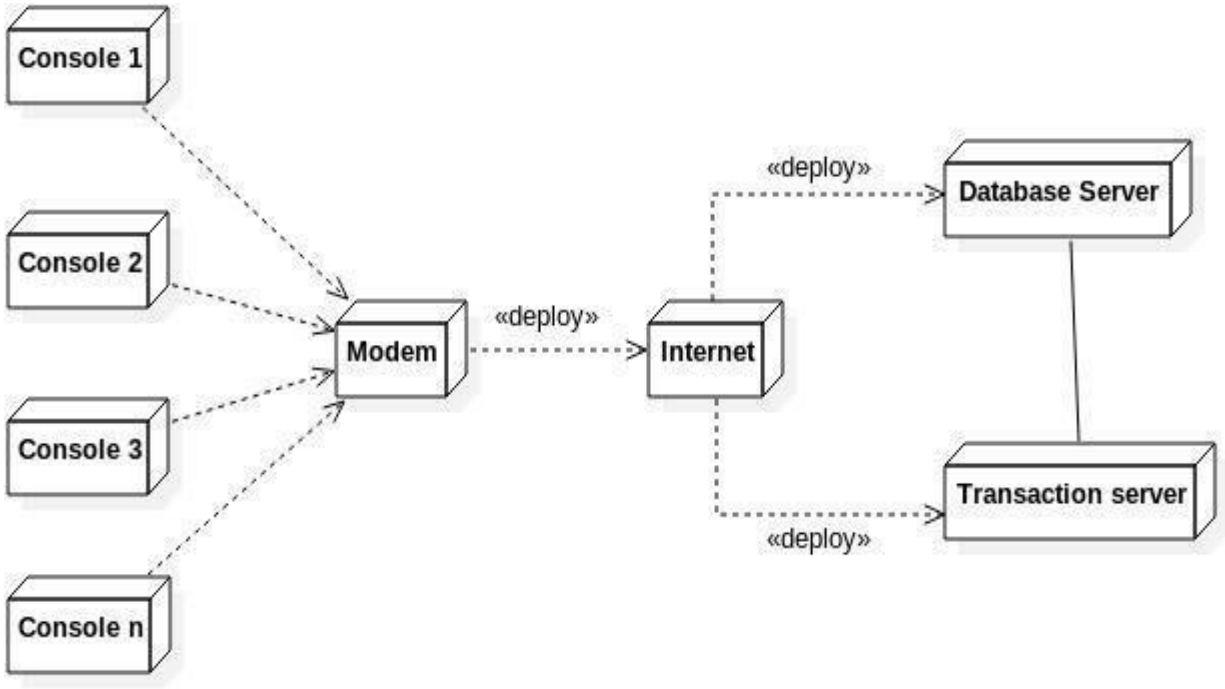
A sequence diagram shows a set of messages arranged in time sequence. Each classifier role is shown as a lifeline—that is, a vertical line that represents the role over time through the entire interaction. Messages are shown as arrows between lifelines. A sequence diagram can show a scenario that is, an individual history of a transaction. User logging in gets authenticated. On successful authentication, he/she can book ticket or cancel ticket. While booking the user will be shown a new ticket and pay accordingly. A message is displayed on successful payment. While canceling a ticket, user gets refund. This whole sequence has been captured by sequence diagram.

COLLABORATION DIAGRAM:



A collaboration is a description of a collection of objects that interact to implement some behavior within a context. It describes a society of cooperating objects assembled to carry out some purpose. A collaboration contains slots that are filled by objects and links at run time. A collaboration slot is called a role because it describes the purpose of an object or link within the collaboration. Given is the collaboration diagram. In which collaboration of user are shown, that include the entire process of user logging, booking the new ticket, canceling ticket.

DEPLOYMENT DIAGRAM:



Deployment diagrams are a set of nodes and their relationships. These nodes are physical entities where the components are deployed. The deployment view represents the arrangement of runtime component instances on node instances. A node is a runtime resource, such as a computer, device, or memory. This view permits the consequences of distribution and resource allocation to be assessed. Deployment diagrams are used for visualizing deployment view of a system. This is generally used by the deployment team. So, the deployment diagrams are used to describe the static view of system. When we are going to build a software intensive system, you have to consider both its logical and physical dimensions. On the logical side, you'll find things such as classes, interfaces, collaborations, interactions, and state machines. On the physical side, we'll find components and node which are used in the deployment diagram.

Program-06

Banking System

ABSTRACT

This project aims at creation of a secure Internet banking system. This will be accessible to all customers who have a valid User Id and Password. This is an approach to provide an opportunity to the customers to have some important transactions to be done from where they are at present without moving to bank. In this project we are going to deal the existing facts in the bank i.e.; the transactions which takes place between customer and bank. We provide a real time environment for the existing system in the bank. We deal in the method transaction in the bank can be made faster and easier that is our project is an internet based computerized approach towards banking.

Modules: 1. Balance enquiry

2. Funds Transfer to another account in the same bank

3. Request for cheque book/change of address/stop payment of Cheques

4. Viewing Monthly and annual statements.

The Project SAFE AND SECURE INTERNET BANKING SYSTEM provides comprehensive electronic fund transfer and payment solutions that enable thousands of Citizens, Financial Institutions and hundreds of businesses the convenience of receiving and transferring their funds online. It's fast, easy and puts you in complete control – you decide who to transfer funds, checking of the account details. Receive and pay all your paper bills at one site – at your bank, credit union.

REQUIREMENTS:

Functional Requirements

- Purpose To register a new customer.
- Inputs The required data for registration of a new customer in the bank (Like Name, Address, Designation etc.).
- A Success Message be displayed on successful registration or else an error message will be displayed.

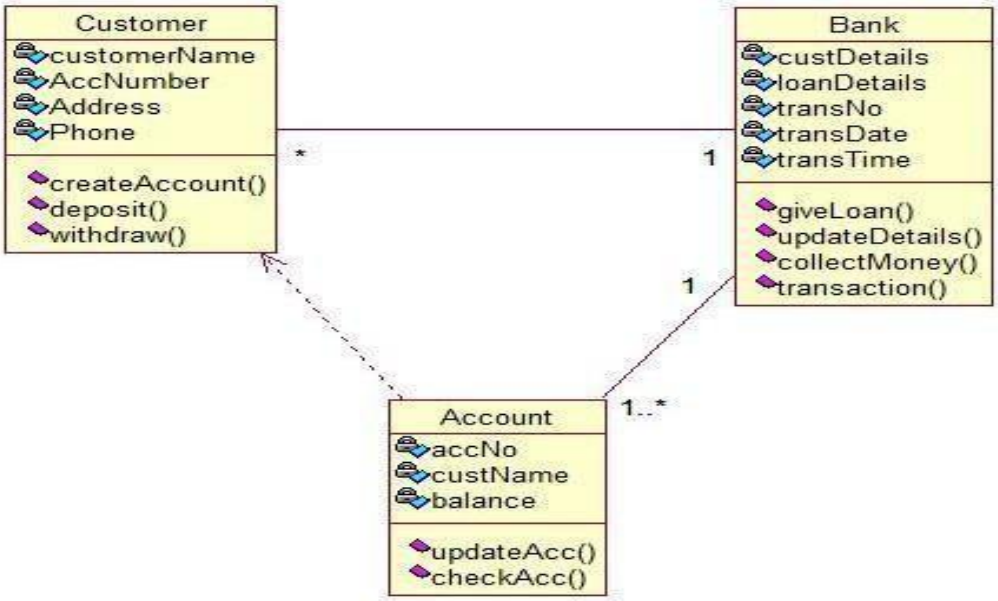
Non-functional Requirements

1. Performance
2. Quality
3. Secure access of confidential data (user's details).
4. 24 X 7availability

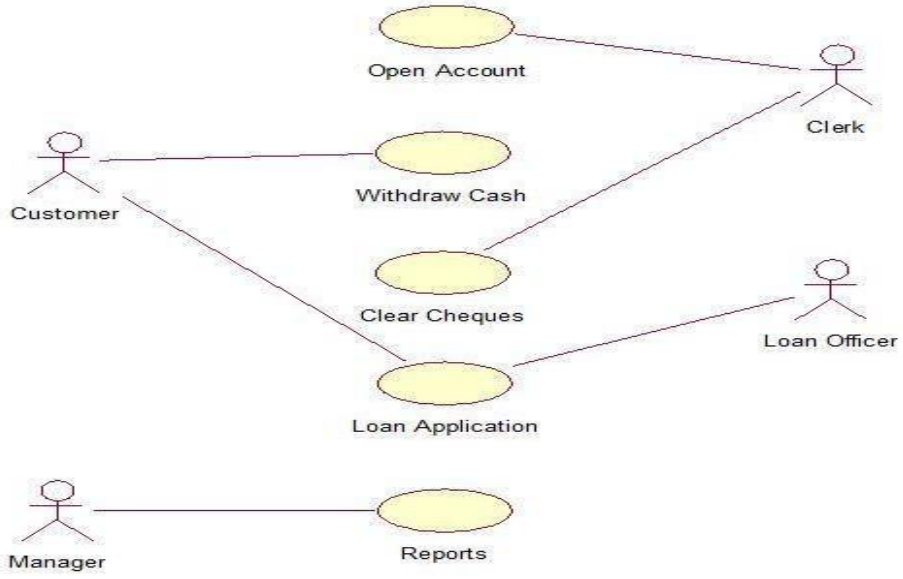
Technical Requirements

1. Browser
2. Server
3. My SQL
4. PHP
5. Java script
6. HTML &CSS

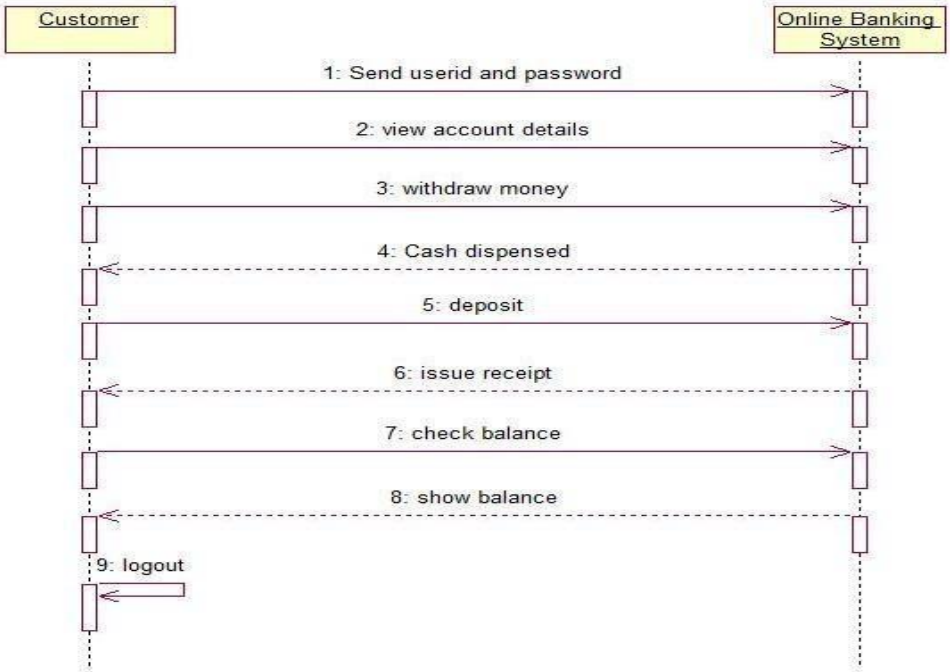
Class Diagram



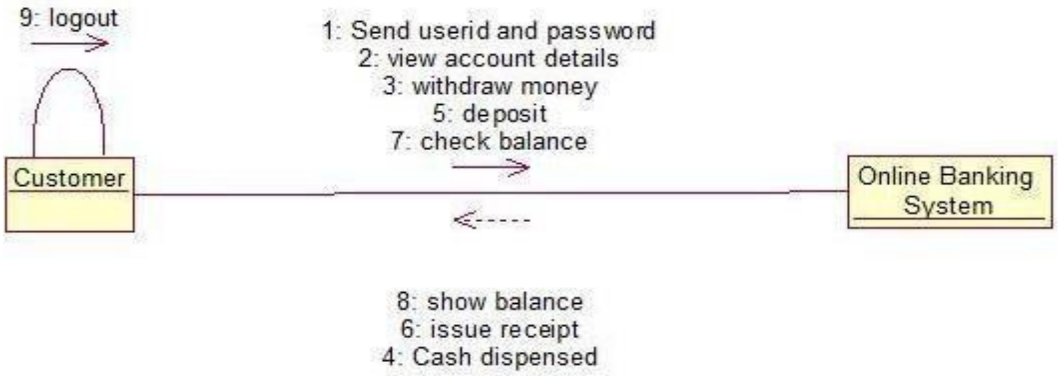
Use Case Diagram



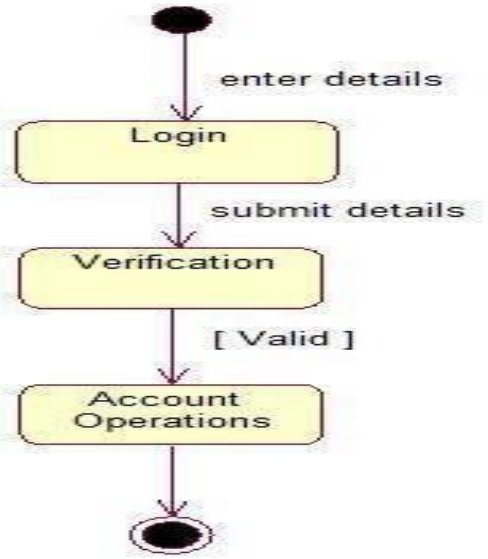
Sequence Diagram



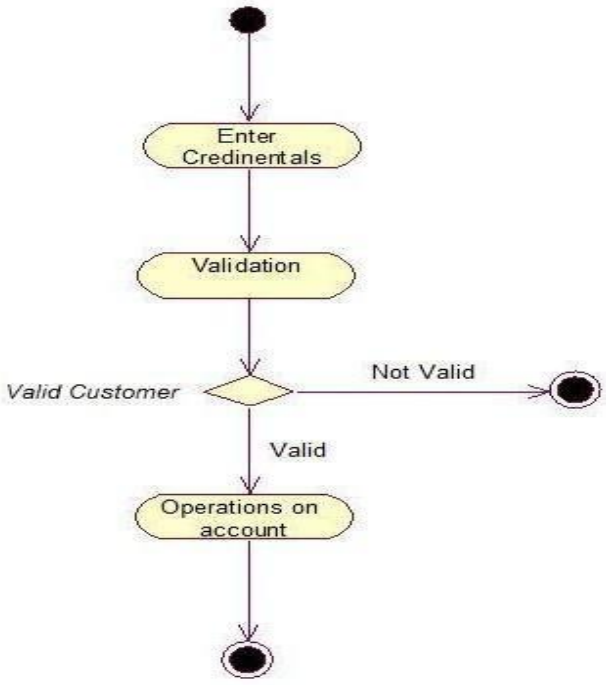
Collaboration Diagram



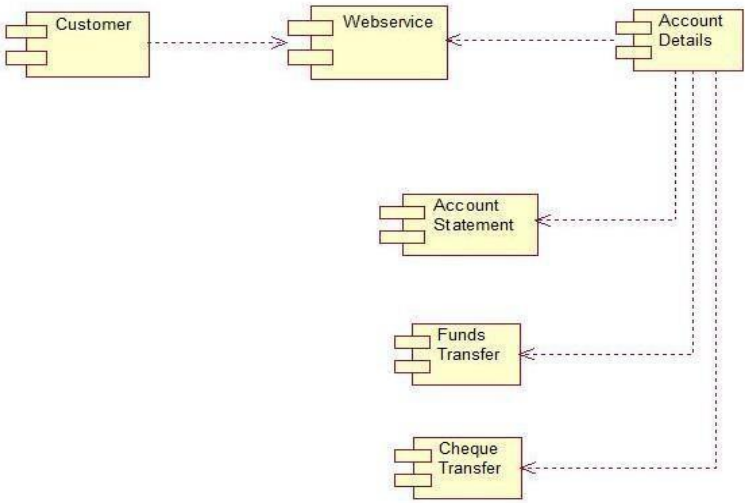
State chart Diagram



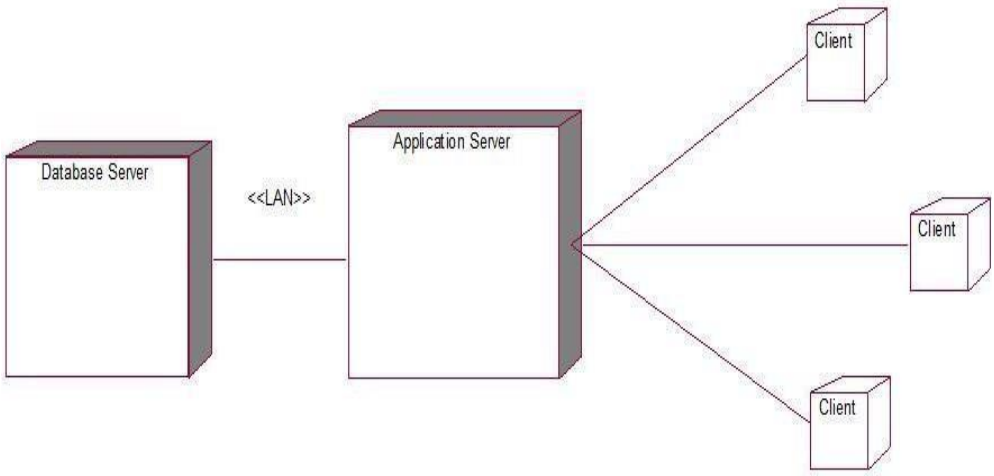
Activity Diagram



Component Diagram



Deployment Diagram



Program -07
To design a Document Editor

This case study is the design of a "What-You-See-Is-What-You-Get" (or "WYSIWYG") document editor called **Lexi**.[1] We'll see how design patterns capture solutions to design problems in Lexi and applications like it. By the end of this chapter, you will have gained experience with eight patterns, learning them by example.

In the given figure depicts Lexi's user interface. A WYSIWYG representation of the document occupies the large rectangular area in the center. The document can mix text and graphics freely in a variety of formatting styles. Surrounding the document are the usual pull-down menus and scroll bars, plus a collection of page icons for jumping to a particular page in the document.

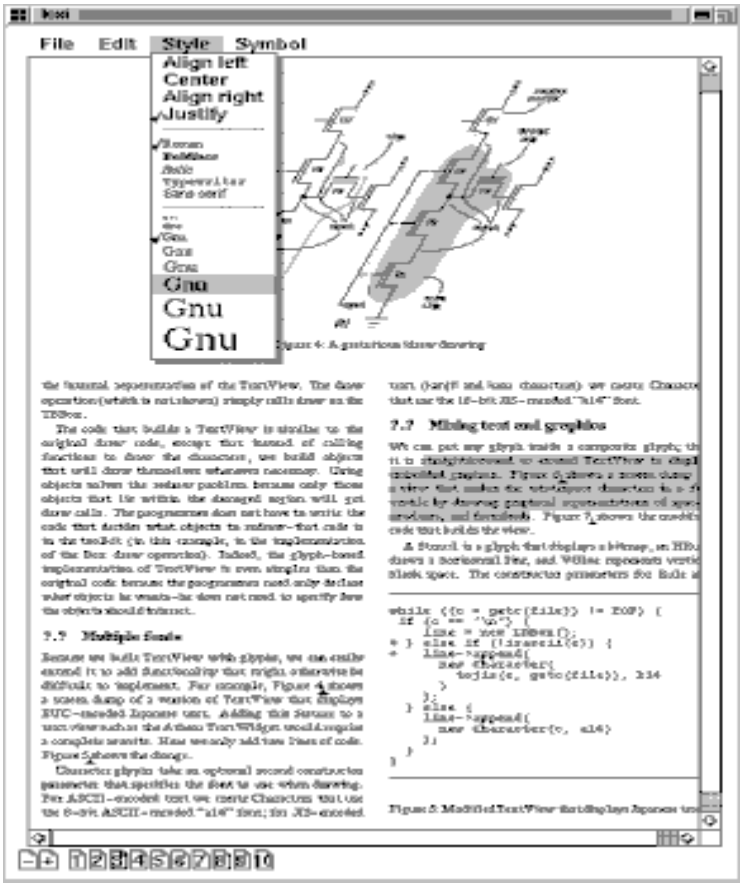


Figure: Lexi's user interface

Design Problems

We will examine seven problems in Lexi's design:

1. *Document structure.* The choice of internal representation for the document affects nearly every aspect of Lexi's design. All editing, formatting, displaying, and textual analysis will require traversing the representation. The way we organize this information will impact the design of the rest of the application.
2. *Formatting.* How does Lexi actually arrange text and graphics into lines and columns? What objects are responsible for carrying out different formatting policies? How do these policies interact with the document's internal representation?
3. *Embellishing the user interface.* Lexi's user interface includes scroll bars, borders, and drop shadows that embellish the WYSIWYG document interface. Such embellishments are likely to change as Lexi's user interface evolves. Hence, it's important to be able to add and remove embellishments easily without affecting the rest of the application.
4. *Supporting multiple look-and-feel standards.* Lexi should adapt easily to different look-and-feel standards such as Motif and Presentation Manager (PM) without major modification.
5. *Supporting multiple window systems.* Different look-and-feel standards are usually implemented on different window systems. Lexi's design should be as independent of the window system as possible.
6. *User operations.* Users control Lexi through various user interfaces, including buttons and pull-down menus. The functionality behind these interfaces is scattered throughout the objects in the application. The challenge here is to provide a uniform mechanism both for accessing this scattered functionality and for undoing its effects.
7. *Spelling checking and hyphenation.* How does Lexi support analytical operations such as checking for misspelled words and determining hyphenation points? How can we minimize the number of classes we have to modify to add a new analytical operation?

Document Structure

A document is ultimately just an arrangement of basic graphical elements such as characters, lines, polygons, and other shapes. These elements capture the total information content of the document. Yet an author often views these elements not in graphical terms but in terms of the document's physical structure—lines, columns, figures, tables, and other substructures.[2] In turn, these substructures have substructures of their own, and so on.

Lexi's user interface should let users manipulate these substructures directly. For example, a user should be able to treat a diagram as a unit rather than as a collection of individual graphical primitives. The user should be able to refer to a table as a whole, not as an unstructured mass of text and graphics. That helps make the interface simple and intuitive. To give Lexi's implementation similar qualities, we'll choose an internal representation that matches the document's physical structure.

In particular, the internal representation should support the following:

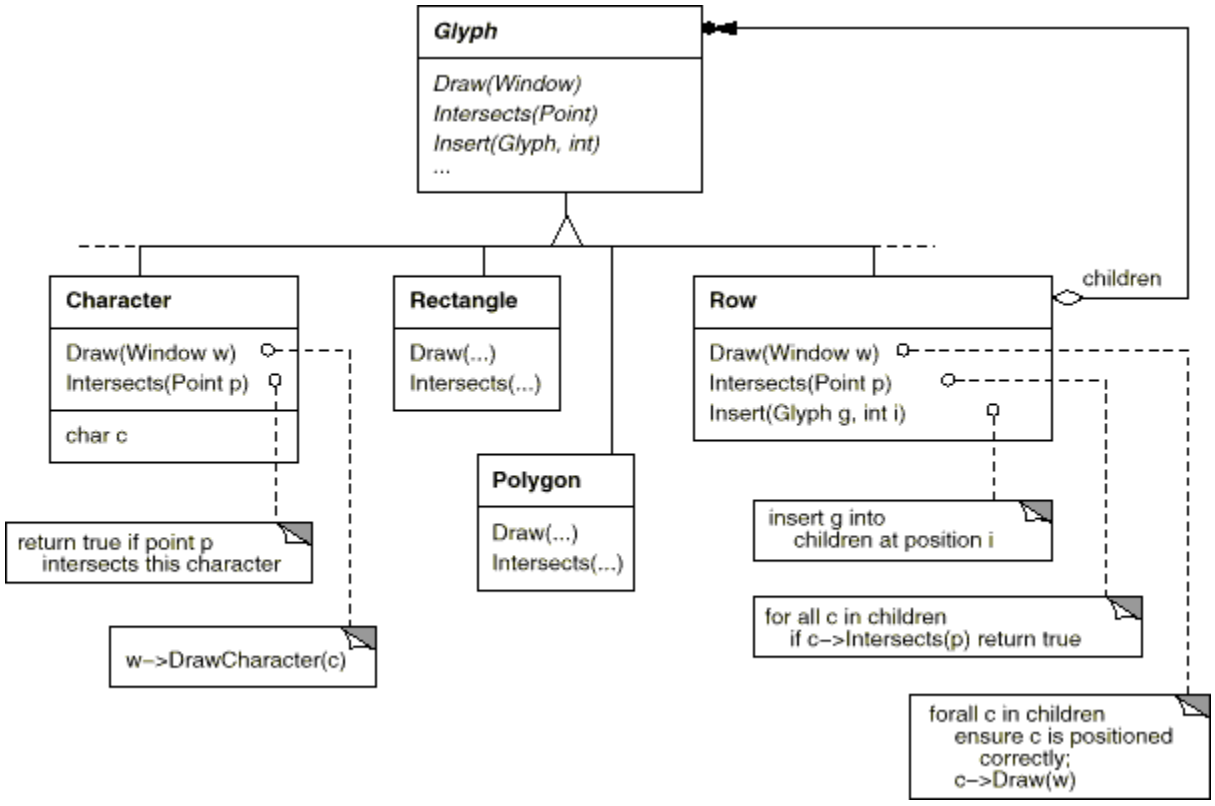
- Maintaining the document's physical structure, that is, the arrangement of text and graphics into lines, columns, tables, etc.
- Generating and presenting the document visually.
- Mapping positions on the display to elements in the internal representation. This lets Lexi determine what the user is referring to when he points to something in the visual representation.

In addition to these goals are some constraints. First, we should treat text and graphics uniformly. The application's interface lets the user embed text within graphics freely and vice versa. We should avoid treating graphics as a special case of text or text as a special case of graphics; otherwise, we'll end up with redundant formatting and manipulation mechanisms. One set of mechanisms should suffice for both text and graphics.

Second, our implementation shouldn't have to distinguish between single elements and groups of elements in the internal representation. Lexi should be able to treat simple and complex elements uniformly, thereby allowing arbitrarily complex documents. The tenth element in line five of column two, for instance, could be a single character or an intricate diagram with many sub elements. As long as we know this element can draw itself and specify its dimensions, its complexity has no bearing on how and where it should appear on the page.

Glyphs

We'll define a **Glyph** abstract class for all objects that can appear in a document structure.[3] Its subclasses define both primitive graphical elements (like characters and images) and structural elements (like rows and columns). Figure depicts a representative part of the Glyph class hierarchy, and presents the basic glyph interface in more detail using C++ notation.



Responsibility	Operations
appearance	virtual void Draw (Window*) virtual void Bounds (Rect&)
hit detection	virtual bool Intersects (const Point&)
structure	virtual void Insert (Glyph*, int) virtual void Remove (Glyph*) virtual Glyph* Child(int) virtual Glyph* Parent ()

Table: Basic glyph interface

Formatting

We've settled on a way to *represent* the document's physical structure. Next, we need to figure out how to construct a *particular* physical structure, one that corresponds to a properly formatted document. Representation and formatting are distinct: The ability to capture the document's physical structure doesn't tell us how to arrive at a particular structure. This responsibility rests mostly on Lexi. It must break text into lines, lines into columns, and so on, taking into account the user's higher-level desires. For example, the user might want to vary margin widths, indentation, and tabulation; single or double space; and probably many other formatting constraints.[6] Lexi's formatting algorithm must take all of these into account.

By the way, we'll restrict "formatting" to mean breaking a collection of glyphs into lines. In fact, we'll use the terms "formatting" and "line breaking" interchangeably. The techniques we'll discuss apply equally well to breaking lines into columns and to breaking columns into pages.

Compositor and Composition

We'll define a **Compositor** class for objects that can encapsulate a formatting algorithm. The interface lets the compositor know *what* glyphs to format and *when* to do the formatting. The glyphs it formats are the children of a special Glyph subclass called **Composition**. A composition gets an instance of a Compositor subclass (specialized for a particular line breaking algorithm) when it is created, and it tells the compositor to Compose its glyphs when necessary, for example, when the user changes a document. Figure depicts the relationships between the Composition and Compositor classes.

Responsibility	Operations
what to format	void Set Composition (Composition*)
when to format	virtual void Compose ()

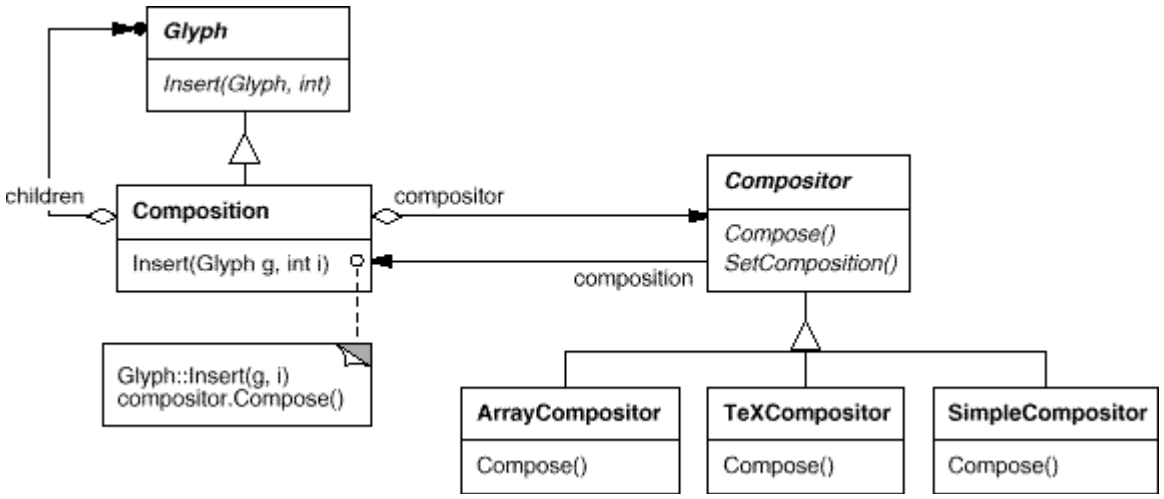


Figure: Composition and Compositor class relationships

An unformatted Composition object contains only the visible glyphs that make up the document's basic content. It doesn't contain glyphs that determine the document's physical structure, such as Row and Column. The composition is in this state just after it's created and initialized with the glyphs it should format. When the composition needs formatting, it calls its compositor's Compose operation. The compositor in turn iterates through the composition's children and inserts new Row and Column glyphs according to its line breaking algorithm.[7] Figure shows the resulting object structure. Glyphs that the compositor created and inserted into the object structure appear with gray backgrounds in the figure.

Monoglyph

We can apply the concept of transparent enclosure to all glyphs that embellish other glyphs. To make this concept concrete, we'll define a subclass of Glyph called **MonoGlyph** to serve as an abstract class for "embellishment glyphs," like Border . MonoGlyph stores a reference to a component and forwards all requests to it. That makes MonoGlyph totally transparent to clients by default. For example, MonoGlyph implements the Draw operation like this:

```
void MonoGlyph::Draw (Window* w) {
_component->Draw(w);
}
```

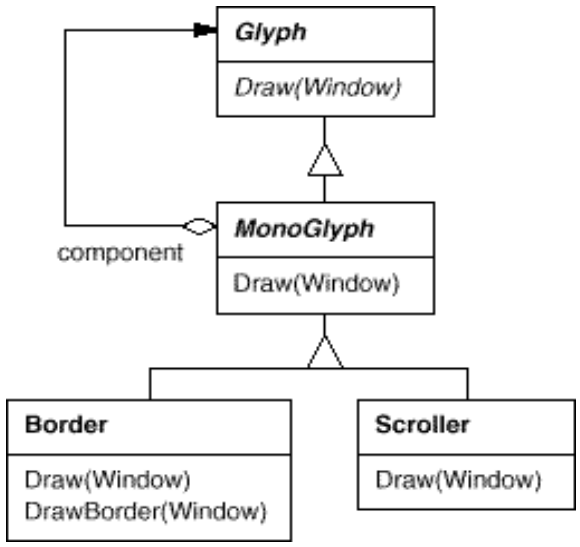


Figure: MonoGlyph class relationships

MonoGlyph subclasses re implement at least one of these forwarding operations. Border::Draw, for instance, first invokes the parent class operation MonoGlyph::Draw on the component to let the component do its part—that is, draw everything but the border. Then Border::Draw draws the border by calling a private operation called DrawBorder, the details of which we'll omit:

```
void Border::Draw (Window* w) {
MonoGlyph::Draw(w);
DrawBorder(w);
}
```

Notice how Border::Draw effectively *extends* the parent class operation to draw the border. This is in contrast to merely *replacing* the parent class operation, which would omit the call to MonoGlyph::Draw.

Another MonoGlyph subclass appears in . **Scroller** is a MonoGlyph that draws its component in different locations based on the positions of two scroll bars, which it adds as embellishments. When Scroller draws its component, it tells the graphics system to clip to its bounds. Clipping parts of the component that are scrolled out of view keeps them from appearing on the screen.

Visitor Pattern

What we've described here is an application of the Visitor (331) pattern. The Visitor class and its subclasses described earlier are the key participants in the pattern. The Visitor pattern captures the technique we've used to allow an open-ended number of analyses of glyph structures without having to change the glyph classes themselves. Another nice feature of visitors is that they can be applied not just to composites like our glyph structures but to *any* object structure. That includes sets, lists, even directed-acyclic graphs. Furthermore, the classes that a visitor can visit needn't be related to each other through a common parent class. That means visitors can work across class hierarchies.

An important question to ask yourself before applying the Visitor pattern is, Which class hierarchies change most often? The pattern is most suitable when you want to be able to do a variety of different things to objects that have a stable class structure. Adding a new kind of visitor requires no change to that class structure, which is especially important when the class structure is large. But whenever you add a subclass to the structure, you'll also have to update all your visitor interfaces to include a Visit... operation for that subclass. In our example that means adding a new Glyph subclass called Foo will require changing Visitor and all its subclasses to include a VisitFoo operation. But given our design constraints, we're much more likely to add a new kind of analysis to Lexi than a new kind of Glyph. So the Visitor pattern is well-suited to our needs.

Concluding Remarks

- design reuse
- uniform design vocabulary
- understanding, restructuring, & team
- communication provides the basis for automation
- a “new” way to think about design