

## **1. Introduction**

Distributed Ledger Technology (DLT) is a digital transaction record system that allows multiple parties in different locations to have access, validation, and data manipulation across a networked database. The DLT infrastructure and protocols were designed to facilitate data record management among several distributed users by recording the history of modifications, ensuring data reliability and providing immutable records. This technology is enabled by cryptography, consensus algorithms, validity rules, governance and smart contracts. Blockchain is the most famous form of DLT that has attracted the interest of many researchers and practitioners leading to technological advancements in the area and the development of various types of blockchains and DLT systems that serve different applications. Blockchain has gained popularity owing to its applications in the field of cryptocurrencies and digital asset management. The proliferation of cryptocurrency blockchain-based applications and the wide variety of digital currencies have necessitated the development of asset transfer and exchange techniques among different cryptocurrencies.

## 1.1 Purpose

The primary purpose of this project is to address the challenge of interoperability between different permissioned blockchain networks, particularly in the context of business and enterprise applications. As enterprises increasingly adopt blockchain technology for secure and transparent transactions, the need for seamless communication across heterogeneous blockchain networks has become crucial. Permissioned blockchains, while providing a high degree of security and control over participant access, often operate in silos, limiting the scope of their use cases. The proposed blockchain oracle interoperability technique aims to bridge these silos by enabling cross-network transactions and smart contract executions without compromising the decentralized nature of the systems. By leveraging blockchain oracles, which act as intermediaries between disparate blockchain platforms, this project strives to create a solution that ensures smooth and private communication between permissioned blockchains like Hyperledger Fabric and R3 Corda, thereby expanding the potential applications of blockchain technology in enterprise environments. This project also seeks to explore and optimize the efficiency of cross-network transactions in permissioned blockchain environments.

## 1.2 Scope

The scope of this project revolves around the design, implementation, and evaluation of a blockchain oracle interoperability technique specifically tailored for permissioned blockchain platforms. The focus is on enabling seamless communication and transaction execution between two widely used permissioned blockchain platforms—Hyperledger Fabric and R3 Corda. By leveraging blockchain oracles, the project seeks to bridge the communication gap between heterogeneous blockchain systems, ensuring that transactions can flow securely and efficiently across different networks without requiring changes to the underlying blockchain infrastructure. The project also explores the preservation of the decentralized features of the blockchain networks involved, as each participating blockchain will have its own oracle nodes to manage interoperability. The scope includes building a prototype of this solution and conducting performance evaluations to analyze the impact of the interoperability technique on cross-network transaction latency.

### **1.3 Need For System**

In today's rapidly evolving digital landscape, organizations are increasingly turning to blockchain technology to manage sensitive data and transactions securely. However, one of the primary challenges that enterprises face when adopting blockchain systems is ensuring interoperability between different blockchain platforms. Many businesses employ multiple permissioned blockchains, each serving specific functions, such as Hyperledger Fabric for supply chain tracking or R3 Corda for financial transactions. The inability of these platforms to communicate effectively with each other creates silos, preventing seamless data exchange and undermining the potential of blockchain technology. The need for a system that enables cross-network transactions and interactions between heterogeneous blockchain platforms is critical to overcoming these limitations. This is particularly important for industries such as finance, healthcare, and logistics, where the integration of different blockchain solutions is necessary to achieve comprehensive, real-time visibility and streamline business processes across various stakeholders.

#### **1.3.1 Existing System**

The current landscape of permissioned blockchains presents significant limitations in terms of interoperability. Permissioned blockchains offer secure, private ledgers for managing organizational data but typically operate in isolation from one another. This lack of connectivity restricts their ability to exchange information or execute transactions across different networks. Consequently, businesses face challenges in integrating multiple permissioned blockchains, leading to inefficiencies in data management and increased complexity. Existing solutions for blockchain interoperability often overlook the specific needs of permissioned networks, focusing instead on public blockchains or lacking standardized methods for cross-network interactions. As a result, these systems experience disadvantages such as high transaction latency and security concerns due to the absence of robust interoperability mechanisms.

## Disadvantages of Existing System

- **Inadequate Security:** Many cloud storage solutions lack strong encryption and security protocols, exposing sensitive data to unauthorized access.
- **Poor User Management:** Insufficient permissions control can lead to data mishandling and unauthorized sharing of files.
- **Inefficient File Retrieval:** Existing systems often do not employ optimized search algorithms, making it difficult for users to locate files quickly.
- **Limited Access Control:** Users may not have clear visibility or control over who can access their data, increasing the risk of breaches.
- **Vulnerability to Data Breaches:** Without robust security measures, existing systems are at higher risk of data breaches and cyberattacks.
- **Manual Encryption Processes:** Users may need to manually encrypt files before uploading, increasing the risk of human error.
- **Lack of Transparency:** Users often have limited insight into data storage practices and security measures employed by service providers.
- **Ineffective Monitoring:** Many systems do not provide adequate tracking of file access and modifications, hindering accountability.
- **Scalability Issues:** Some solutions may struggle to accommodate growing data needs without compromising security.
- **Limited Integration Capabilities:** Existing systems may not easily integrate with other tools or platforms, limiting their effectiveness in diverse environments.

### 1.3.2 Proposed System

In response to these challenges, the proposed system introduces a blockchain oracle interoperability technique specifically designed for permissioned blockchains. This technique enables seamless communication and transaction exchanges between different permissioned networks, addressing the isolation issue prevalent in existing systems. By leveraging blockchain oracles, the proposed system facilitates efficient cross-network transactions by using the SHA - 256, significantly improving operational efficiency. The technique also enhances data integration, reduces

transaction latency, and ensures that interactions are carried out with high security and privacy. Additionally, the proposed system supports scalability, making it suitable for dynamic and evolving enterprise applications.

### **Advantages of Proposed System**

- **Enhanced Data Security:** Utilizes hybrid cryptography for robust encryption, protecting sensitive information from unauthorized access.
- **Automated Encryption and Decryption:** Streamlines file security during uploads and downloads, minimizing the risk of human error.
- **UserFriendly Interface:** Offers an intuitive design that simplifies navigation for data owners, users, and administrators.
- **Efficient File Retrieval:** Implements optimized search algorithms, such as binary search trees, for quick and accurate file access.
- **Granular Access Control:** Allows administrators to effectively manage user permissions.

## 1.4 System Architecture

The architecture diagram consists of three primary modules: the Admin Module, Company Module, and Authority Module, all integrated into a blockchain-based system. The **Admin Module** acts as the central hub, overseeing system performance, managing company and authority roles, and ensuring smooth blockchain operations. Admins have control over permissions, activity logs, and announcements, ensuring the proper functioning of the network. The **Company Module** allows businesses to register, manage shipments, track products, and file damage claims. Companies interact directly with the blockchain to record transaction details securely. The **Authority Module** ensures the accuracy of shipments by validating product conditions upon receipt and updating shipment statuses, which are then sent back to the shipping company. Authorities confirm or reject claims based on product conditions and have oversight of all transactions. These modules work together through the blockchain, ensuring secure, transparent, and verifiable data storage, while providing real-time updates and validation of shipments across companies and authorities.

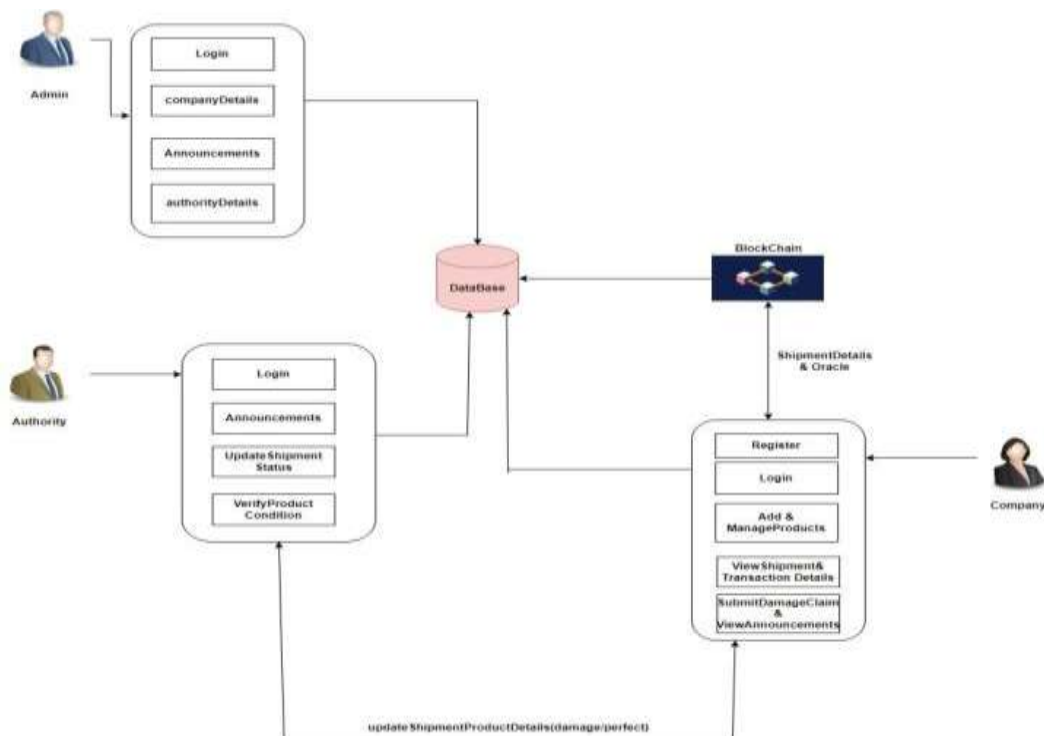


Fig.No.1.4.1 Architecture diagram

## 2. Software Requirement Analysis and Specifications

### 2.1 Product Perspective

An essential aspect necessitates assurance, which involves bridging the gap between product management and development. It entails thoroughly defining a product in terms of its associated requirements, encompassing all the necessary specifications that must be explicitly described and continuously available.



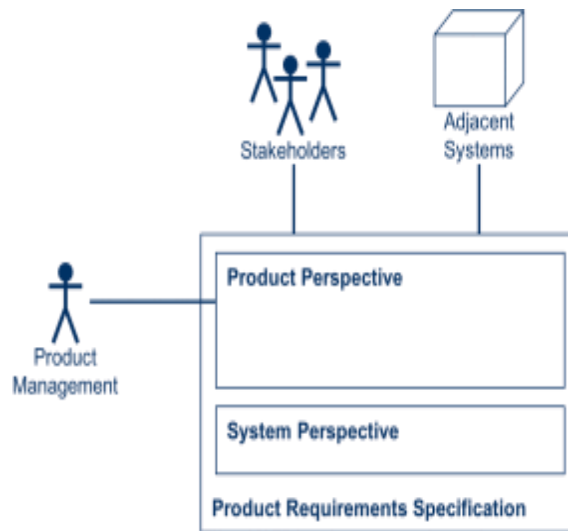
**Fig No. 2.1** Product Perceptive

### 2.2 Product Function

In product management, the product requirements specification serves as the central tool, fulfilling the following purposes:

- It articulates how product management addresses partner needs and requests.
- It communicates to development teams what the product or new features to be developed will entail.
- The accompanying figure illustrates these relationships, focusing on the product perspective.
- Product management is responsible for defining a product perspective that aligns with the expectations and requirements of the product's stakeholders.

Additionally, it should consider and serve relevant relationships with adjacent systems.



**Fig No. 2.2.1** Characteristics of product perspective

The key characteristics of the product perspective include:

- Describes the external view of the product.
- Answers the question "What"...
- Represents what the product is.
- Describes what the product does.
- Addresses both the business and usage aspects of the product.
- Utilizes problem domain language and concepts.
- Presents a "Black Box" view, focusing on the external behavior and functionality of the product without delving into its internal workings.

## 2.3 User Characteristics Definition

This section outlines the characteristics of users in three distinct contexts: the end user who interacts with the machine translation system; the end user of the final product of the translation process, which may include post- editing; the organization producing the machine translation system.



Relevant stakeholders may include:

- Interpreters
- Post-editors
- Translation buyers
- Translation managersHigher-level

## 2.4 Constrains

Pragmatic requirements are product features or functionalities that developers must implement to enable users to accomplish their tasks. It is essential to clearly define them for both the development team and the stakeholders. Typically, pragmatic requirements describe system behavior under unambiguous conditions. For example:

- The system sends a confirmation request after the user enters personal information.
- A search feature allows a user to search among multiple orders to allocate a provided receipt.
- The system sends a confirmation email when a new user account is created.
- Types of functional requirements and their specifics:

Functional requirements can be categorized by various criteria. For instance, they can be grouped based on the roles a given component should perform in the final product. Naturally, these requirements may vary depending on the product under development, but examples of functional requirements might include:

- Authentication
- Authorization levels
- Compliance with regulations or standards
- External interfaces

## **2.5 Modules**

- **Admin Module**
- **Company Module**
- **Authority Module**

### **Admin Module**

The Admin Module serves as the backbone of the system, providing oversight and management of the entire blockchain network. Admins can log in using their credentials and access a comprehensive dashboard that displays critical system information. They can view the details of companies and the authorities operating within the platform, as well as any updates or announcements related to system maintenance or other important notices. This module offers features like company management (removing, and modifying company roles), network monitoring (to ensure blockchain operations are running smoothly), and authority management (ensuring compliance with system protocols). Admins can also have direct control over company permissions, making sure that only authorized personnel have access to sensitive data. The Admin Module acts as a central hub to ensure the overall functionality of the system, streamline workflows, and keep track of any system issues or updates.

### **Key Features**

- Company management for companies and authorities.
- Real-time monitoring of blockchain network performance.
- Viewing of company and authority details.
- Announcements for maintenance and updates.
- Permission and access control management.
- Activity logs to track system usage and performance.

## **Company Module**

The Company Module allows companies to register and log in with their credentials, granting them access to their specific blockchain network.

Upon logging in, companies can manage the details of the products they ship, including shipment status and relevant transaction information. They can generate blocks for every shipment, ensuring the details are securely recorded on the blockchain. If products are damaged during shipment, the company can file a claim for the damaged goods. The module also enables companies to interact with other companies, sending products and receiving data about transactions. The shipment details, including product information, destination, and tracking, are securely stored on the blockchain. Additionally, companies can view their shipment history and transaction details to keep track of their operations. The Company Module empowers businesses to manage product deliveries, claims, and shipment tracking seamlessly, using blockchain to ensure security and transparency in every transaction.

### **Key Features**

- User registration and login for secure access.
- Blockchain-based shipment tracking and block generation.
- Damage claim submission process for damaged goods.
- Viewing of company details and transaction history.
- Real-time updates on shipped products and claims.
- Transaction management for sending and receiving goods.

## **Authority Module**

The Authority Module plays a crucial role in ensuring the legitimacy and accuracy of the shipments in the system. Authorities, who act as external validators, log in to this module to access shipment data provided by the companies. The authority is responsible for verifying whether the products have been correctly received by the receiving company or if they are damaged. Once the authority receives the verification details, they update the status of the shipment, indicating whether it is damaged or in good condition.

## Key Features

- Secure login for authority verification and validation.
- Viewing of shipment details across multiple companies.
- Verification of product condition (received in good condition or damaged).
- Update of shipment status, which is relayed to the shipping company.
- Confirmation and approval of claims based on product condition.
- Oversight and audit of transaction and shipment data.

## 2.6 Functional and Non Functional Requirements

### 2.6.1 Functional Requirements

Functional requirements delineate the intended actions of the system and can be segmented into various categories:

- **Inputs:** Specifies the data the system should accept.
- **Outputs:** Defines the results or data the system should generate.
- **Data Storage:** Determines the data that must be stored by the system.

Computations: Describes the calculations or processes the system needs to perform.

### Input Design

Input design facilitates interaction between the information system and users, establishing processes for data preparation and entry. It involves methods to transform transaction data into usable formats, such as reading from documents or direct entry. Input design aims to streamline processes, minimize errors, and ensure user security and privacy. Key considerations include:

- Defining input data requirements.
- Organizing and coding data

## 2.6.2 Non-functional Requirement

Non-functional requirements (NFRs) address fundamental aspects of software systems, focusing on qualities beyond specific functionalities. Failure to address NFRs adequately can result in user dissatisfaction, software conflicts, and increased time and cost to rectify issues.

### Types of Non-functional Requirements

- **Scalability:** The system's ability to handle increasing loads without compromising performance.
- **Reliability:** The system's ability to perform consistently and predictably under various conditions.
- **Regulatory Compliance:** Ensuring the system adheres to relevant regulations or standards.
- **Maintainability:** Ease of maintaining and updating the system over time.
- **Serviceability:** The ease with which the system can be repaired or serviced.
- **Utility:** The usefulness and value provided by the system to users.
- **Availability:** Ensuring the system is accessible and operational when needed.
- **Usability:** The system's ease of use and user experience.
- **Interoperability:** The system's ability to interact and operate with other systems or components.
- **Environmental Considerations:** Factors related to the environmental impact of the system, such as energy consumption or sustainability.

## 2.7 System Specification

### 2.7.1 Hardware Requirements

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the system does and not how it should be implemented.

PROCESSOR	:	INTEL CORE -I5
RAM	:	4GB RAM
MONITOR	:	15" COLOR
HARD DISK	:	250 GB

### 2.7.2 Software Requirements

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the team's and tracking the team's progress throughout the development activity.

Front End	:	Html,Css,Reactjs, Bootstrap
Back End	:	Springboot,Java
Database	:	My Sql

## 2.8 SDLC Methodologies

Software Development Life Cycle Models and Methodologies



**Fig No: 2.8 SDLC**

The Software Development Life Cycle (SDLC) is a series of stages that provide a structured approach to the software development process. It encompasses understanding the business requirements, eliciting needs, converting concepts into functionalities and features, and ultimately delivering a product that meets business needs. A proficient software developer should possess adequate knowledge to select the appropriate SDLC model based on project context and business requirements.

Therefore, it is essential to select the right SDLC model tailored to the specific concerns and requirements of the project to ensure its success. To explore more about choosing the right SDLC model, you can follow [this link](#) for additional information. Furthermore, to delve deeper into software lifecycle testing and SDLC stages, follow the highlighted links [here](#).

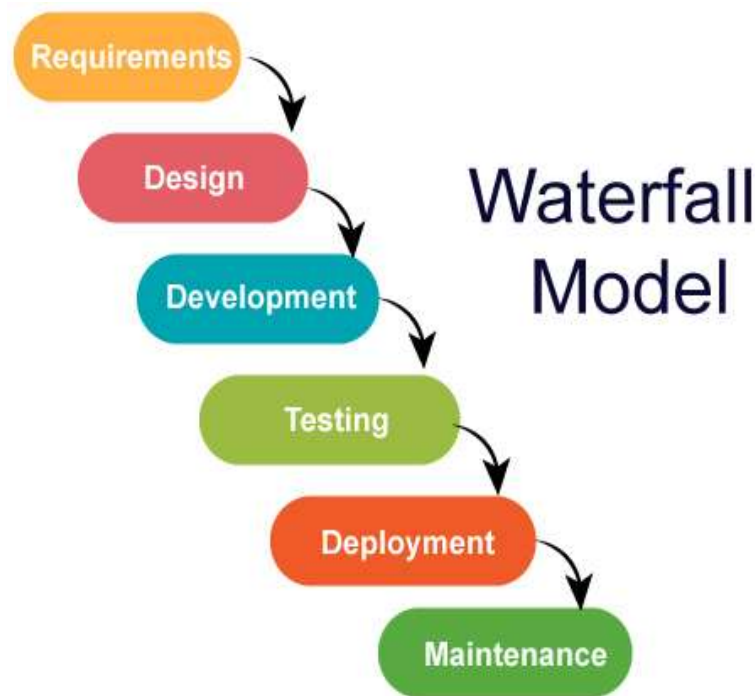
### 2.8.1 Waterfall Model

The Waterfall Model follows a linear, sequential flow, where progress moves steadily downwards (like a waterfall) through the phases of software development. Each stage in the development cycle begins only after the previous stage is completed. The waterfall approach does not accommodate going back to a previous stage to address changes in requirements. It is the oldest and most well-known method used for software development. The five-stage waterfall model, based on Winston W. Royce's requirements, divides development processes into the following stages:

- analysis
- design
- implementation
- testing
- operation



The waterfall model can be broken down into multiple phases.:



**Fig.No 2.8.1** water fall model

The **Waterfall Model** is a traditional and sequential approach to software development that divides the process into distinct phases: Requirements Analysis, System Design, Implementation (Coding), Testing, Deployment, and Maintenance. In this model, each phase must be completed before moving on to the next, creating a clear, linear flow of tasks and deliverables. One of the key advantages of the Waterfall Model is its structured nature, which makes it easy to manage and track progress, especially for projects with well-defined requirements. However, this rigid approach can be a disadvantage when dealing with complex or evolving projects, as it does not accommodate changes or feedback once a phase has been completed. As such, while the Waterfall Model is effective for small-scale, clearly scoped projects, it may not be suitable for dynamic projects that require frequent adjustments.

### **Advantages**

- Simple to clarify for the clients.
- Structures approach.
- Stages and exercises are distinct.
- Assists with arranging and timetable the task.

## **2.9 System Study**

The feasibility study evaluates the practicality of the project and enhances fundamental understanding through a comprehensive approach. During system assessment, the integrity evaluation of the proposed structure is crucial to ensure it does not burden the organization. Three key assessments conducted during feasibility appraisal are:

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

### **Economic Feasibility**

This study examines the financial impact of the system on the organization. It assesses the resources available for system development and justifies expenses. The design should be economically viable, leveraging freely available enhancements wherever possible, with consideration given to necessary purchases.

### **Technical Feasibility**

This study assesses the specific requirements of the system and ensures it does not overly strain existing technical resources. Excessive demands on technical resources can lead to burdens on users. The design should have modest technical requirements, minimizing unnecessary changes.

### **Social Feasibility**

This aspect examines the level of acceptance of the system by users. It involves establishing effective means to familiarize users with the system .

## 2.10 Technologies used java technology

Java technology serves as both a programming language and a platform, offering a versatile environment for software development. The Java Programming Language encompasses several key characteristics, including simplicity, architecture neutrality, object orientation, portability, distribution capability, high performance, interpretation, multithreading support, robustness, and dynamism.

### The Java Programming Language

The Java programming language is characterized by a myriad of buzzwords that underscore its versatility and effectiveness:

**Simple:** Java boasts a straightforward syntax and clear structure, making it easy to learn and use.

**Architecture neutral:** Java's architecture-neutral design allows it to run on any platform without modification, ensuring compatibility across diverse environments.

**Object-oriented:** Java follows the object-oriented programming paradigm, emphasizing the creation and manipulation of objects to achieve functionality.

**Portable:** Java programs can be compiled into platform-independent bytecode, enabling them to run on any device or operating system with a Java Virtual Machine (JVM) installed.

**Distributed:** Java supports distributed computing, allowing applications to be easily distributed across multiple systems and networks.

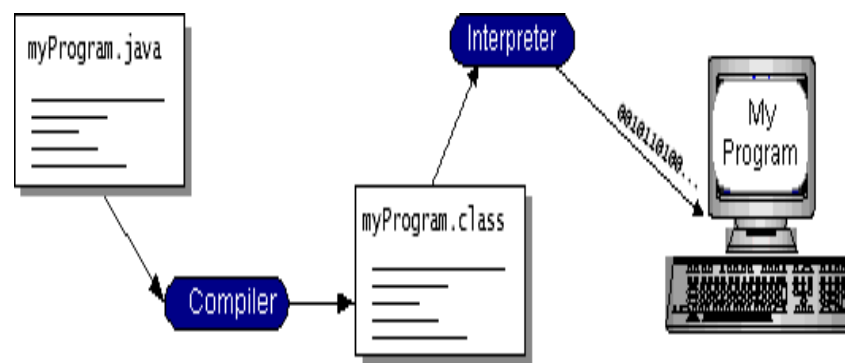
**High performance:** Despite its platform independence, Java offers high performance through efficient bytecode execution and optimization techniques.

**Interpreted:** Java programs are compiled into bytecode by the Java compiler and then interpreted by the JVM at runtime, facilitating dynamic execution.

**Multithreaded:** Java enables concurrent programming through its built-in support for multithreading, allowing multiple tasks to run simultaneously within a single program.

**Robust:** Java prioritizes reliability and robustness by incorporating features such as strong memory management, exception handling, and type safety.

**Dynamic:** Java's dynamic nature allows for runtime adaptation and modification of program behavior, enhancing flexibility and responsiveness.



**Fig No.2.10.1** Working of Java Program

Java bytecode serves as the machine code instructions for the Java Virtual Machine (Java VM), enabling the "write once, run anywhere" paradigm. Every Java interpreter, whether it's a development tool or a web browser capable of running applets, functions as an implementation of the Java VM. By compiling a program into bytecode, it becomes platform-independent, allowing it to run on any system with a Java VM installed. This flexibility means that a Java program written on one platform, such as Windows 2000, can seamlessly execute on other platforms like a Solaris workstation or an iMac, as long as they support the Java VM.

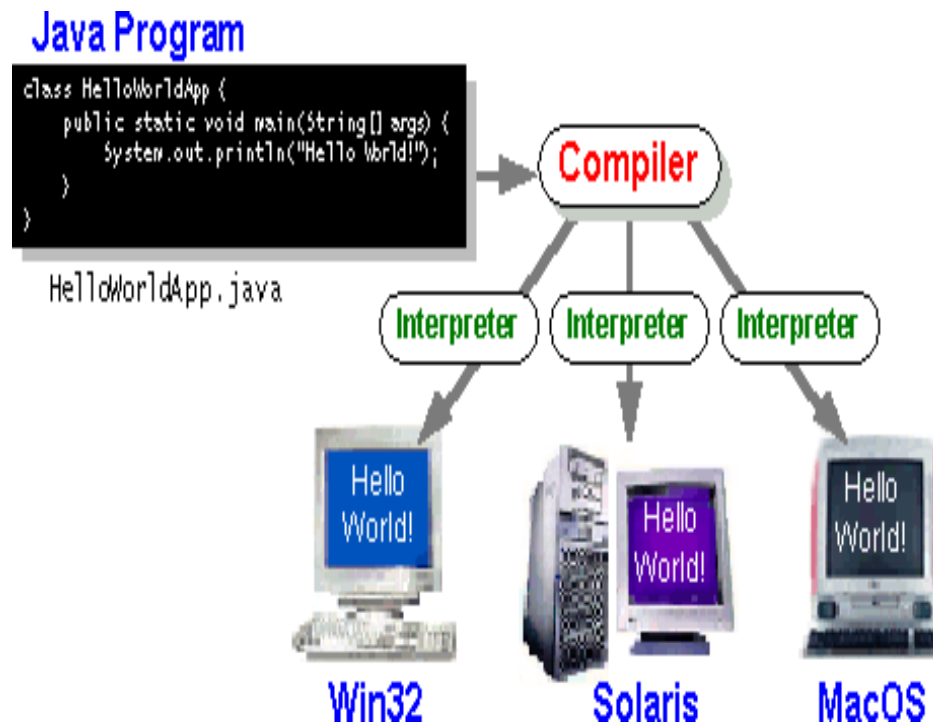


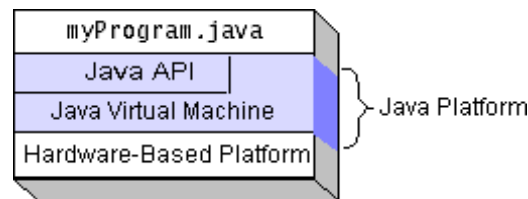
Fig No.2.10.2 Implementation of Java Virtual Machine

### The Java Platform

A platform refers to the hardware or software environment where a program operates. Common platforms include Windows 2000, Linux, Solaris, and MacOS, which typically combine both operating system and hardware components. The Java platform, however, distinguishes itself by being a software-only platform that operates atop other hardware-based platforms. Comprising two main components, the Java platform consists of:

The Java Virtual Machine (Java VM): This serves as the foundation for the Java platform and is adapted to various hardware-based platforms. The Java Application Programming Interface (Java API): This encompasses a vast collection of pre-built software components offering various functionalities, such as graphical user interface (GUI) widgets. The Java API organizes these components into libraries of related classes and interfaces, known as packages.

In the subsequent section, "What Can Java Technology Do?" highlights the functionalities provided by some of the packages within the Java API. The illustration below demonstrates a program operating within the Java platform, with the Java API and the virtual machine shielding the program from direct interaction with the hardware.



**Fig.2.10.3 Program Running on the Java Platform**

Native code refers to code that, once compiled, runs directly on a specific hardware platform. In contrast, the Java platform operates as a platform-independent environment, where programs can run across different hardware architectures. While the Java platform may exhibit slightly lower performance compared to native code, optimizations such as smart compilers, well-tuned interpreters, and just-in-time byte code compilers help bridge this performance gap while preserving portability.

### **What Can Java Technology Do?**

Java technology offers a versatile set of programming options. Applets and applications are among the most common types of programs written in the Java programming language. Applets, familiar to web users, adhere to specific conventions allowing them to run within Java-enabled browsers. However, Java's capabilities extend beyond web applets. The Java programming language serves as a robust software platform suitable for various types of applications.

Applications in Java can range from standalone programs to servers that support clients on a network. Servers, such as web servers, proxy servers, mail servers, and print servers, run directly on the Java platform, serving clients' needs. Servlets, another specialized type of program, function similarly to applets but operate on the server side within Java Web servers, enhancing interactive web applications.

The Java API supports a wide array of programs through packages of software components, providing diverse functionalities. A full implementation of the Java platform offers essential features like objects, strings, threads, numbers, input and output mechanisms, data structures, system properties, and date and time functionalities.

Furthermore, the Java API includes support for applets, networking (including URLs, TCP, UDP sockets, and IP addresses), internationalization for localized programs, comprehensive security measures (both low-level and high-level), software components known as JavaBeans, object serialization for lightweight persistence and communication via Remote Method Invocation (RMI), and Java Database Connectivity (JDBC) for uniform access to relational databases.

Moreover, the Java platform provides APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The Java 2 SDK encompasses these functionalities, facilitating the development of diverse applications and enhancing the overall capabilities of the Java platform.

### **3. SYSTEM DESIGN**

#### **3.1 E-R Diagram**

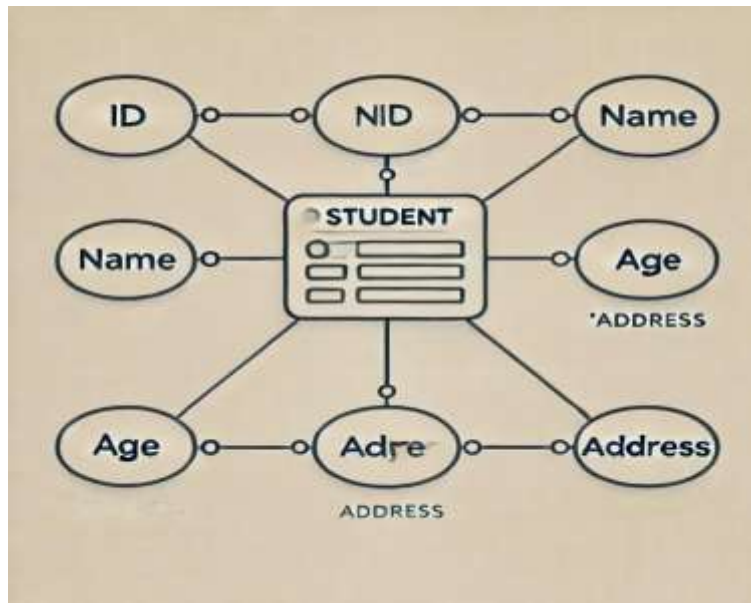
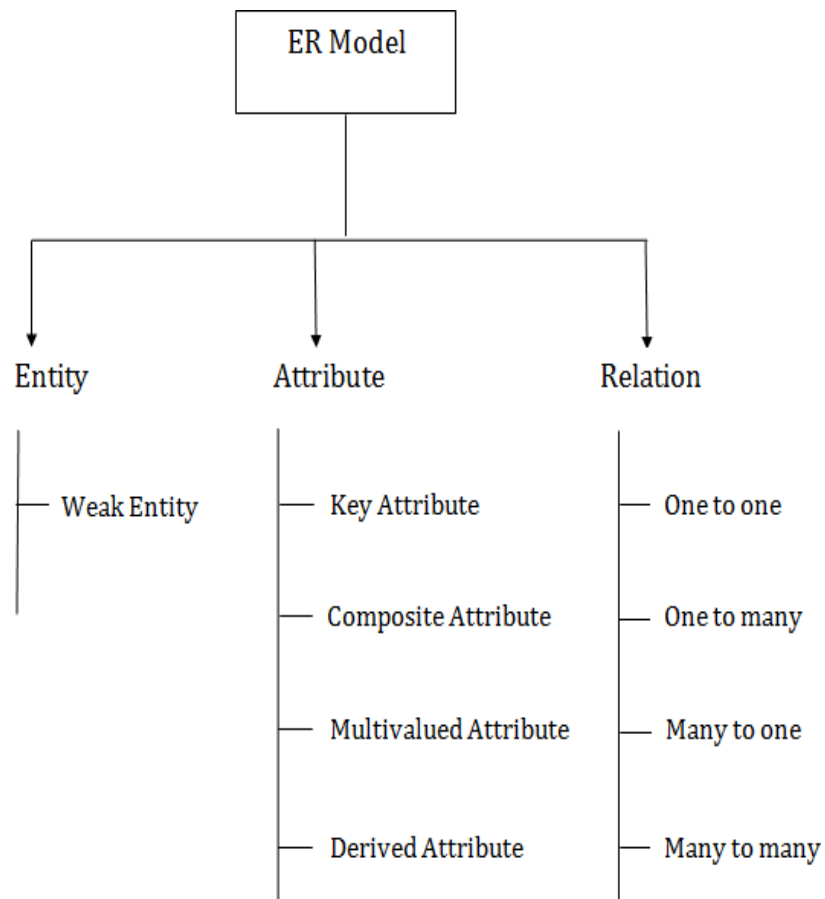
An Entity-Relationship (ER) model illustrates the structure of a database using a visual representation known as an Entity-Relationship Diagram (ER Diagram). This model serves as a blueprint for designing the database schema and capturing the relationships between different entities and attributes.

The ER model provides a systematic approach to organizing and conceptualizing the data within a database system. It represents entities as well as the relationships between them, helping to clarify how data elements are connected and organized.

#### **ER model**

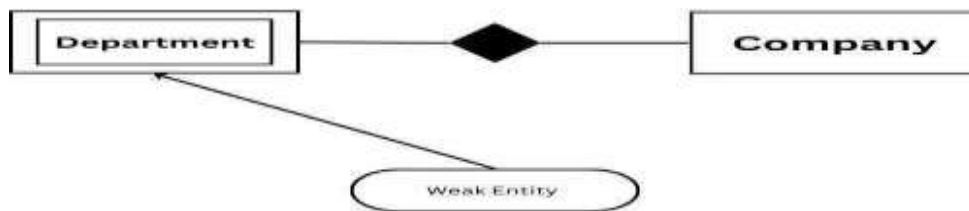
1. The Emergency Room model corresponds to an Entity-Relationship model, serving as a high-level representation of data structures. It is utilized to illustrate the data components and relationships within a defined system.
2. It establishes a structured framework for the database. Moreover, it provides a straightforward and easily understandable perspective on the data.
3. In Entity-Relationship modeling, the organizational database structure is depicted through a design known as an Entity-Relationship diagram.
4. For instance, consider designing a school database. An educational record could be represented as an entity with attributes such as name, ID, age, etc. Similarly, the address could be another entity with attributes like city, street name, zip code, etc., and there would be a relationship between them.



**Fig no 3.1.1** ER-Model

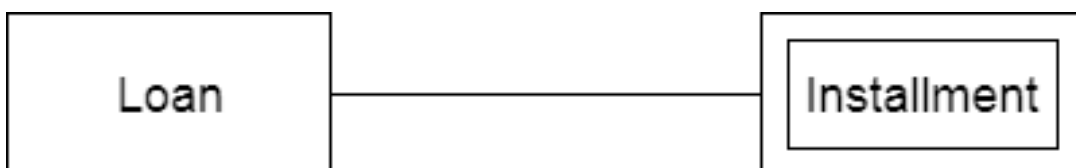
## Component of ER Diagram Entity

A substance may be anything, class, individual or spot. In the ER frame, a substance can be tended to as square shapes, Think about a relations.



### 1. Weak Entity

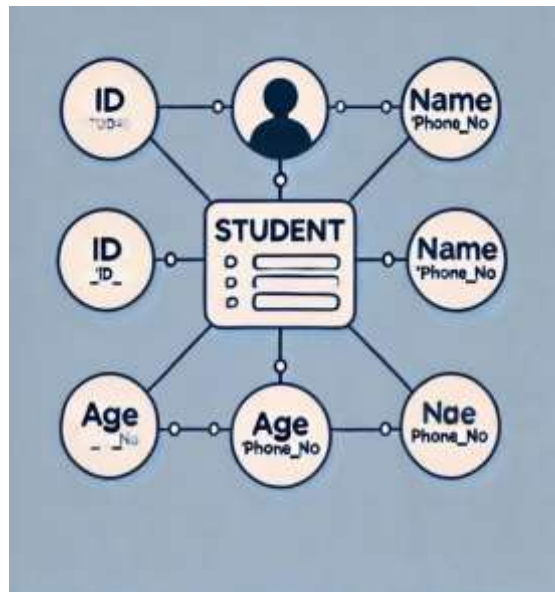
**A substance that depends upon another component called a frail substance.** The frail element contains no critical trait of its own. The feeble substance is addressed by a twofold square shape.



### 2. Characteristic

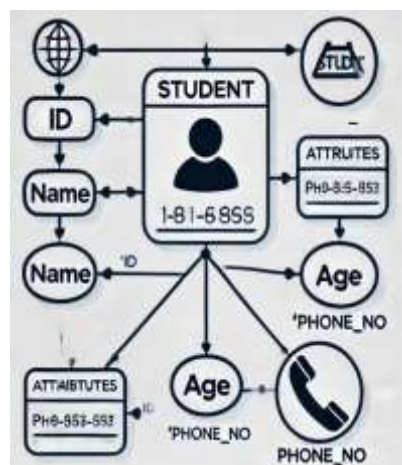
The quality is utilized to depict the property of a section. Obscure is utilized to address a quality.

For example, id, age, contact number, name, etc can be attributes of a student.



#### a. Key Attribute

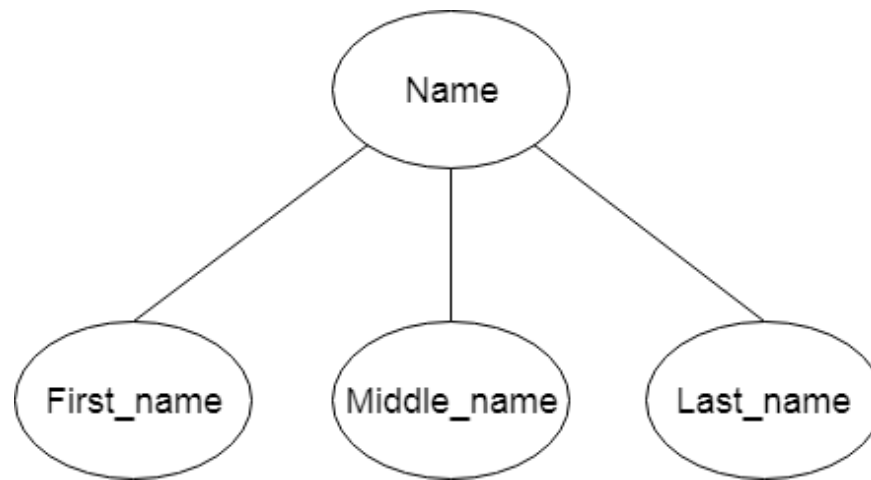
The key quality is used to address the essential ascribes of a substance. It tends to a fundamental key. The key property is tended to by a circle with the text underlined.



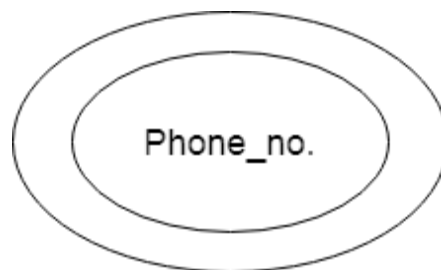
**Fig no: 3.1.3 Key Attribute**

#### b. Composite Attribute

A property that made from various attributes is known as a composite quality. The composite trademark is tended to by an oval, and those circles are related with a circle.

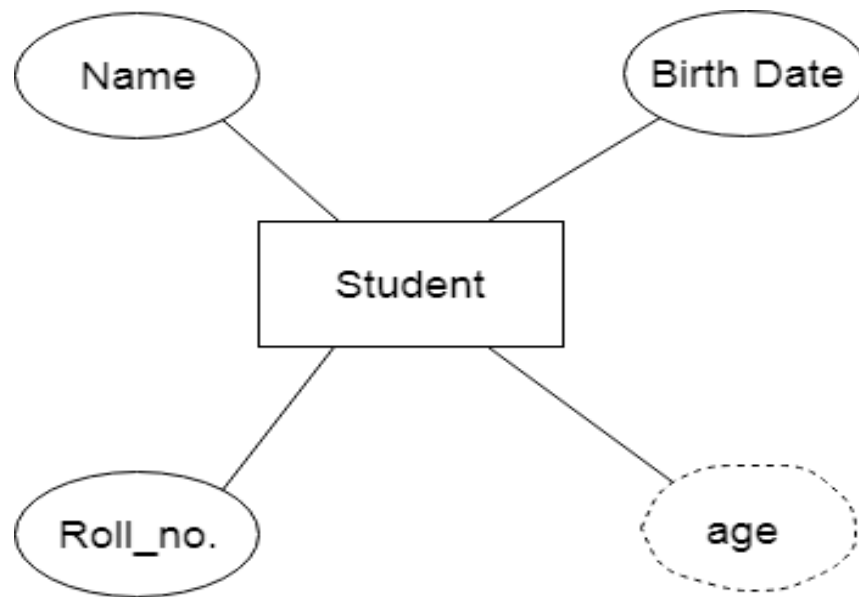
**c. Multivalued Attribute**

A quality can have more than one worth. These qualities are known as a multivalued property. The twofold oval is used to address multivalued property. For example, a student can have more than one phone number.

**d. Determined Attribute**

A property that can be gotten from another quality is known as a decided attribute. It will in general be tended to by a ran circle.

For example, a singular's age changes long term and can be gotten from one more quality like Date of birth.



### 3. Relationship

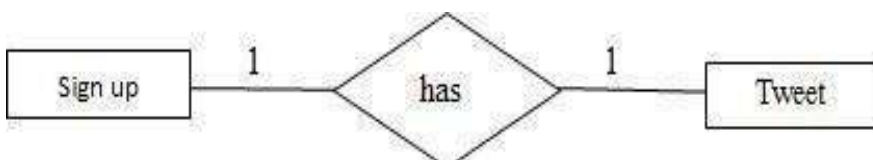
A relationship is used to depict the connection between substances. Important stone or rhombus is utilized to address the relationship.



**Sorts of relationship are as per the following:**

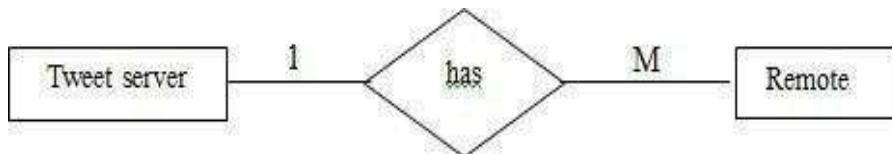
#### a. One-to-One relationship

At the point when just a single instance of a component is connected with the relationship, then it is known as facilitated relationship. For instance, A female can wed to one male, and a male can wed to one female.



**b. One-to-many relationship**

Exactly when more than one event of the component on the left, and simply a solitary event of a substance on the right associates with the relationship then it is known as a many-to-one relationship. For example, Student enrolls for only a solitary course, but a course can have various students.

**c. Many-to-one relationship**

Exactly when more than one event of the component on the left, and simply a solitary event of a substance on the right associates with the relationship then it is known as a many-to-one relationship.



*Beginnerbook.com*

**d. Many-to-many relationship**

At the point when more than one event of the substance on the left, and more than one event of a component on the right associates with the relationship then it is known as a many-to-various connections.



*Beginnerbook.com*

## **E-R DIAGRAM**

E-R Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships. ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

The diagrams depict the interaction and data flow between three key entities in the system: Company, Authority, and Admin. Each of these entities has distinct modules and interacts with a centralized database for data storage and retrieval. The Company entity has modules for Register & Login, managing products, viewing shipments and transactions, and submitting damage claims and announcements, allowing companies to efficiently manage their operational needs. The Authority entity, responsible for monitoring and verifying product conditions, has modules for Login, making announcements, updating shipment status, and verifying products. The Admin entity oversees the entire system, managing company details, authority details, and announcements, and ensuring the system runs smoothly. The data flow between these modules and the database ensures a structured and coordinated system, enabling efficient management of shipments, product conditions, and administrative tasks.

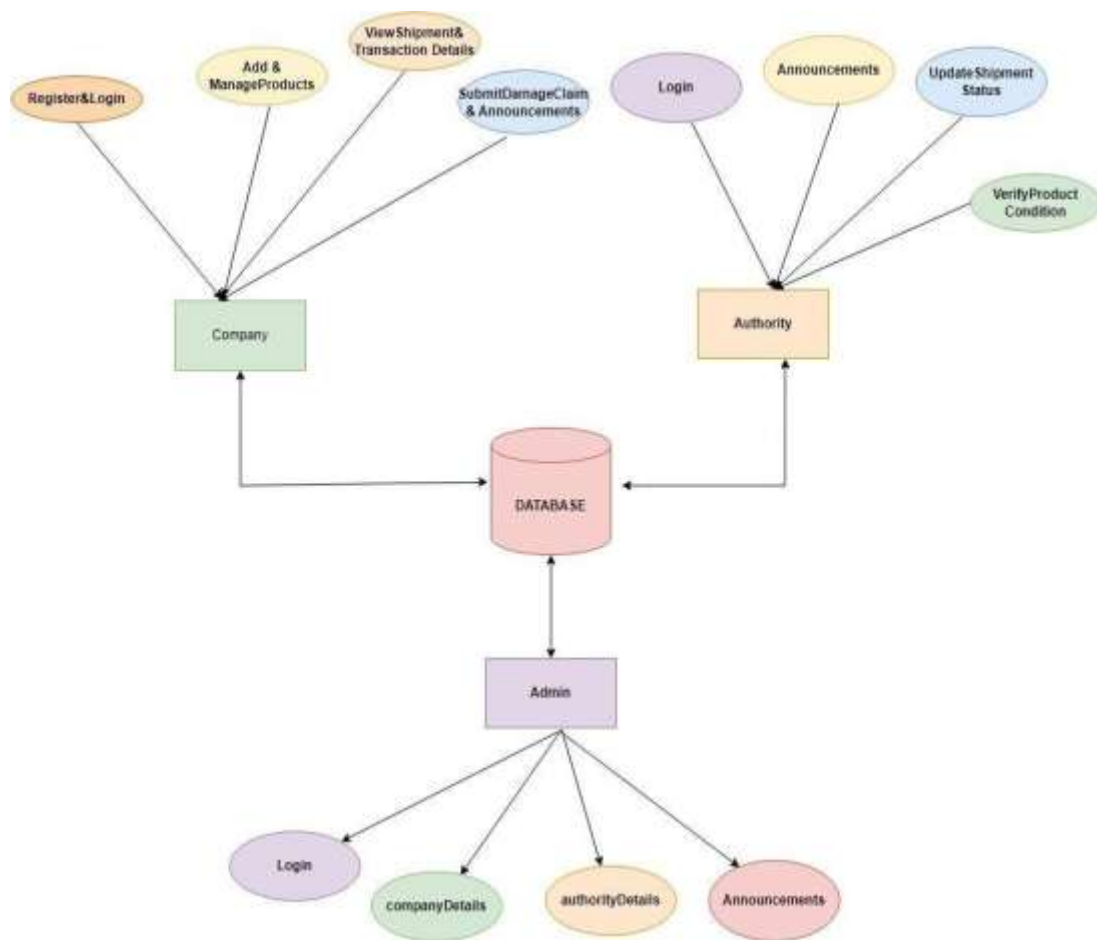


Fig No:3.1.4 ER Diagram



### 3.2 Data Dictionary

A Data Dictionary compiles names, definitions, and attributes concerning data elements utilized or stored within a database, information system, or part of a research project. It delineates the meanings and functions of data elements within the context of a project and offers guidance on understanding, recognizing meanings, and description. Additionally, a Data Dictionary offers metadata about data elements, aiding in defining the scope and attributes of data elements, as well as the guidelines for their usage and application.

Name	Type	Collation	Attributes	Null	Default
id	Int(11)			No	None
User name	Varchar (30)	Latin_swedish_ci		No	None
email	Varchar (30)	Latin_swedish_ci		No	
pswd	Varchar (30)	Latin_swedish_ci		No	
phone no	Varchar (30)	Latin_swedish_ci		No	
country	Varchar (30)	Latin_swedish_ci		No	
state	Varchar (30)	Latin_swedish_ci		No	

**Table No.3.2.1** Data Dictionary

Id	User name	email	pswd	phno	country	state	city
1	vijay	vkappati@gmail.com	*****	6281407028	India	AP	kurnool
2	vinod	vinod@gmail.com	*****	8790557392	India	AP	kurnool
3	Ajay	ajay@gmail.com	*****	7780490892	India	AP	mdk
4	vishnu	Vishnukvr@gmail.com	*****	9849466191	India	AP	rzp

**Table No 3.2.2** Data Table

Id	uname	ureview	sanalysis	tname	suggestion
1	Kumar	The traffic for ambulance bad	False positive	Traffic	Better to avoid traffic
2	kumar	Exam System is good fees students	False Negative	Exam	To know about exam
3	Vinod	We can find more details in too and can view all	Malicious Bots	Sexual_assults	To know about sexual assaults
4	Manjuna th	This is excellent	False Negative	Carona_vaccine	Consume all people
5	Chinni	Invention if vaccine comes to	False Negative	IPL	Provide more information

**Table No 3.2.3** Received User Data Analysis

### 3.3 Normalization

Normalization is the primary method for optimizing data in a database to fulfill two essential criteria:

Data dependencies are logical, ensuring that all related data items are stored together. Normalization is crucial for various reasons, primarily because it enables databases to occupy minimal disk space, resulting in enhanced performance. Normalization is also referred to as data standardization.

The three primary types of normalization are outlined below. Note: "NF" stands for "normal form." Normalization is a key database design technique that organizes data to reduce redundancy and improve data integrity. By applying normalization, data is structured into tables and relationships to ensure efficient storage and retrieval. The process involves dividing large, complex tables into smaller, more manageable tables and establishing clear relationships through primary and foreign keys. Common normal forms, such as 1NF, 2NF, 3NF, and BCNF, progressively remove anomalies like update, insert, and delete issues. This not only optimizes database performance but also enhances data consistency and accuracy, making it easier to maintain and scale.

Normalization also helps to eliminate data anomalies by ensuring that each piece of information is stored only once, minimizing the risk of inconsistencies. For example, it ensures that updates to a customer's address are made in only one place, rather than across multiple tables. Furthermore, normalization promotes data integrity by enforcing referential integrity constraints, which maintain logical relationships between tables and protect data from becoming orphaned or invalid. In addition to its technical benefits, normalization can enhance query performance by reducing table size and minimizing the amount of data that needs to be scanned during database operations. However, it's important to balance normalization with practical considerations, as over-normalization can lead to excessive joins and may impact performance in certain scenarios. Ultimately, normalization provides a structured and efficient foundation for building robust database applications.

### ➤ First typical structure (1NF)

Tables in 1NF should comply with certain standards:

- Every cell should contain just a solitary (nuclear) esteem.
- Each part in the table ought to be astoundingly named.
- All characteristics in a part ought to connect with a comparative region.

UserID	Username	Password
015	John	*****
016	Princess	*****
027	Tom	*****
028	Claire	*****
029	Robert	*****

**Table No.3.3.1 1NF**

structure, each cell contains a single, indivisible (atomic) value, ensuring that no cell has multiple values or nested records. Each column is uniquely named, representing a distinct attribute of the entity being recorded. In this case, the table records user information, including UserID, Username, and Password. The entries in each column are related to the same subject—user accounts—and are free of repeating groups or multivalued attributes.

This adherence to 1NF lays the foundation for more advanced normal forms and eliminates issues such as data redundancy and inconsistencies in the dataset. The table ensures that updates, deletions, and insertions can be performed efficiently, maintaining data integrity at this basic level of normalization.

### ➤ Second typical structure (2NF)

Tables in 2NF ought to be in 1NF and not have any most of the way dependence (e.g., each non-prime quality ought to be dependent upon the table's fundamental key).

User Id	Received Data through IOT	pswd	Login
1	11	*****	Sign_up
2	12	*****	Sign_up
3	13	*****	Sign_up
4	14	*****	Sign_up
5	15	*****	Sign_up

**Table No.3.3.2 2NF**

The illustrates a typical example of a table in Second Normal Form (2NF). To achieve 2NF, the table must first satisfy 1NF and then eliminate any partial dependencies—meaning that all non-prime attributes (attributes that are not part of any candidate key) must depend on the entire primary key. In this example, the table stores information about users, including User Id, Received Data through IOT, pswd, and Login. Here, the User Id serves as the primary key, and each non-key attribute (Received Data through IOT, pswd, and Login) is fully functionally dependent on it.

This structure ensures that no attribute depends only on a part of the primary key (which would only occur in composite keys), thereby reducing redundancy and enhancing data consistency. Achieving 2NF is a crucial step in database normalization, as it eliminates partial dependencies and ensures that each attribute is directly related to the entity represented by the table, laying the groundwork for more advanced normalization forms like 3NF.

### ➤ Third ordinary structure (3NF)

Tables in 3NF ought to be in 2NF and have no transitive reasonable circumstances on the fundamental key. The going with two NFs furthermore exists anyway are only here

and there used:

- **USERDETAILS**

ID	NAME	EMAIL	STATE	CITY	COUNTRY
11	Vijay	vijay@gmail.com	AP	RZP	INDIA
12	Vinod	vinod@gmail.com	AP	RZP	INDIA
13	Ramu	Ramu@gmail.com	AP	RZP	INDIA
14	Vishnu	vishnu@gmail.com	AP	RZP	INDIA

**Table No 3.3.3** User Details

- **USER DETAILS**

USER ID	PASSSSWORD	LOGIN
server	*****	Sign_up
Vijay	*****	Sign_up

**Table No3.3.4** User details

The tables Table no. 3.5.3 User Details and Table no. 3.5.4 User details illustrate the application of Third Normal Form (3NF) in a database. For a table to be in 3NF, it must already be in 2NF and have no transitive dependencies—meaning non-prime attributes cannot depend on other non-prime attributes. In Table no. 3.5.3 User Details, attributes such as ID, NAME, and EMAIL depend directly on the primary key ID, while location-specific data like STATE, CITY, and COUNTRY are also directly related to ID, avoiding transitive dependencies. Similarly, Table no. 3.5.4 User details maintains USER ID, PASSSSWORD, and LOGIN with direct dependence on the primary key USER ID. This separation ensures that each table focuses on a specific set of attributes directly tied to its primary key, enhancing data integrity and reducing redundancy. By achieving 3NF, these tables support efficient data operations and maintain a clear structure that aligns with the principles.

## ➤ **Boyce-Codd Normal Form (BCNF)**

Normalization is a critical process in database management aimed at organizing tables to minimize anomalies and ensure data integrity. It follows a series of stages known as normal forms. These normal forms help structure tables efficiently and reduce redundancy and inconsistency in data.

**Unnormalized Form (UNF):** The initial state of a table where data is not organized according to any specific rules.

**First Normal Form (1NF):** In 1NF, each column contains atomic values, and there are no repeating groups or arrays within a row.

**Second Normal Form (2NF):** 2NF requires that every non-key attribute be fully functionally dependent on the primary key.

**Elementary Key Normal Form (EKNF):** EKNF is a further refinement of 3NF, emphasizing the use of elementary keys.

**Boyce-Codd Normal Form (BCNF):** BCNF addresses anomalies that may arise when multiple candidate keys exist. It requires that for every non-trivial functional dependency ( $X \rightarrow Y$ ),  $X$  must be a superkey.

**Fourth Normal Form (4NF):** To achieve 4NF, a table must be in BCNF and should not have multi-valued dependencies.

**Essential Tuple Normal Form (ETNF):** ETNF is a condition where each attribute in a tuple is essential to the understanding of the tuple itself.

Normal Form	Description
1NF	An alliance is in 1NF enduring it contains an atomic worth.
2NF	An association will be in 2NF expecting it is in 1NF and all non-key credits are totally down to earth ward on the fundamental key.
3NF	An alliance will be in 3NF enduring it is in 2NF and no change dependence exists.
BCNF	A more grounded importance of 3NF is known as Boyce Codd's common design.
<a href="#">4NF</a>	An association will be in 4NF expecting it is in Boyce Codd's commonplace construction and has no multi-regarded dependence.
<a href="#">5NF</a>	An association is in 5NF. In case it is in 4NF and contains no join dependence, joining should be lossless.

**Table No3.3.5** Normal Forms

### Benefits of Normalization

Reduction of data redundancy: Normalization helps eliminate redundant data by organizing it efficiently across tables. Improved overall database organization: By structuring data according to normalization rules, databases become more organized and easier to manage. Data consistency within the database: Normalization ensures that data remains consistent across tables, reducing the risk of inconsistencies. More flexible database design: Normalization allows for more flexibility in database design, making it easier to accommodate changes and updates. Upholds the principle of data integrity: Normalization promotes data integrity by minimizing anomalies and ensuring accurate representation of data relationships.



- **Disadvantages of Normalization**

Careless decomposition: If normalization is done without a clear understanding of user requirements, it can lead to excessive decomposition and unnecessary complexity in the database design

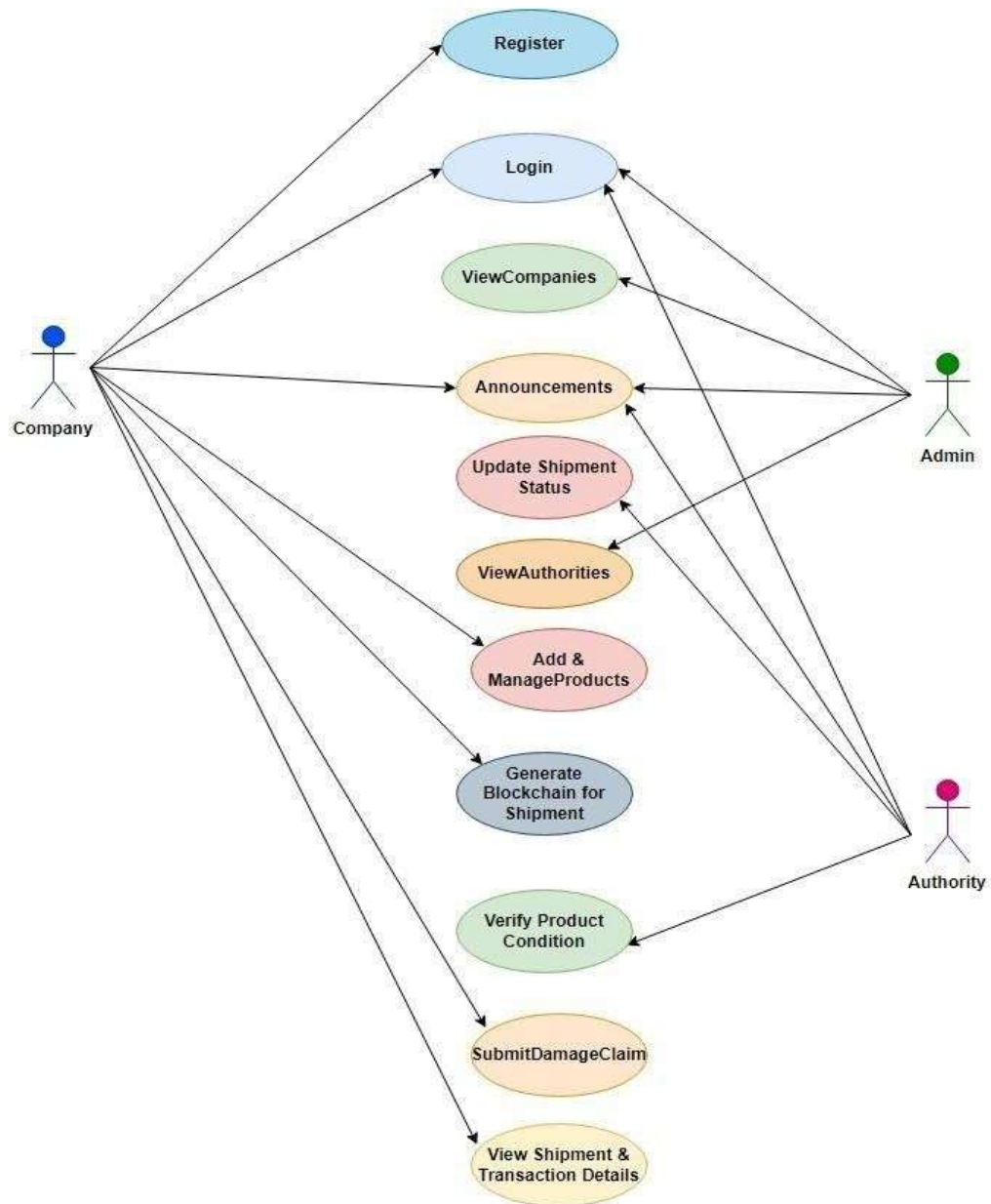
Decreased performance: As tables are normalized to higher normal forms such as 4NF or 5NF, it may lead to decreased performance due to increased join operations and complexity in querying the database.

1. Complex Queries: Joins across many tables make queries more complex and slower.
2. Reduced Performance: Increased joins can lead to slower read operations.
3. Difficult Reporting: Aggregated or summary reports become harder to generate.
4. Increased Development Time: More effort is needed for design and maintenance.
5. Overhead in Joins: Too many foreign keys and relationships add processing overhead.
6. Less Flexibility: Adapting to changing business needs becomes more complicated.

### 3.4 UML DIAGRAMS

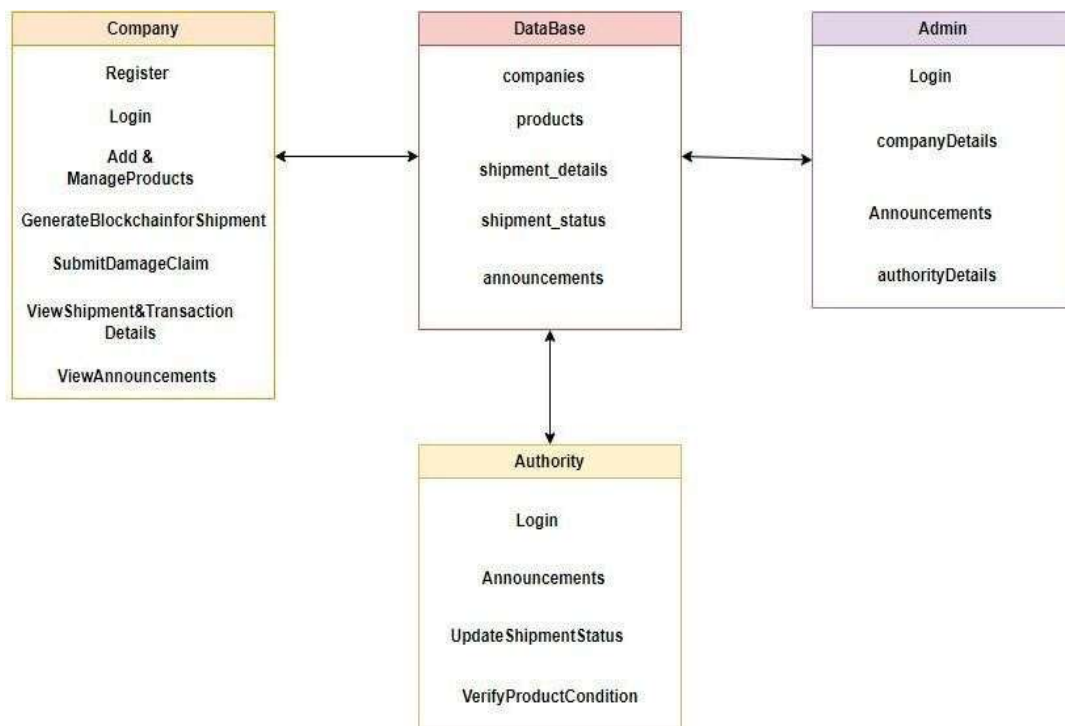
#### Usecase Diagram

The **Use Case Diagram** captures the system's functionality and dynamic behavior, illustrating the interactions between different actors and the system. The actors in this case are the **Admin**, **Company**, and **Authority** modules. Each actor has specific roles and responsibilities: the Admin manages users, monitors system operations, and oversees company and authority details, while the Company handles registration, product management, shipment, and claim submissions, utilizing blockchain for secure transactions. The Authority validates the product conditions and verifies claims based on the received products. Each actor has separate **registration** and **login** processes to access their respective functions. Furthermore, the Company generates blockchain entries for shipment details and updates them with relevant information, while the Admin ensures smooth operations, and the Authority confirms shipment conditions, guaranteeing the integrity of transactions within the system.

**Fig No:3.4.1** Usecase diagram

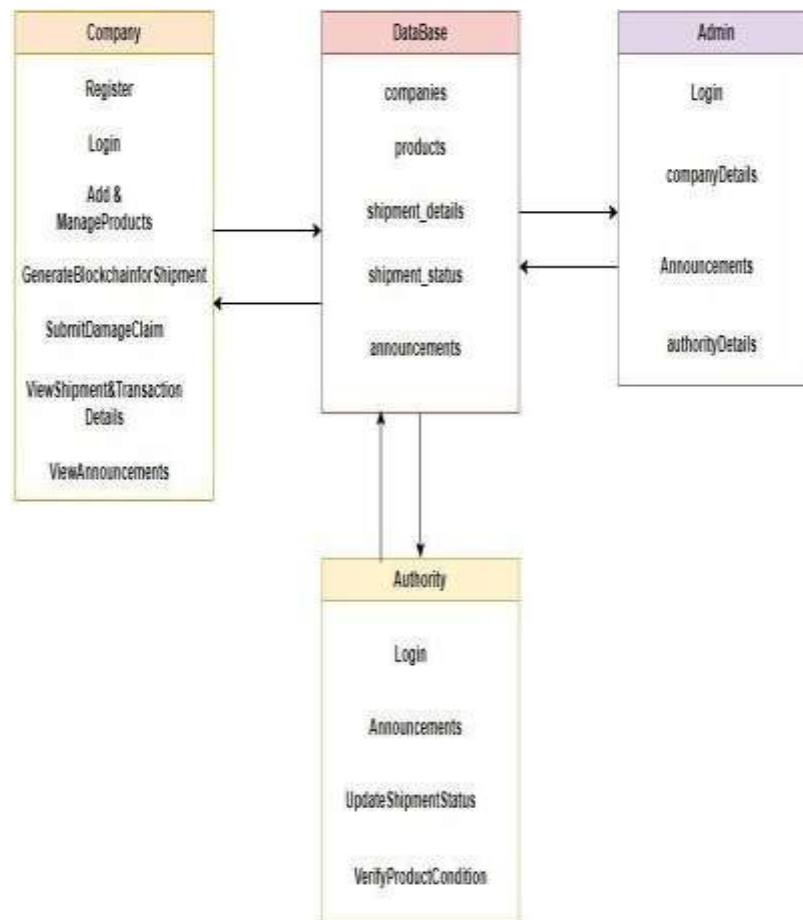
## Class Diagram

Class diagrams are a key component in object-oriented modeling, used to represent the different objects in a system, their attributes, operations, and relationships. In the context of our project, the primary objects are the **Admin**, **Company**, and **Authority**. These objects interact with each other in various ways, each with their own responsibilities. The **Admin** manages system users, monitors the blockchain network, and oversees the activities of both companies and authorities. The **Company** object is responsible for registering, managing products, initiating shipments, generating blockchain blocks for transactions, and submitting claims for damaged goods. The **Authority** object validates product conditions, verifies whether goods are damaged, updates shipment statuses, and confirms claims. Each object has specific attributes and operations such as handling transactions, generating blockchain blocks for shipment details, verifying claims, and ensuring secure communication between actors. The relationships among these objects are vital for ensuring a seamless and secure blockchain-based system for cross-company transactions and claims.



**Fig No.3.4.2** Class diagram

## Object Diagram



**Fig No:3.4.3** Object diagram

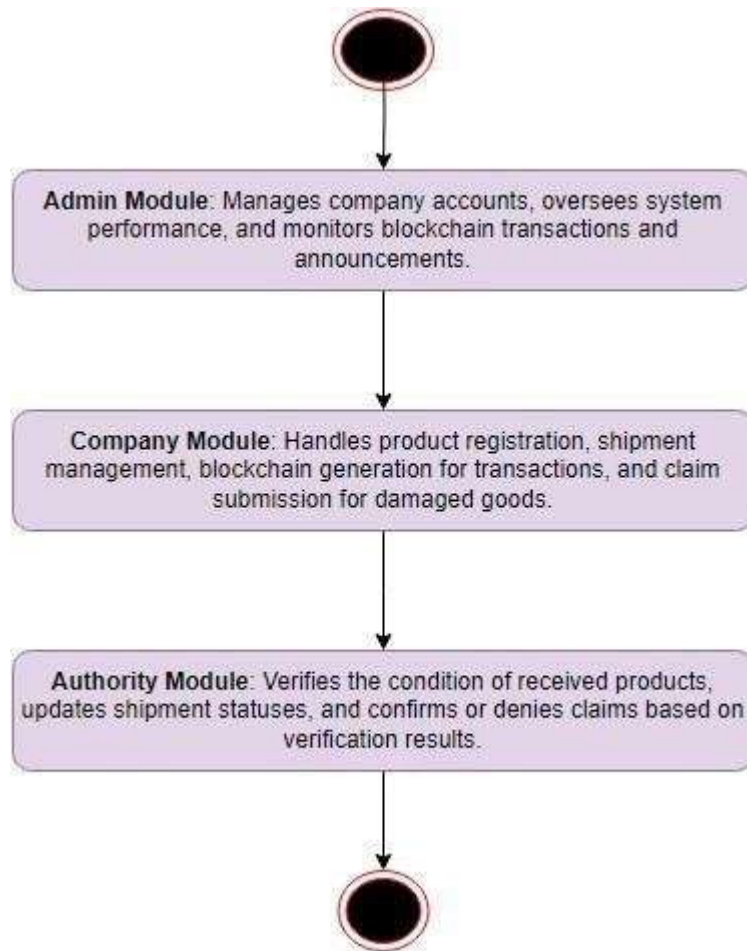
An object diagram shows this relation between the instantiated classes and the defined class, and the relation between these objects in the system. They are be useful to explain smaller portions of your system, when your system class diagram is very complex, and also sometimes modeling recursive relationship in diagram. The best way to illustrate what an object diagram look like is to show the object diagram derived from the corresponding class diagram.

## State Diagram

A state diagram, also known as a state machine diagram or state chart diagram, is an illustration of the states an object can attain as well as the transitions between those states in the Unified Modeling Language. Then, all of the possible existing states are placed in relation to the beginning and the end.

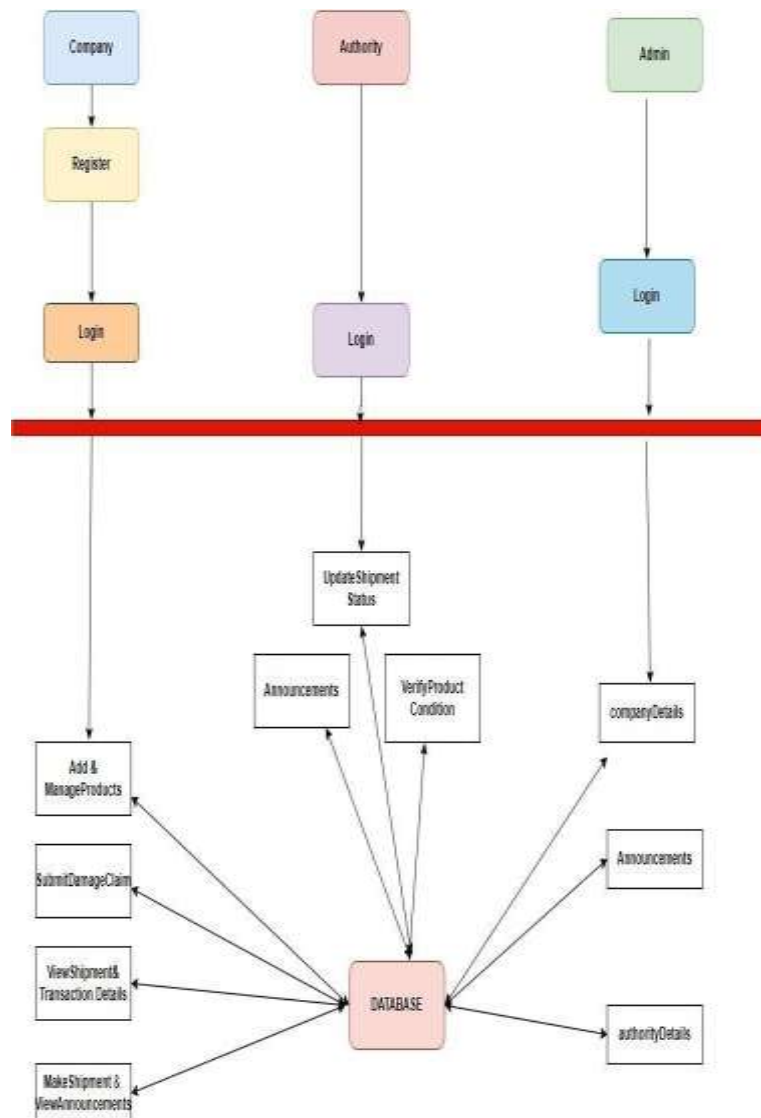
A UML State Diagram (also known as a state machine diagram) is used to model the dynamic behavior of a system by showing how an object changes its state in response to events. It focuses on the life cycle of a single object, illustrating the sequence of states it passes through during its lifetime, triggered by events and transitions. The diagram typically includes states (represented by rounded rectangles), transitions (arrows between states), events, and sometimes actions or conditions. State diagrams are especially useful for modeling reactive systems, such as user interfaces, workflow systems, or communication protocols, where the behavior depends on past interactions. It helps developers understand how an object behaves in different scenarios, making it a valuable tool for both analysis and implementation phases of software development.

A State Diagram uniquely represents the states and transitions of an object or system in response to events over time. What makes it distinct is its ability to model event-driven behavior, focusing not just on structure but on how an entity reacts and evolves. Unlike other UML diagrams, the state diagram emphasizes control logic by showing how an object moves between states like "Idle," "Processing," or "Completed" based on specific triggering events. It can also include entry/exit actions, guard conditions, and internal transitions, offering detailed insight into how a system maintains stability or shifts behavior dynamically. This makes it especially valuable in domains like embedded systems, real-time systems, and workflow automation, where state-dependent logic is critical to system functionality.

**Fig No.3.4.4** State diagram

## Activity diagram

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination.

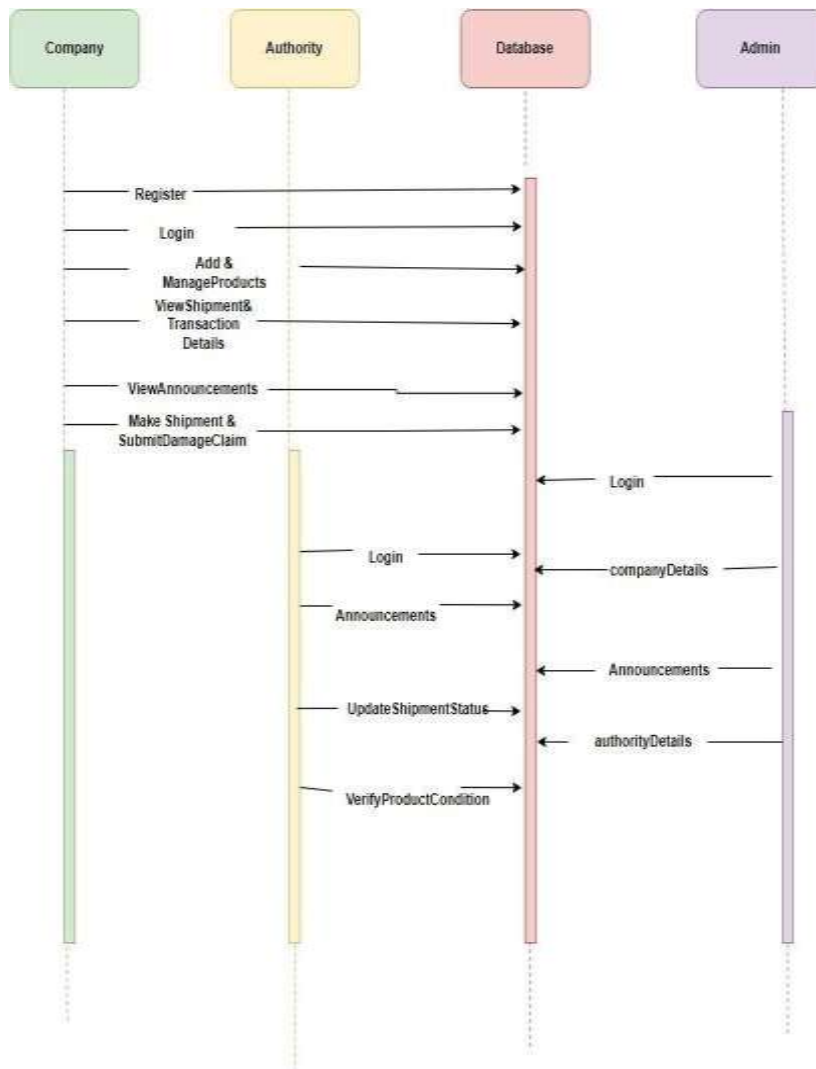


**Fig No:3.4.5** Activity diagram



## Sequence Diagram

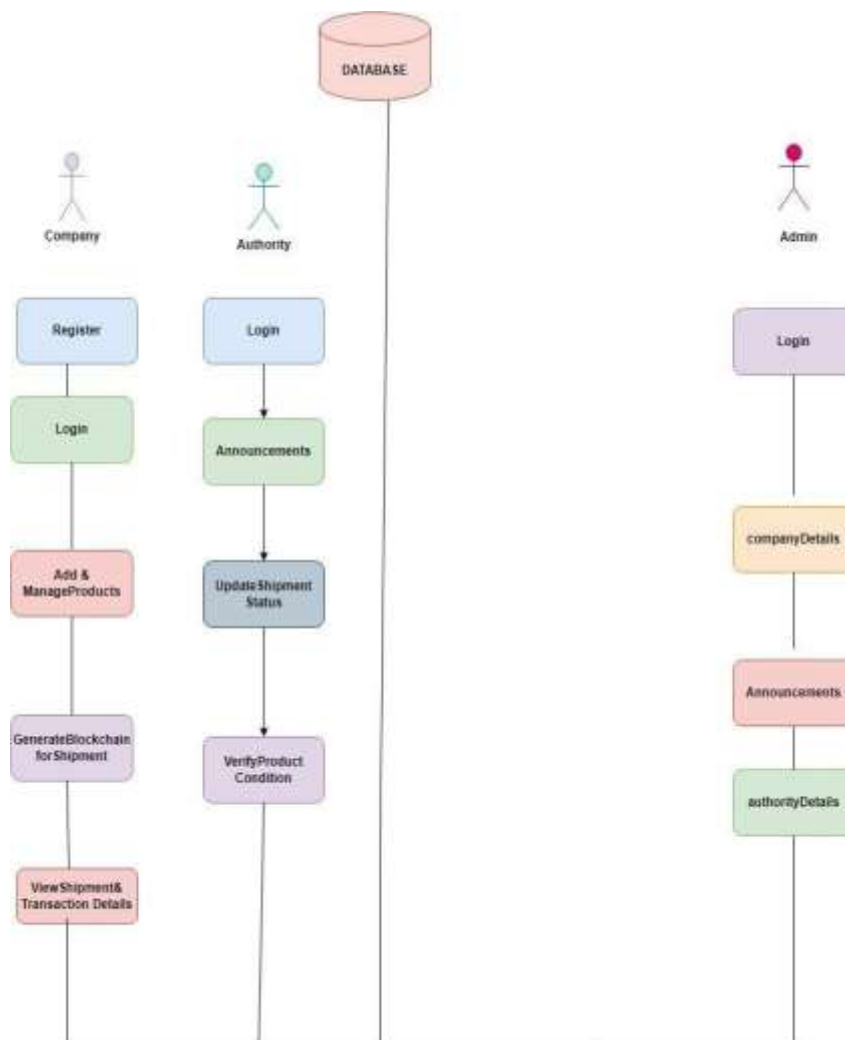
A sequence diagram is a type of interaction diagram because it describes how and in what order a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.



**Fig No:3.4.6** Sequence diagram

## Collabration Diagram

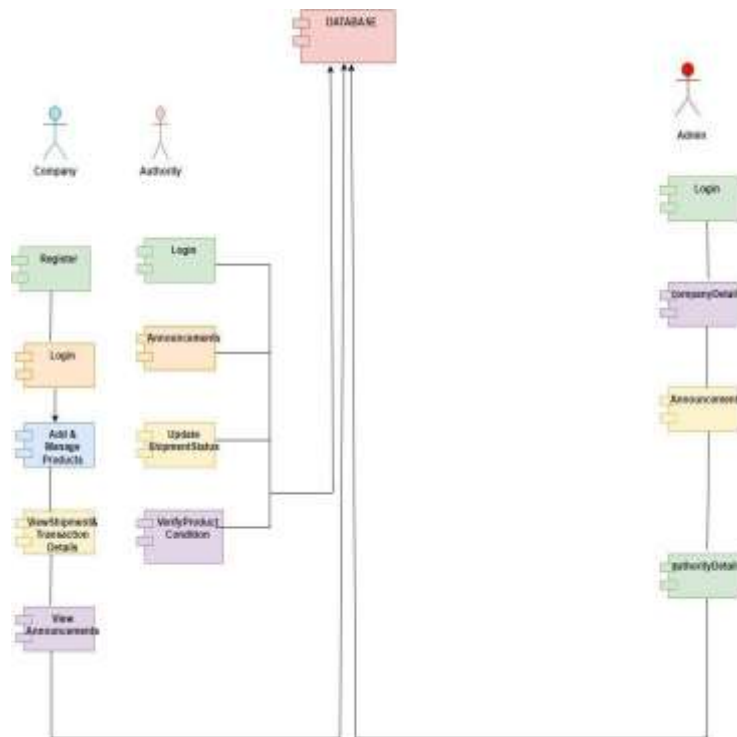
A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.



**Fig No.3.4.7** Collabration diagram

## Component Diagram

The Component diagrams are special type of UML diagrams used for different purposes. These diagrams show the physical components of a system. To clarify it, we can say that component diagrams describe the organization of the components in a system.



**Fig No.3.4.8** Component diagram

## Dataflow Diagram

Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation. Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

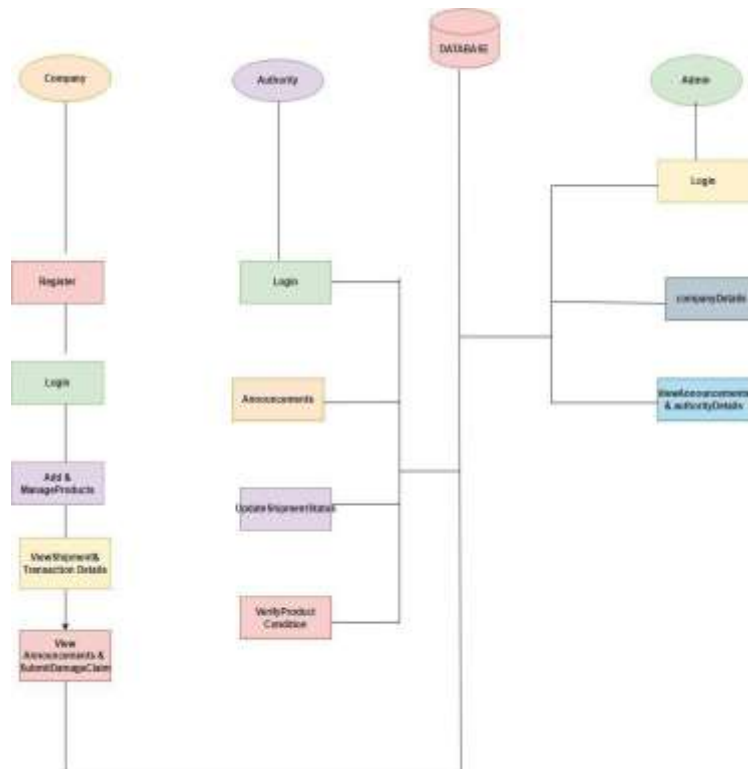


Fig No.3.4.9 Dataflow diagram

## Deployment Diagram

The deployment diagram illustrates the physical hardware on which the software will be deployed. It represents the static deployment view of a system, depicting the nodes and their relationships. It details how software is distributed across the hardware. The deployment diagram maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. As it involves multiple nodes, the relationships are depicted using communication pathways.



**Fig No.3.4.10** Deployment Diagram

## **4. Testing**

### **4.1 Introduction**

In this chapter, we develop achievability protocols and outer bounds for the secure network coding setting, where the edges are subject to packet erasures, and public feedback of the channel state is available to both Eve and the legitimate network nodes. Secure network coding assumes that the underlying network channels are error-free; thus, if our channels introduce errors, we need to first apply a channel code to correct them, and then build security on top of the resulting error-free network. We show that by leveraging erasures and feedback, we can achieve secrecy rates that are in some cases multiple times higher than the alternative of separate channel-error-correction followed by secure network coding; moreover, we develop outer bounds and prove optimality of our proposed schemes in some special cases.

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

### **4.2 Types of Tests**

#### **4.2.1 Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately.

### 4.2.2 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error. Integration testing focuses specifically on the interactions between modules, verifying data transfer, communication protocols, and dependency management. This level of testing exposes issues that unit testing cannot detect, such as incompatible interfaces, incorrect assumptions between components, and timing problems. Integration testing may employ various strategies including big-bang, top-down, bottom-up, or sandwich approaches depending on system architecture and development methodology. Common integration test scenarios include API testing, database integration, third-party service integration, and middleware communication verification. Effective integration testing requires careful test environment setup with appropriate test data and well-defined test cases that target integration points.

### 4.2.3 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points. This phase examines the complete end-to-end functionality of the application within its intended environment, including hardware, network infrastructure, databases, and peripheral systems. System testing validates both functional and non-functional requirements such as performance, reliability, security, usability, and compliance specifications. Test cases in system testing simulate real-world usage scenarios and workflows that end users will encounter. This testing phase often uncovers issues related to system resources, system dependencies, configuration settings, and environment-specific behaviors that weren't apparent in earlier testing stages. System testing may include specialized tests like stress testing, load testing, recovery testing, and compatibility testing to ensure the system performs optimally under various conditions.

#### 4.2.4 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements. This final phase of testing determines whether the software is ready for release by verifying that it satisfies business requirements and is acceptable to end users. Acceptance testing typically occurs in an environment that closely resembles the production setting and uses realistic data sets. There are several types of acceptance testing including alpha testing (internal users), beta testing (external users), contract acceptance testing (contractual requirements), regulation acceptance testing (compliance requirements), and operational acceptance testing (operational readiness). Success criteria must be clearly defined before acceptance testing begins, often in the form of acceptance criteria attached to user stories or formal acceptance test plans. The results of acceptance testing directly influence the go/no-go decision for deployment and may identify necessary improvements for future releases.

##### **Acceptance testing for Data Synchronization:**

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updation process
- Acceptance testing ensures that the system behaves as expected from the user's perspective, confirming that the functional requirements and business goals are met.
- Testing in a production-like environment with realistic data helps identify potential issues in data synchronization, such as delays or data loss, thereby improving reliability.
- Involving real users or stakeholders in the acceptance testing process builds trust and confidence that the solution will work as expected in the production environment.
- Results of acceptance testing provide objective data for go/no-go decisions, reducing deployment risks.



### 4.3 Testing Methodologies

Testing methodologies form the foundation of a structured approach to software quality assurance. These methodologies provide systematic frameworks that guide teams through the verification and validation process, ensuring comprehensive coverage of system functionality. Methodologies typically encompass test planning, test case design, execution strategies, defect tracking, and reporting mechanisms that collectively enable organizations to maintain consistency across testing cycles. Common methodologies include Waterfall testing with its sequential phases, Agile testing with its iterative approach emphasizing continuous feedback, Test-Driven Development (TDD) where tests are written before code implementation, and Behavior-Driven Development (BDD) which focuses on collaboration between technical and non-technical stakeholders through scenarios written in natural language. Each methodology offers distinct advantages depending on project constraints, team composition, and organizational goals, with many modern teams adopting hybrid approaches that combine elements from multiple methodologies to maximize efficiency and effectiveness. Selection of an appropriate testing methodology should align with development processes, project complexity, risk factors, resource availability, and quality objectives.

### 4.4 Testing Approach

#### 4.4.1 Bottom Approach

The Bottom-Up testing approach begins with the lowest level components of the software system and progressively works upward through the integration hierarchy. This approach first tests individual units or modules in isolation, often utilizing stubs to simulate higher-level components that aren't yet integrated. As testing progresses, these individual modules are combined into subsystems which are then tested as cohesive units before being assembled into the complete application. Bottom-Up testing offers several advantages, including early detection of core functionality issues, parallel testing opportunities across independent modules, and easier fault isolation since lower-level components are thoroughly verified before integration occurs. This methodology is particularly valuable for systems with well-defined base components and when lower-level modules implement critical functionality that

forms the foundation for higher-level operations. However, it may delay the testing of high-level functionality and user interfaces until later stages in the testing process.

#### **4.4.2 Top down Approach**

The Top-Down testing approach begins with the highest level components and systematically works downward toward the more granular modules. Testing initiates with the main control structures and core interfaces, using stubs or mock objects to simulate the behaviour of lower-level components that haven't yet been integrated or thoroughly tested. This methodology allows early verification of the overall architectural design and critical system functions that users will interact with directly. Top-Down testing enables faster detection of integration issues between major components and provides early feedback on user experience aspects, making it particularly valuable for user-centric applications. As testing progresses, stubs are gradually replaced with actual components until the entire system is integrated and verified. This approach facilitates early demonstration of system functionality to stakeholders and helps identify architectural flaws before detailed implementation is complete, though it may sometimes mask low-level defects until later in the testing cycle.

#### **4.4.3 Validation**

Validation testing focuses on determining whether the developed software meets specified requirements and fulfills its intended purpose in the operational environment. Unlike verification which confirms the system is built correctly, validation confirms that the correct system was built to address actual user needs. This process involves evaluating the software against user requirements, business rules, and expected behaviors through techniques such as acceptance testing, alpha/beta testing, and usability assessment. Validation typically involves end-users or their representatives and examines the system from their perspective rather than from a technical standpoint.

It answers the fundamental question: "Are we building the right product?" by confirming the software provides value to users and solves their actual problems. Effective validation requires clear requirement specifications, representative test scenarios, and meaningful evaluation criteria that reflect real-world usage. The

outcome of validation testing directly impacts the decision to release software, as it ultimately determines whether the system satisfies stakeholder expectations and business objectives.

## 4.5 Test cases

Unit tests focus on individual functions and modules to verify their logic and correctness.

Test Case ID	Test Case Description	Input	Expected Output	Actual Output
UT001	Admin login authentication	Valid admin credentials	Admin dashboard access granted	Pass
UT002	Company registration	Valid company info	Company successfully registered	Pass
UT003	File upload by company	Upload PDF file	File stored and block generated	Pass
UT004	Oracle login	Valid oracle credentials	Access to assigned shipment verification	Pass
UT005	Shipment creation	Product ID, sender, receiver	Shipment record created on blockchain	Pass
UT006	Shipment status update	Product marked as 'Delivered OK'	Shipment status updated in blockchain	Pass
UT007	Document download	Click download for a verified shipment	File is downloaded	Pass

**Fig No.4.5.1** Test cases

System test cases validate end-to-end flows and integration between components.

Test Case ID	Test Case Description	Input	Expected Output	Actual Output
ST001	Admin adds new company and assigns roles	New company data and role selection	Company added	Pass
ST002	Upload document without selecting company	Without Selecting company and upload image file	Please Select company	Fail
ST003	Oracle verifies delivered product	Oracle marks product as delivered	Shipment status updated	Pass
ST004	Cross-network file transfer	Company A uploads, Company B receives	File securely transferred and confirmed	Pass
ST005	Admin views all transaction history	Login and select transaction history	Comprehensive transaction report displayed	Pass
ST006	Unauthorized user tries login	Incorrect credentials	Access denied	Pass
ST007	Real-time product tracking	Enter tracking ID	Display current shipment status	Pass
ST008	System handles invalid file type	Upload .exe file	Upload rejected with error message	Pass

**Fig No.4.5.2** Test cases

## 5. Implementation

### 5.1 Libraries Used To Impact In Project

#### Frontend Libraries

- **React.js:** Core JavaScript library for building the user interface with reusable components, enabling a responsive and dynamic user experience for the blockchain messaging platform.
- **React Router:** Used to implement navigation and routing within the single-page application, allowing seamless transitions between different sections of the messaging platform.
- **Redux:** State management library that maintains application state, crucial for managing user sessions, message histories, and blockchain transaction states.
- **Bootstrap:** CSS framework providing responsive design components and grid systems, ensuring consistent styling and mobile compatibility across the messaging interface.
- **Axios:** Promise-based HTTP client for making API requests to the Spring Boot backend, handling asynchronous operations for message sending and blockchain interactions.

#### Backend Libraries

- **Spring Boot:** Java-based framework for building the RESTful API web server, providing dependency injection, security features, and application configuration.
- **Spring Security:** Authentication and authorization framework integrated with Spring Boot to secure API endpoints and user sessions.
- **Spring Data JPA:** Simplifies database operations and ORM (Object-Relational Mapping) to interact with MySQL database.
- **MySQL Connector/J:** JDBC driver enabling Java applications to connect with MySQL database for storing user profiles and message metadata.

## **Cryptography and Blockchain Libraries**

- **SHA-256:** Cryptographic hash function implemented for blockchain block validation, transaction verification, and secure password hashing.

Each of these libraries contributes significantly to the development of a secure, private blockchain-based instant messaging platform, addressing the requirements for decentralized communication, end-to-end encryption, and responsive user experience.

## 5.2 SAMPLE SCREENS

### Frontend code:

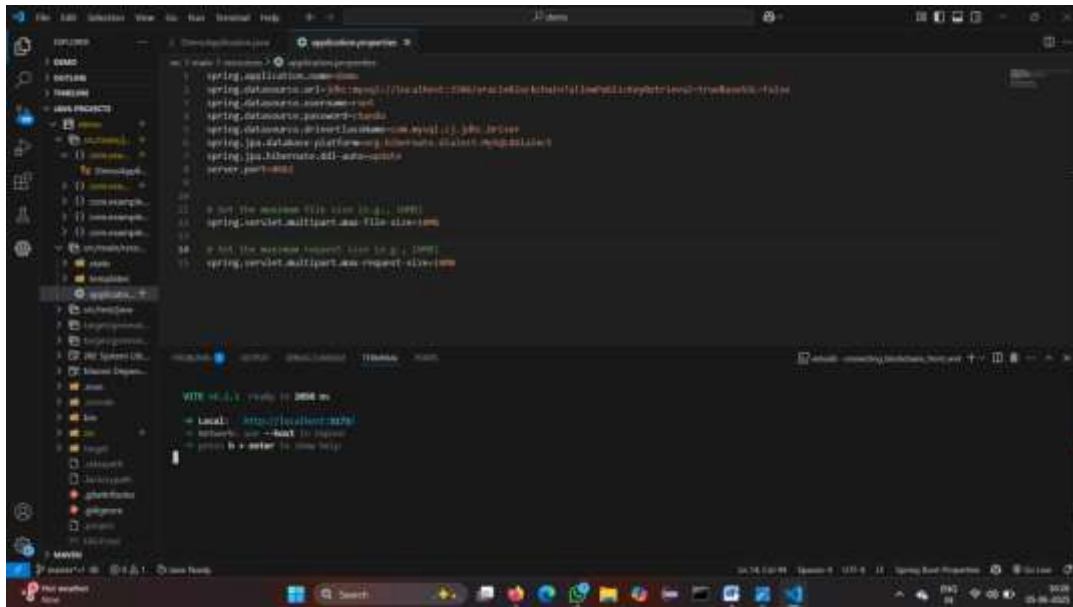
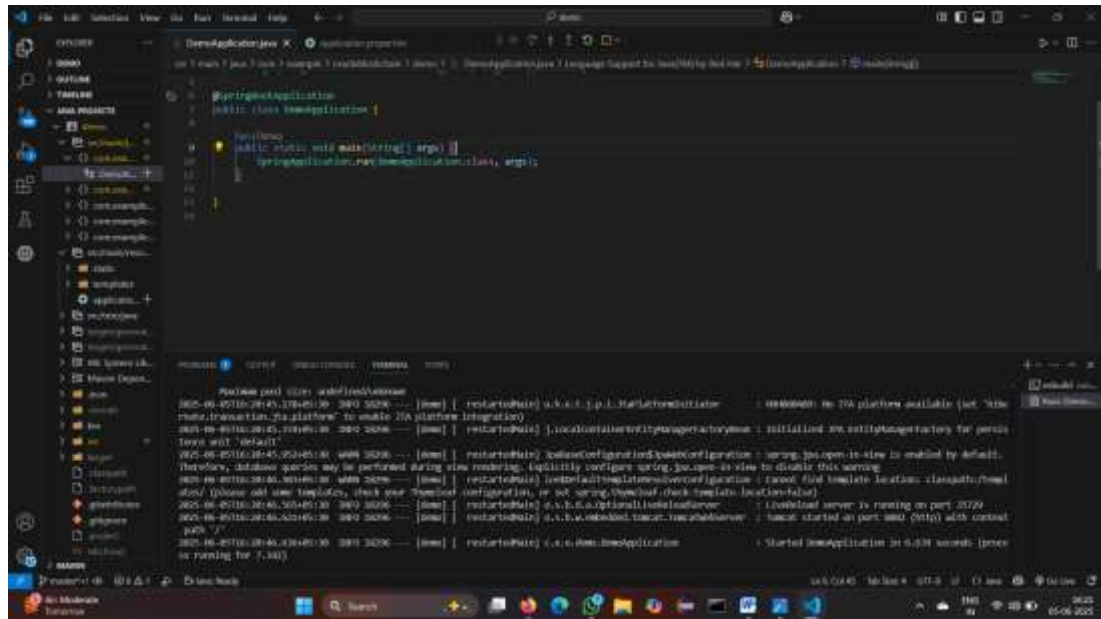


Fig No.5.2.1 Frontend code

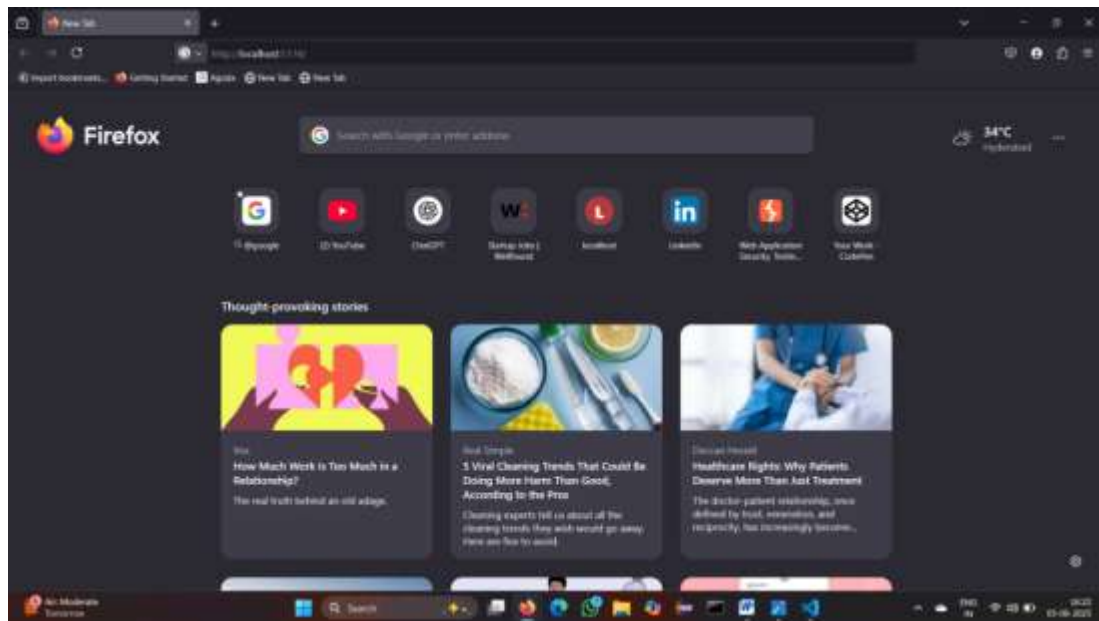
### Description:

This is the frontend code with the using of Html, Css,ReactJs .and the terminal using the command is “npm run dev” its give the url .

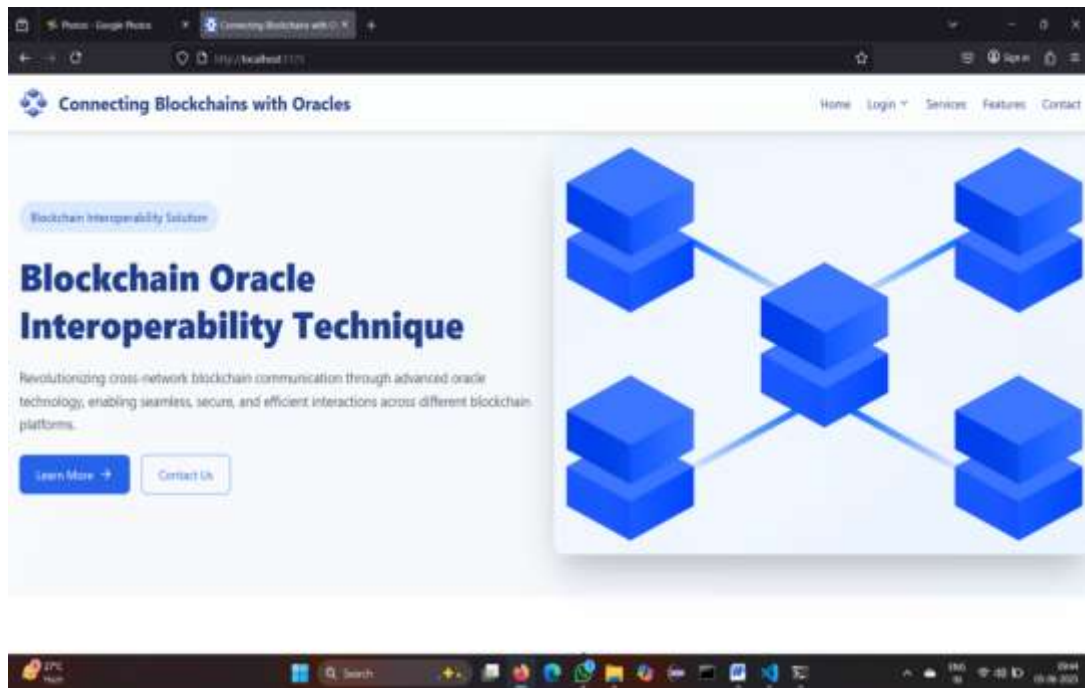
**Backend Code:****Fig No.5.2.2** Backend code**Description:**

This is the Backend code with the using of Java, Springboot and mySql.

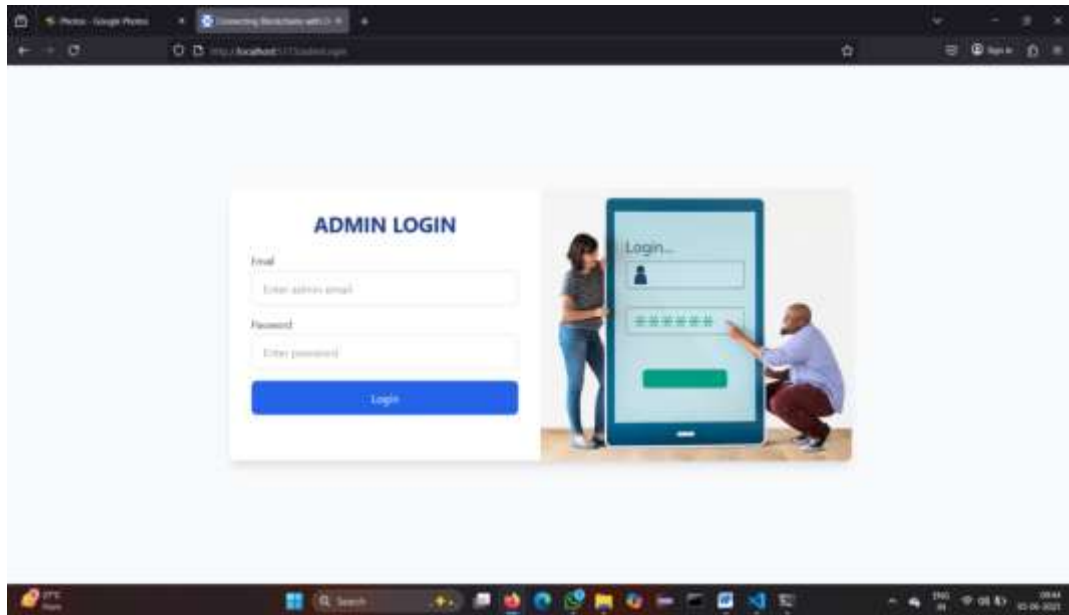


**Browser Interface:****Fig No.5.2.3** Browser Interface**Description:**

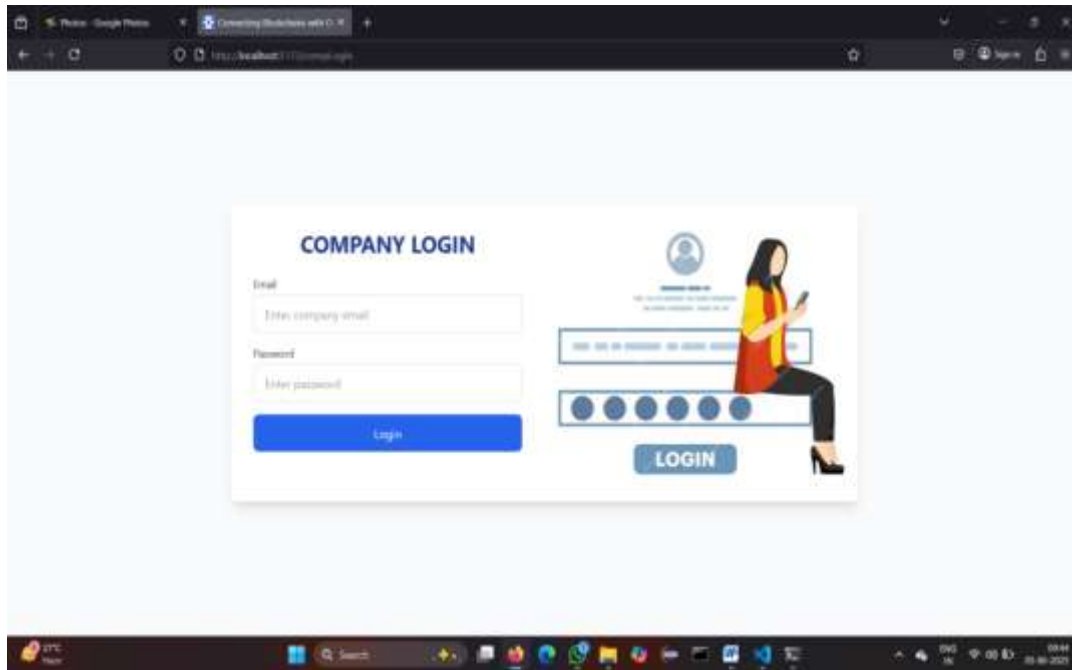
After completion of the successful the run the fronted and backend code then paste the url in browser and its connect to the server then show the dashboard of “blockchain oracle interapability technique for permissioned blockchain”.

**Home Page:****Fig No.5.2.4** Home**Description:**

This project showcases an advanced solution for seamless cross-blockchain communication

**Admin Login:****Fig No.5.2.5** Admin login**Description:**

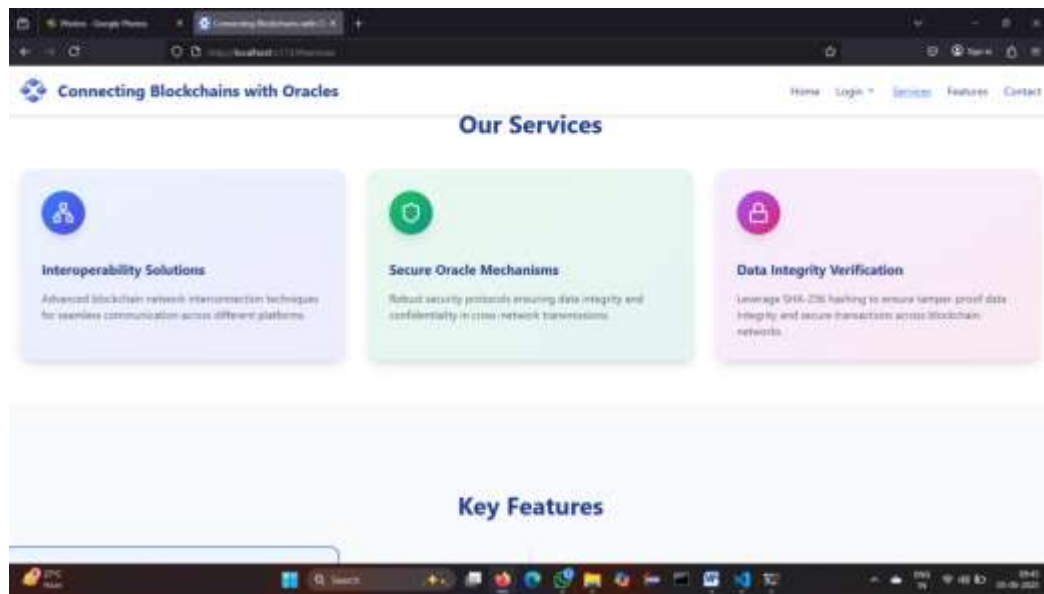
This **Admin Login** interface allows administrators to securely access the system by entering their credentials. On the right, an engaging illustration reinforces the login concept for an intuitive and user-friendly experience.

**Company Login Page:****Fig No.5.2.6** Company login page**Description:**

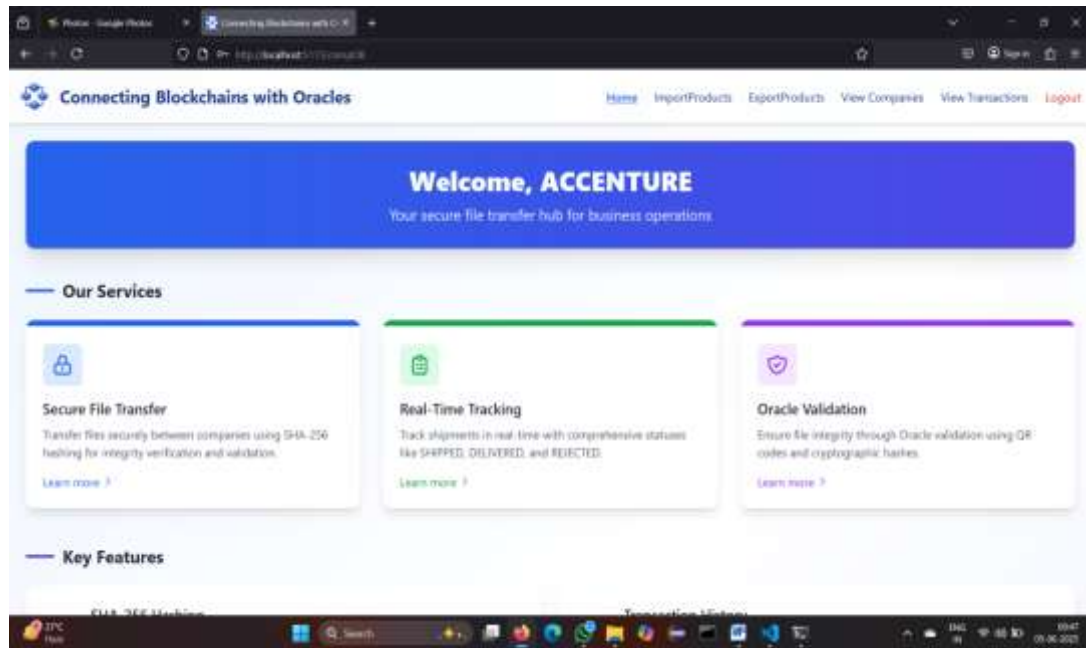
This **Company Login** page provides a secure gateway for company employees to access the system using their email and password.

**Oracle Login Page:****Fig No.5.2.7** Oracle login page**Description:**

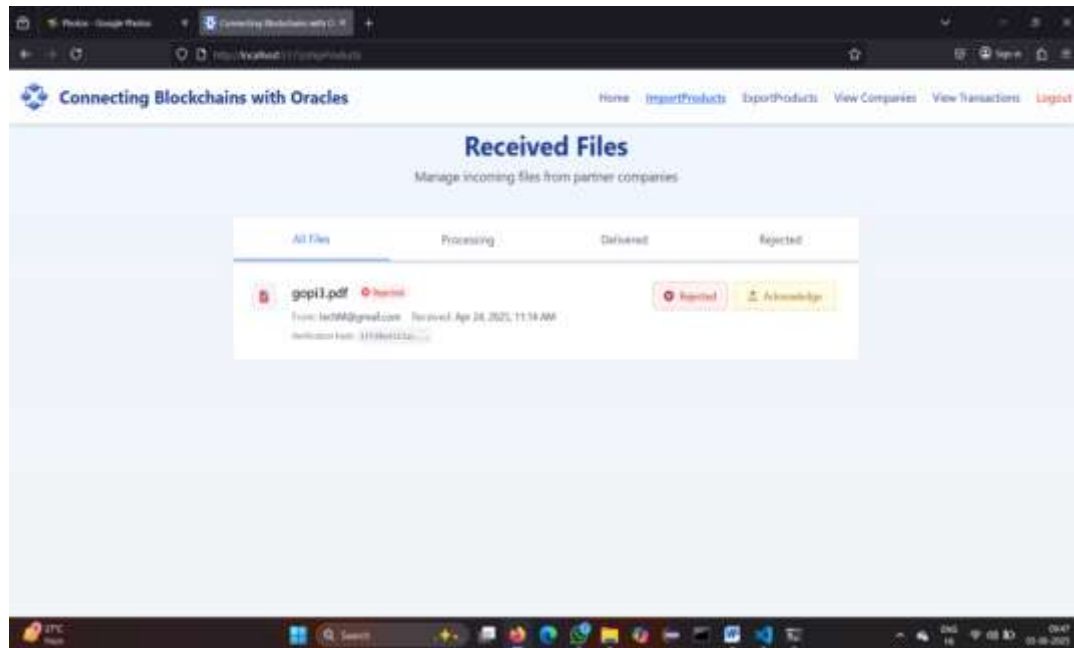
This **Oracle Login** page allows admin-level access to the blockchain oracle system with a modern, intuitive interface.

**Services Page:****Fig No.5.2.8** Service Page**Description:**

The services include Interoperability Solutions for seamless blockchain communication, Secure Oracle Mechanisms for data integrity, and Data Integrity Verification using advanced cryptographic methods.

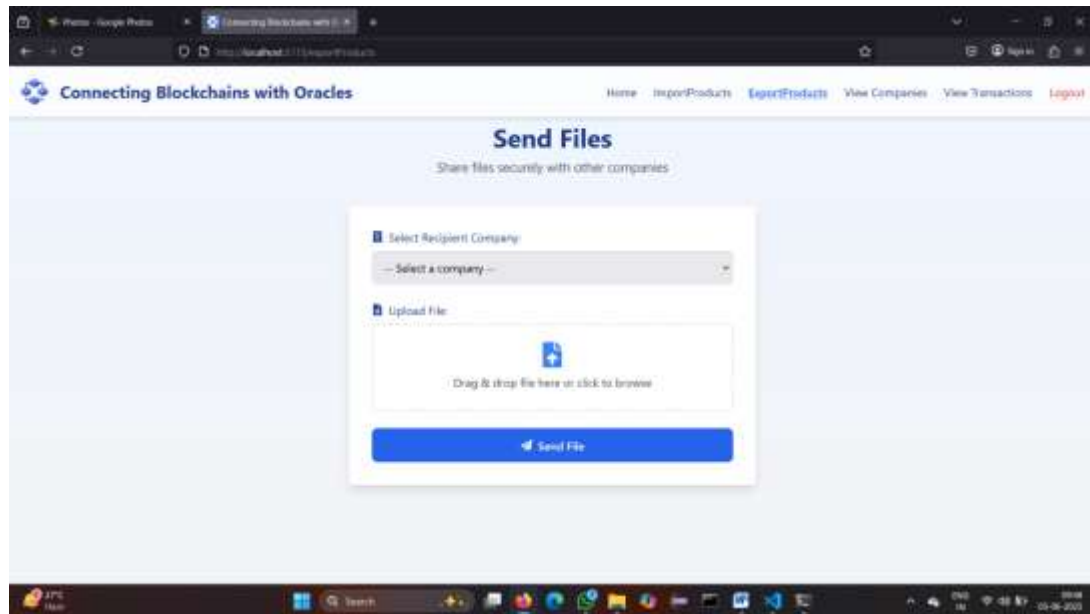
**Company Home Page:****Fig No.5.2.9** Company Home Page**Description:**

This is the company home page after successfully login the company login shows like this features .

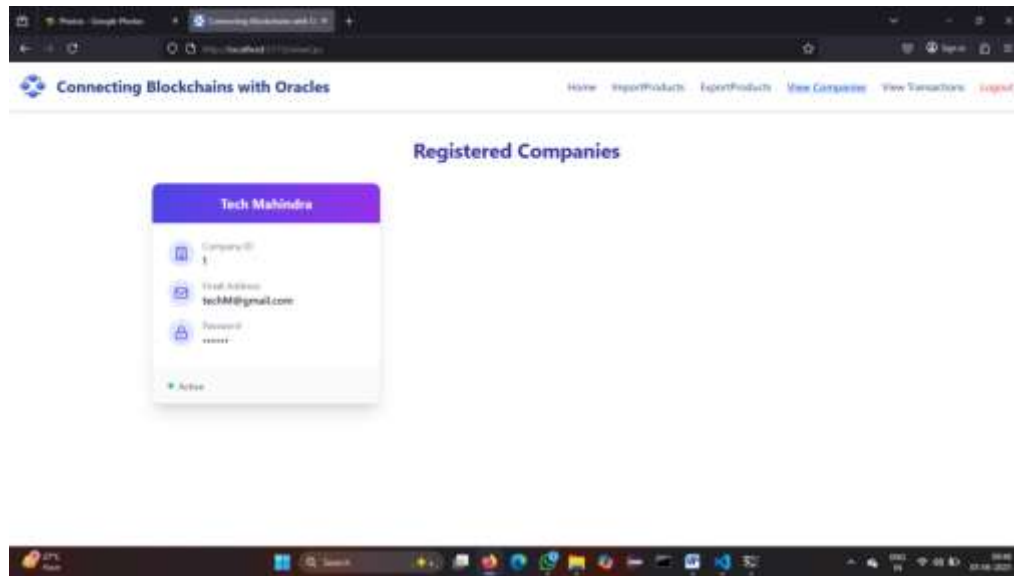
**ImportProducts Page:****Fig No.5.2.10** ImportProducts Page**Description:**

After completion the transfer files between one company to another company in this page we will see the received files.

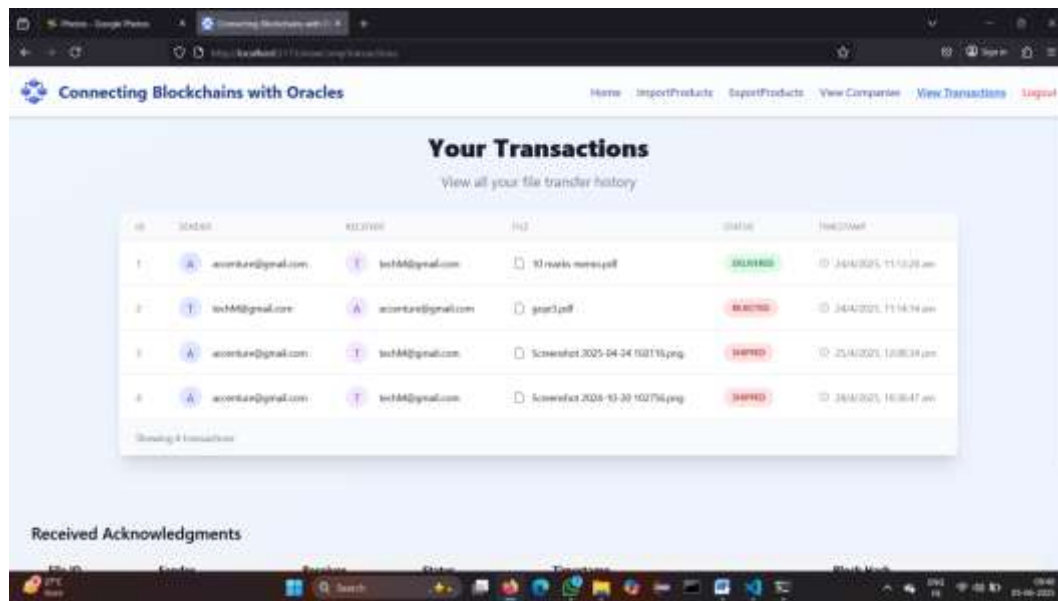


**ExportProducts Page:****Fig No.5.2.11** ExportProduct Page**Description:**

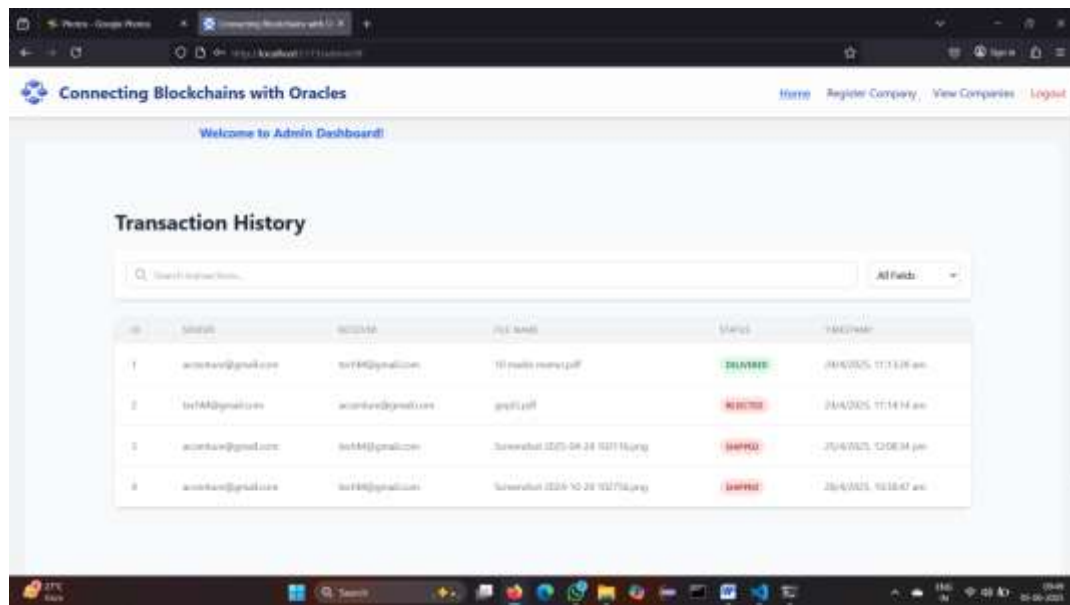
The page enables users to securely send files by selecting the recipient company and uploading the desired file.

**View Companies:****Fig No.5.2.12** View companies**Description:**

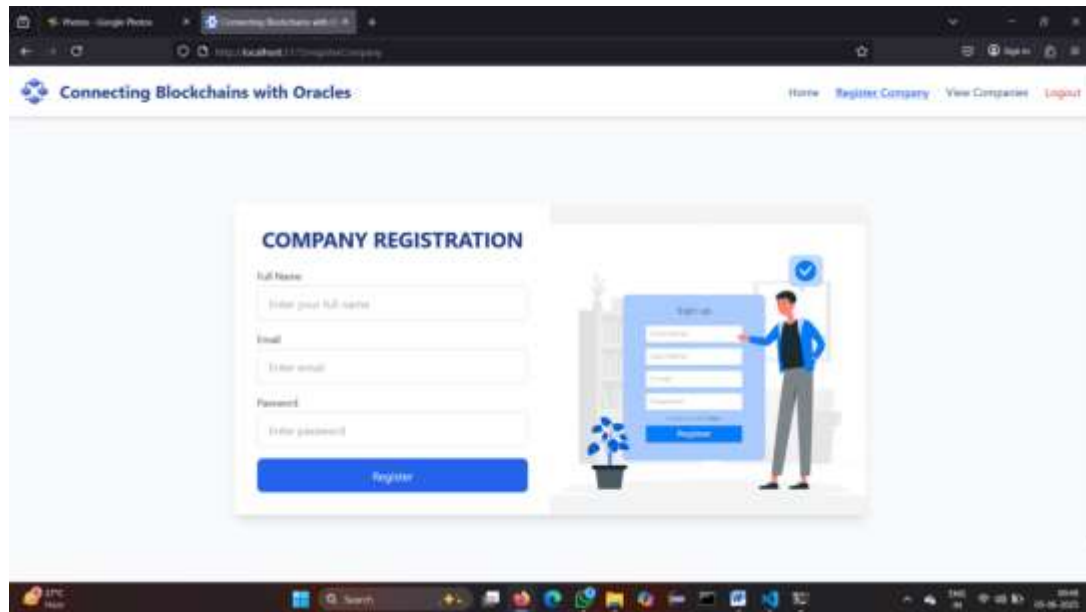
The page displays a list of registered companies, showing their name, ID, email, and account status.

**View Transaction Page:****Fig No.5.2.13** View Transation Page**Description:**

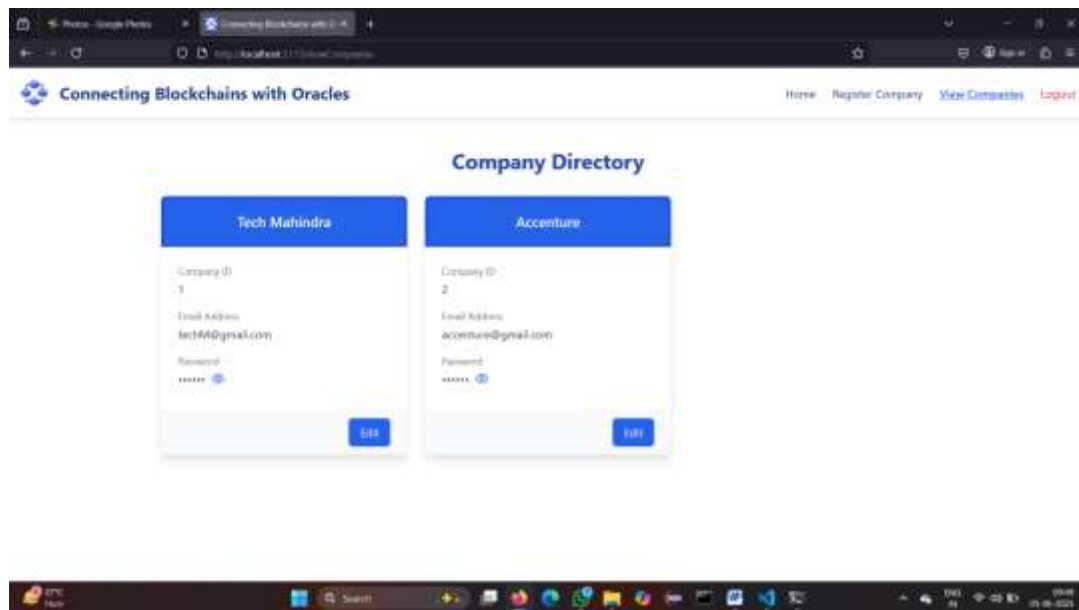
This page provides a history of all file transactions between companies, showing sender, receiver, file details, status, and timestamp.

**Admin Dashboard Page:****Fig No.5.2.14** Admin Dashboard Page**Description:**

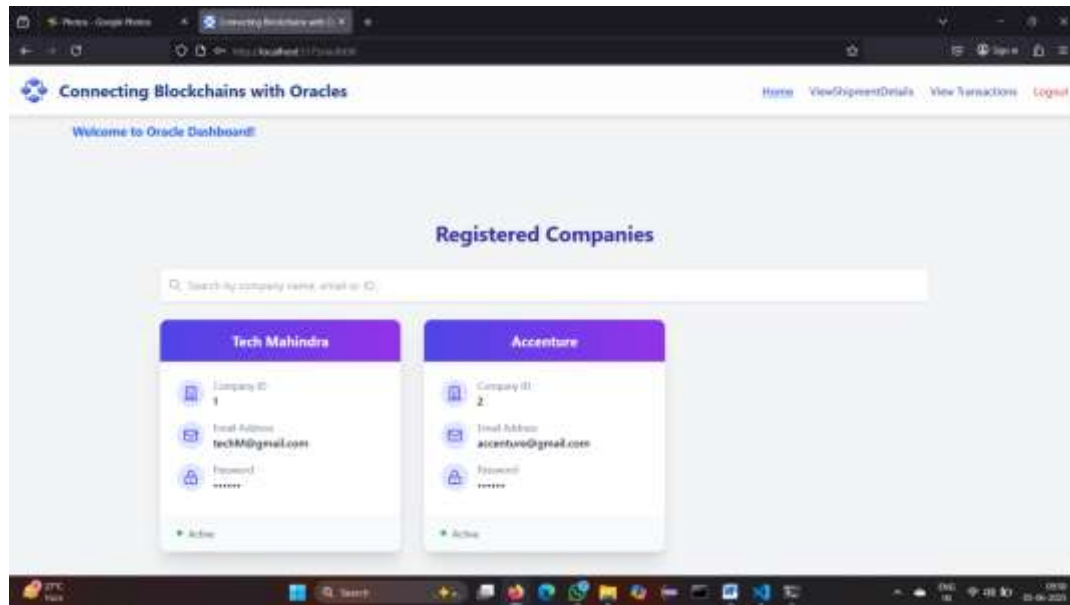
This is the admin dashboard page here shows the entire transaction history across the all companies.

**Company Registration Page:**The image is a screenshot of a web browser displaying a company registration page. The browser's address bar shows the URL 'http://localhost:1177/registerCompany'. The page header features the logo 'Connecting Blockchains with Oracles' on the left and navigation links 'Home', 'Register Company', 'View Companies', and 'Logout' on the right. The main content area is titled 'COMPANY REGISTRATION' and contains a registration form with three input fields: 'Full Name' (placeholder: 'Enter your full name'), 'Email' (placeholder: 'Enter email'), and 'Password' (placeholder: 'Enter password'). A blue 'Register' button is positioned below the form. To the right of the form is an illustration of a person in a blue jacket and grey pants standing next to a large blue screen that displays a simplified version of the registration form. A small blue checkmark icon is visible above the person's head. The browser's taskbar at the bottom shows various application icons and the system clock indicating 10:46 on 02-06-2023.**Fig No.5.2.15** Company Registration Page**Description:**

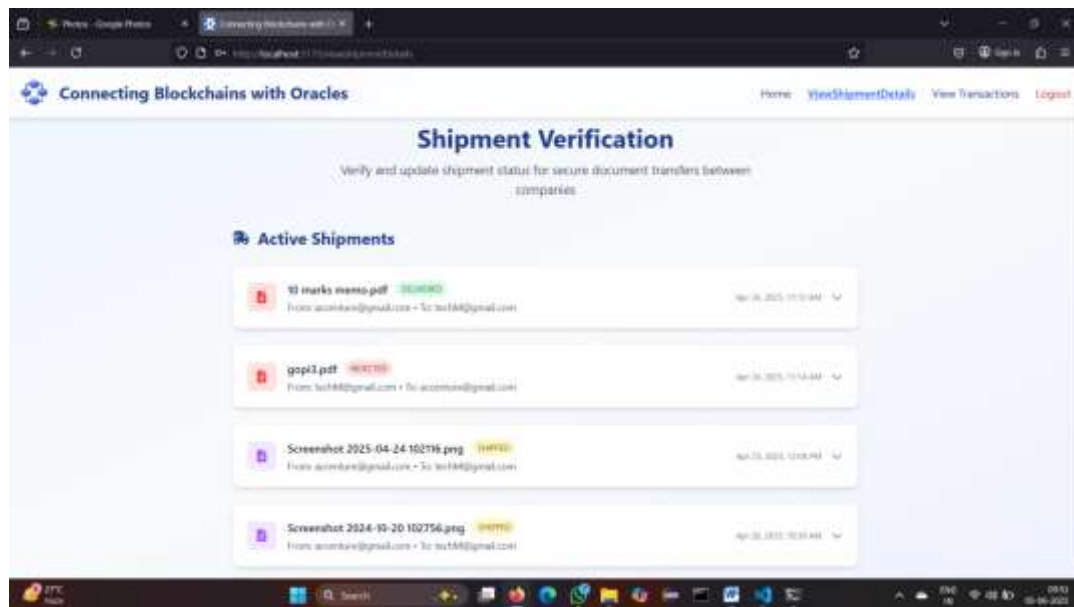
This page allows new companies to securely register on the platform by entering their full name, email, and password.

**View Companies Page:****Fig no:5.2.16** View Companies Page**Description:**

The image shows a web page titled "**Connecting Blockchains with Oracles**", displaying a **Company Directory** with cards for **Tech Mahindra** and **Accenture**, each showing company ID, email and details.

**Oracle Dashboard page:****Fig No.5.2.17** Oracle Dashboard Page**Description:**

This is the Oracle Dashboard and here shows the register companies.

**ViewShipment Verification page:****Fig No.5.2.18** View Shipment Verification Page**Description:**

The image displays a **Shipment Verification** page listing **active document shipments** between companies with file names, sender and recipient emails, status tags (DELIVERED, REJECTED, SHIPPED), and timestamps.



**Transaction History Page:**

ID	SENDER	RECIPIENT	FILE NAME	STATUS	TIMESTAMP
1	account@gmail.com	tech@b@gmail.com	10 rocks manuscript.pdf	SUCCESS	24/6/2023, 11:13:06 am
2	tech@b@gmail.com	account@gmail.com	apple.pdf	REJECTED	24/6/2023, 11:14:56 am
3	account@gmail.com	tech@b@gmail.com	Screenshot 2023-04-04 10:11:16.png	SUCCESS	25/6/2023, 5:08:34 pm
4	account@gmail.com	tech@b@gmail.com	Screenshot 2024-10-22 10:27:56.png	SUCCESS	20/6/2023, 10:18:47 am

**Fig No.5.2.19** Transaction History**Description:**

The image shows a **Transaction History** table listing document exchanges between companies with details like sender, receiver, file name, status, and timestamp.

## **CONCLUSION**

In conclusion, the proposed Blockchain Oracle Interoperability Technique for permissioned blockchains significantly enhances cross-network communication and secure data exchange within enterprise environments. By integrating the Admin, Company, and Oracle modules, the system ensures robust user authentication, seamless shipment tracking, and verifiable transaction management. The modular architecture promotes scalability, maintainability, and clear separation of responsibilities, making it suitable for real-world enterprise deployment. The use of oracles facilitates trusted interaction between disparate blockchain networks, enabling secure and efficient smart contract execution. With real-time product tracking, document verification, and role-based access controls, the system enhances operational transparency and data integrity. The implementation supports multiple file formats, ensuring flexibility in handling various types of organizational data. Cross-network transaction performance has been optimized, with latency kept to a minimum for real-time operations. Through comprehensive testing and a working prototype, the project demonstrates the feasibility and effectiveness of the solution. This approach not only addresses interoperability challenges but also paves the way for broader adoption of permissioned blockchains in multi-organizational ecosystems.

