INDUSTRY INTERNSHIP REPORT

On

"CORE JAVA"

By

KALASAPATI JAYANI (219X1A2819)

Submitted in accordance with the requirement for the degree of B-Tech in

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTHAPURAMU)

KURNOOL - 518007

2024 – 2025

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTHAPURAMU)

CERTIFICATE

This is to certify that the internship titled 'CORE JAVA' **is a bonafide record of work carried out by KALASAPATI JAYANI (219X1A2819), and the Internship report is submitted in partial fulfillment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering.**

……..
Internship Guide
Sri. M. Raghavendra Reddy
Assistant Professor
Department of C.S.E.,
G. Pulla Reddy Engineering College,
Kurnool.

……..
HEAD OF THE DEPARTMENT
Dr. N. Kasiviswanath
Professor & Head of the Department,
Department of C.S.E.,
G. Pulla Reddy Engineering College,
Kurnool.

INTERNSHIP DETAILS

Name of the College: G. Pulla Reddy Engineering College

Department: Computer Science and Engineering

Name of the Student: Kalasapati Jayani

Register Number: 219X1A2819

Name of the Faculty Guide: Sri. M. Raghavendra Reddy

Duration of the Internship: From 02-01-2025 To 12-04-2025

Name and address of the Organization: Global Quest Technologies,Yelhanka ,Bangalore.

Internship Domain: Core Java

Internship Title: Core Java

Internship Stipend: ---

Date of Internship Report Submission:

Internship Coordinator

Dr.B.Geetha Vani
Professor
CSE Department GPREC

DECLARATION

I **Kalasapati Jayani**, a student bearing the Roll No. **219X1A2819,** of the Department of Computer Science and Engineering, G. Pulla Reddy Engineering College, do hereby declare that the I have completed the mandatory internship titled **Core java** from **02-01-2025** to **12-04-2025** in **Global Quest Technologies**, **Bangalore** under the Faculty Guide ship of **G. R. Narendra**.

(Student Signature and Date)

Certificate From Organization

Certificate from Organization

This is to certify that Jayani (Name of the Intern) Reg.No 21XMA2819 of G. Pulla Reddy Engineering College (Name of the College) underwent Internship in Global Quest Technologies (Name of the Intern Organization) from 02-01-25 to 12-04-2025

The overall performance of the Intern during his/her internship is found to be Good (Satisfactory/Good).

GLOBAL QUEST TECHNOLOGIES Judicial Layout Yelahanka Bengaluru 560 064 080-4128 0000

29/4/25

Authorized Signatory with Date and Seal

Internship Project completion certificate

GLOBAL QUEST
TECHNOLOGIES

12-04-2025

Dear Kalasapati Jayani,
G Pulla Reddy Engineering College,
Near Pasupula Village, Kurnool - Nandyal,
Main Road, Kurnool, Andhra Pradesh 518007

SUB: INTERNSHIP COMPLETION LETTER

Dear Sir/Madam,

We are glad to inform you that **Ms. Kalasapati Jayani**, USN NO. **219X1A2819 from "G Pulla Reddy Engineering College" –Andhra Pradesh** has completed her internship in our organization from 02-01-2025 to 11-4-2025 on **Internship on Core Java.**

During the internship, she was exposed to the various activities in **Internship/ Core Java.**

We found she extremely inquisitive and hardworking, she is also willing to put her best effort and get into the depth of the subject to understand it better.

We congratulate her on the Completion of the Internship and wish her all the Best for all her future endeavors!

Sincerely,
For Global Quest Technologies

GLOBAL QUEST TECHNOLOGIES
Judicial Layout
Yelahanka
Bengaluru
560 064
080 - 4128 0[illegible]

G. R. Narendra Reddy

Managing Director

12/4/25

Note: - All work/developments as part of the Project/ Internship will be the property of Global Quest Technologies or its clients and are to be treated as confidential & not shared with third parties.

324, 2nd Floor, 3rd 'A' Cross
Near Seshadripuram First Grade College
Above City City Union Bank
Yelahanka New Town, Bengaluru - 560064

+91 - 944 840 3469

info@gqtech.in
www.gqtech.in

ACKNOWLEDGEMENT

I wish to express my deep sense of gratitude to my internal supervisor **Sri.M. Raghavendra Reddy**, **Assistant Professor** of Computer Science and Engineering Department, G. Pulla Reddy Engineering College.

My sincere thanks to our Internship Coordinator **Dr.B.GeethaVani** for her immaculate guidance, constant encouragement and cooperation which have made possible to bring out this internship work.

My sincere thanks to **Mr. G. R. Narendra**, **Director,** of Organization , for helping me and giving me the required information needed for my internship work.

I am thankful to our Head of the Department **Dr.N.Kasiviswanath**, for his whole hearted support and encouragement during the internship.

We are grateful to our Internship Dean **Dr.Y.V.Siva Reddy** and to our respected Principal **Dr.B.Sreenivasa Reddy** for providing requisite facilities and helping us in providing such a good environment.

I wish to convey my acknowledgements to all the staff members of the Computer Science and Engineering Department for giving the required information needed for my internship work.

Finally, I wish to thank all our friends and well wishers who have helped me directly or indirectly during the course of this internship work.

Table of Contents

LIST OF FIGURES

1. INTRODUCTION

1. INTRODUCTION

1.1. ABOUT THE INTERNSHIP COMPANY/ORGANIZATION

Global Quest Technologies: Building Coders. Creating Futures.

Founded in 2015, Global Quest Technologies is a leading force in software development and IT training, headquartered in Bangalore—the tech capital of India. With a strong emphasis on industry-relevant skills, we specialize in delivering expert-led training in Core Java, equipping learners with the tools needed to thrive in today's dynamic software landscape. At Global Quest, we merge academic precision with real-world relevance. Our mission is to empower aspiring developers with deep foundational knowledge, practical coding skills, and problem-solving capabilities—setting them up for success in software development careers.

Our Core Java program is meticulously designed to meet industry standards and covers everything a developer needs to build robust, scalable applications. From object-oriented programming principles to exception handling, file I/O, and multithreading, we ensure our learners gain a thorough understanding of the language's core concepts and how they apply to enterprise-level projects.

1.1.1. Our Mission

To deliver industry-aligned, hands-on Java training that builds strong programming foundations and prepares learners for real-world challenges in software development.

1.1.2. Our Vision

To be a benchmark in Java education by combining rigorous training, mentorship, and project-based learning that empowers a new generation of software professionals.

1.1.3. Pioneering Core Java Training Across Domains

Software Development

Our Core Java course forms the backbone for careers in backend development, application programming, and web technologies. With a strong emphasis on real-time projects and debugging practices, we prepare students to be job-ready from day one.

Corporate Training

Trusted by IT firms, we offer tailored Core Java programs to upskill teams and improve software quality. Our training modules integrate best coding practices, design patterns, and real-time use cases.

Academic Bridging

We partner with colleges and universities to bridge the gap between academic theory and industry needs. Through workshops and project-based learning, we equip students with the coding skills that employers value.

Core Java Curriculum Highlights:

- Fundamentals of Java & OOPs Concepts
- Data Types, Variables, Operators, and Control Statements
- Classes, Objects, Constructors, and Inheritance
- Polymorphism and Abstraction
- Interfaces and Packages
- Exception Handling
- File Handling (java.io, java.nio)
- Collections Framework (List, Set, Map)
- Threads and Multithreading
- Java API Libraries and Utilities

At Global Quest Technologies, we're more than educators—we’re enablers of transformation. Whether you're a student aiming to break into the tech industry or a professional looking to strengthen your Java foundation, our training ensures you're equipped for tomorrow’s software challenges.

1.2. OFFLINE INTERNSHIP DETAILS

During my internship at Global Quest Technologies, Bangalore, I experienced a progressive and immersive journey in Core Java development. Starting with fundamental programming concepts and moving toward object-oriented design and application-level coding, the internship was structured to build my skills step by step. Constant support from mentors and hands-on assignments helped me gain clarity, confidence, and competence in Java programming.

The internship was divided into two key phases:

Phase 1: Training Phase

The internship began with a focused training phase aimed at building a solid foundation in Core Java concepts. This phase involved a blend of theoretical sessions, live coding demonstrations, and regular coding exercises. Each topic was explained in depth, followed by practical tasks that reinforced learning.

Topics covered during training:

- Basics of Java: syntax, variables, data types, operators
- Control Statements: if-else, switch, loops (for, while, do-while)
- Object-Oriented Programming (OOP): classes, objects, inheritance, polymorphism, encapsulation, abstraction
- Constructors, Method Overloading & Overriding
- Arrays and String
- Exception Handlin
- File I/O using java.io
- Multithreading and Concurrency Basics
- Collections Framework: List, Set, Map
- Real-time debugging and use of IDEs (e.g., Eclipse, IntelliJ)

Assignments and mini-projects were part of this phase, helping me to apply the concepts in code and develop logical thinking. The training sessions were interactive and paced to ensure a deep understanding of the language.

Phase 2: Project Phase

After completing the training, I transitioned into the project phase, where I applied my Java knowledge to solve real-world coding problems and build core Java applications. This hands-on phase

was designed to simulate real development environments and challenges.

Project tasks included:

- Designing and implementing a Java-based library management system
- Creating console-based applications with file handling and exception control
- Implementing multi-threaded programs for simulation tasks
- Developing a mini customer management system using collections
- Applying OOP concepts in creating reusable and scalable code modules

Under the mentorship of experienced developers, I was guided through every stage of the project lifecycle—from understanding requirements to writing, testing, and optimizing code.

Live Projects & Industry Application

As a final part of the internship, I had the opportunity to work on four live projects that mirrored real industry scenarios. These projects allowed me to:

- Apply OOP principles to practical problem-solving
- Enhance my debugging, testing, and documentation skills
- Understand best practices in code structure and naming conventions
- Collaborate in a simulated team environment using version control (Git)

These projects served as a vital bridge between academic learning and industrial application. With mentor support, I learned not only how to code, but how to think like a developer—a mindset crucial for any career in software development.

The internship at Global Quest Technologies was an invaluable experience that strengthened my Core Java skills and prepared me for the real-world challenges of software development.

2. WORK DONE DURING INTERNSHIP

-

1. WORK DONE DURING INTERNSHIP

2.1 ABOUT COURSE/DOMAIN

2.1.1 Core Java

Core Java refers to the fundamental aspects of the Java programming language, which form the base for all Java-based development, including web, desktop, mobile, and enterprise applications. Core Java is platform-independent, object-oriented, and robust—making it one of the most widely used programming languages in the software industry today.The Core Java curriculum at Global Quest Technologies is designed to build strong programming fundamentals and problem-solving capabilities. It helps learners understand the key concepts behind object-oriented programming (OOP), application logic, data processing, and user interaction.

By mastering Core Java, developers are equipped to build scalable and maintainable code structures, essential for software development roles in industries such as banking, telecom, healthcare, and more. This knowledge also serves as the gateway to more advanced technologies like Java EE (Enterprise Edition), Android development, and Spring Framework.

2.2.2 Key Applications of Core Java Across Various Domains

(i) Software Development

Core Java is used to build console-based utilities, desktop applications, automation tools, and system-level services. Its strong OOP principles allow for creating reusable and efficient code modules.

(ii) Android Development (Base Layer)

While Android uses its own SDK, Core Java concepts are essential for building Android apps, especially for handling back-end logic, user input, and local data storage.

(iii) Enterprise Applications

Core Java forms the basis for server-side development in large-scale enterprise systems, especially in banking, insurance, and ERP domains.

(iv) Testing & Automation

Core Java is widely used in automation testing frameworks like Selenium, where it helps in scripting test cases, assertions, and automation flows.

(v) Embedded Systems

Java's portability makes it suitable for embedded devices like set-top boxes, smartcards, and IoT devices.

(vi) Big Data Tools Integration

Core Java is used in building modules that interact with big data technologies like Apache Hadoop

and Apache Spark, especially for data ingestion and pre-processing tasks.

(vii) Academic Projects & Simulations

Core Java is the language of choice in academic environments for implementing data structures, algorithms, and logic-based systems.

2.1.3 Popular Tools and Concepts Used in Core Java

(i) JDK (Java Development Kit)

Provides the tools and environment to write, compile, and run Java programs.

(ii) Eclipse / IntelliJ IDEA

Popular IDEs (Integrated Development Environments) used for writing and debugging Java applications.

(iii) Java Virtual Machine (JVM)

Enables Java programs to run on different operating systems without modification.

(iv) OOP Concepts

Includes abstraction, encapsulation, inheritance, and polymorphism—key to building modular and reusable applications.

(v) Exception Handling

Essential for writing stable applications that can gracefully handle runtime errors using try, catch, finally, and throw.

(vi) File I/O (java.io, java.nio)

Used to read and write data to files, perform stream-based operations, and manage file systems.

(vii) Multithreading

Allows concurrent execution of two or more threads, improving the performance of applications that require multitasking.

(viii) Collections Framework

Provides data structures like ArrayList, HashMap, HashSet, and more for efficient data management.

(ix) Java APIs and Utilities

In-built packages and libraries that aid in math operations, date/time handling, regular expressions, and more.

(x) Version Control with Git & GitHub

Used for code versioning, collaboration, and maintaining different versions of Java applications during project development.These tools and concepts were integral throughout the internship and provided a strong base for building scalable, efficient, and real-world Java applications. The course structure and project-based approach at Global Quest Technologies ensured a complete understanding of both the theoretical, practical aspects of Java & preparing interns for challenges.

2.2. TECHNOLOGIES LEARNT DURING INTERNSHIP

During the internship at Global Quest Technologies, Bangalore, the focus was exclusively on mastering Core Java. The training was structured to provide a deep understanding of Java programming fundamentals, object-oriented design principles, and key APIs. The learning journey began with the basics and gradually progressed to more advanced features, equipping us with the skills necessary to develop robust, modular, and efficient Java applications.

2.2.1. Core Java Programming

The training started with a thorough introduction to Core Java, covering the structure and syntax of Java programs. This included:

- Variables, Data Types, and Operators
- Control Flow Statements (if, switch, for, while, do-while)
- Methods and Parameter Passing
- Object-Oriented Programming Concepts:
- Encapsulation, Inheritance, Polymorphism, Abstraction
- Constructor Overloading & Method Overloading

2.2.2. Exception Handling

We learned how to manage run-time errors using Java's exception-handling model. Topics included:

- try, catch, finally, throw, and throws
- Custom Exceptions
- Checked vs Unchecked Exceptions

2.2.3. Collections Framework

A key part of the training was learning how to manage groups of data using Java's Collections Framework, including:

- List, Set, and Map interfaces
- Classes like ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap
- Iterators, Enhanced for-loop, and Lambda expressions for traversing collections

2.2.4. File Handling and Streams

We explored how to handle input and output in Java, focusing on:

- File Reading/Writing using File, FileReader, BufferedReader, FileWriter, BufferedWriter
- Serialization and Deserialization
- Streams and the I/O package

2.2.5. Multithreading and Concurrency

This module introduced the concepts of multithreading to allow concurrent execution:

- Creating threads using Thread class and Runnable interface
- Thread lifecycle and priorities
- Synchronization and Inter-thread Communication

2.3. ASSESSMENTS / TASKS ASSIGNED DETAILS

As part of the internship, each module of Core Java training was reinforced with well-structured assessments, assignments, and practical tasks. These helped solidify understanding and apply concepts to real-world-like scenarios.

2.3.1. Java Assignments

Each completed module contained a mix of study materials (PDFs, slides), assignments, and interview preparation tasks. Assignments were designed to test both conceptual understanding and coding capability.

Topics and assignment categories included:

- Basic Java Programs
  Examples: Calculators, palindrome checkers, prime number generators
- Object-Oriented Programming
  Tasks to implement classes, inheritance hierarchies, abstraction with interfaces, abstract classes, etc.
- Exception Handling Exercises
  Creating robust applications with custom error handling
- Collections & Data Structures
  Building and manipulating ArrayList, HashMap, sorting mechanisms, ad performing CRUD operations
- File Handling Projects
  Reading and writing text files, performing word count, log processing, etc.
- Multithreading Tasks
  Simulating ticket booking systems or bank transaction simulators using threads and synchronization

3.PROJECTS/MODULES COMPLETED

3. PROJECTS / MODULES COMPLETED

3.1. ABOUT MODULE 1

Core Java

Core Java is the foundational platform for developing applications in Java, offering essential features such as object-oriented programming (OOP) principles, data structures, exception handling, multithreading, and more. As a general-purpose, high-level programming language, Java is widely used for web applications, mobile applications, and enterprise software. Java's platform independence is a key feature, meaning it can run on any device or operating system through the Java Virtual Machine (JVM), making it highly portable.

Java's syntax is similar to C++, but it is simpler and focuses on readability and ease of use. The rich ecosystem of libraries and frameworks in Java accelerates development and makes it suitable for building large-scale systems. The object-oriented paradigm helps developers organize and modularize code for better maintainability and scalability.

Core Java is used extensively in various domains, including backend systems, Android applications, and embedded systems. It is known for its stability, robustness, and scalability, making it a prime choice for enterprises and developers who need to handle high volumes of transactions or data. Java's security features also make it a trusted language for building secure applications in sectors like finance and healthcare.

3.1.1. Topics Covered in Core Java Module

Introduction to Java Programming

- Understanding Java's history and importance in modern application development.
- Java's platform independence through JVM.
- Structure of a Java program, compiling and running Java code.

Java Syntax and Data Types

- Basic structure of Java code: classes, methods, and variables.
- Primitive data types (int, float, double, boolean, char) and their usage.
- Understanding reference types and objects.

Control Statements and Loops

- Conditional statements: if, else, switch.
- Looping structures: for, while, and do-while loops.
- Using loops for repetitive tasks and conditions for program flow control.

Object-Oriented Programming Concepts

- Classes and Objects: Defining classes, creating objects, constructors.

- Inheritance: Extending classes, method overriding.
- Polymorphism: Method overloading and overriding, dynamic method dispatch.
- Encapsulation: Access modifiers, getters, setters.
- Abstraction: Abstract classes and interfaces.

Exception Handling

- Try-catch block for catching and handling exceptions.
- Custom exceptions and exception propagation.
- Using finally block for clean-up code.
- Handling different types of errors, including runtime and checked exceptions.

Arrays and Collections

- Working with arrays and multi-dimensional arrays.
- Introduction to collections: List, Set, Map (ArrayList, HashMap, etc.).
- Iterating through collections using loops and iterators.

Java I/O (Input/Output)

- Reading from and writing to files using streams.
- File handling using FileInputStream, FileOutputStream, BufferedReader, and BufferedWriter.
- Serialization: Storing and reading object states from files.

Multithreading and Concurrency

- Introduction to threads in Java.
- Creating threads by extending the Thread class or implementing Runnable.
- Synchronization to avoid race conditions.
- Understanding concurrency concepts and thread-safe code.

Java Memory Management

- The heap and stack in memory management.
- Garbage Collection: How Java handles memory management automatically.
- Working with the finalize() method.

3.1.2. Brief Summary

The Core Java module introduces the key concepts of Java programming, focusing on object-oriented programming principles such as inheritance, polymorphism, and encapsulation. It covers fundamental topics such as syntax, data types, operators, and control structures, which form the foundation of Java programming. Exception handling is a vital topic in Java, ensuring that runtime errors are handled gracefully. Core Java also emphasizes memory management, multithreading, and input/output operations, which are essential for building efficient, scalable applications.

3.2. ABOUT MODULE 2

Java Collections and Frameworks

In addition to core Java features, mastering Java's collection framework is essential for handling dynamic data efficiently. The collection framework provides standard data structures like lists, sets, and maps, along with utilities for manipulating them. Java's collections make it easier to work with large datasets, perform complex queries, and manage elements in various ways.

3.2.1. Topics Covered in Java Collections Module

Introduction to Collections Framework

- Overview of the Java Collections Framework and its interface-based architecture.
- Collection, List, Set, Queue, and Map interfaces.
- Benefits of using collections over arrays in dynamic data handling.

List Interface

- Working with ArrayList, LinkedList, and Vector.
- List operations such as adding, removing, and accessing elements by index.
- Sorting lists using Collections.sort() and custom comparators.

Set Interface

- Understanding HashSet, LinkedHashSet, and TreeSet.
- Set operations like add, remove, and check for existence.
- Benefits of using a set (eliminates duplicates).

Map Interface

- Introduction to HashMap, TreeMap, LinkedHashMap.
- Key-value pair mappings and operations like put(), get(), remove().
- Iterating over keys and values using iterators and for-each loops.

Queue Interface

- Working with PriorityQueue, LinkedList (as a Queue), and Deque.
- Queue operations: adding, removing, and peeking elements.
- FIFO (First In, First Out) principles.

Collections Utility Class

- Sorting and reversing collections using utility methods.
- Using Collections class for common tasks like binary search, shuffling, and filling collections.

3.2.2. Brief Summary

The Java Collections Framework is an essential tool for managing groups of objects in a flexible and efficient way. It provides powerful data structures like List, Set, and Map, which help

store and retrieve data in optimized ways. The course covers how to work with these collections and the utility methods available for sorting, searching, and manipulating elements. Mastery of the collections framework is crucial for building applications that can handle large datasets and complex data manipulations.

3.3. ABOUT MODULE 3

Advanced Core Java Concepts

This module expands on the core Java concepts by diving into advanced topics such as generics, lambda expressions, streams, and Java 8 features. These modern features enhance the power and efficiency of Java programs and enable developers to write cleaner, more concise code.

3.3.1. Topics Covered in Advanced Java Module

Generics in Java

- Understanding Java generics for creating type-safe code.
- Working with generic classes, methods, and interfaces.
- Wildcards in generics for more flexible type bounds.

Lambda Expressions

- Introduction to functional programming concepts in Java.
- Writing lambda expressions for concise and expressive code.
- Using lambda expressions with Collection methods (filter, map, reduce).

Streams API

- Working with Java Streams to process collections of objects in a functional style.
- Using map(), filter(), and reduce() methods to perform operations on data.
- Parallel streams for improved performance with large datasets.

Functional Interfaces

- Introduction to functional interfaces and how they enable lambda expressions.
- Commonly used functional interfaces like Predicate, Function, Consumer, and Supplier.

Java 8 New Features

- Default methods and static methods in interfaces.
- Method references in place of lambda expressions.
- Introduction to Optional for handling null values safely.

3.3.2. Brief Summary

The Advanced Core Java module introduces Java 8 features that have revolutionized the way developers write code. Generics provide flexibility and type safety in code, while lambda expressions and the Streams API enable functional programming in Java. These features allow for more concise and efficient code, which is especially useful when dealing with large datasets or complex operations. Understanding these advanced topics is key to mastering Java and staying up-

to-date with modern Java programming practices.

3.4. ABOUT MINI PROJECT: QUIZ APPLICATION

3.4.1. Title of the Project

Quiz Application using Java in Eclipse

3.4.2. Contents

- Business Objective
- Business Constraints
- Project Architecture
- Data Representation
- GUI Design and Navigation
- Lifelines Logic
- Question Structure (Each Question as a Class)
- Application Flow
- Evaluation and Testing
- Deployment

3.4.3. Project Overview and Scope

This project is a desktop-based quiz application built solely using **Java in Eclipse IDE**. It replicates the experience of a quiz game, complete with **interactive GUI** and two popular **lifelines: 50-50 and Skip**. Each question is modeled as an individual class to ensure modularity and clean object-oriented design. The application provides an intuitive interface for users to engage with multiple-choice questions in a structured flow.

3.4.4. Project Architecture

The application follows a **modular architecture** where:

- A **main class** manages GUI rendering, lifeline logic, score tracking, and navigation
- An **interface** defines a template for all questions.
- Each **question** is created as a separate class implementing the interface.
- **Lifelines** are handled with conditional flags to allow single use per game.

3.4.5. Business Problem

In many educational and training environments, there is a need for a lightweight, offline quiz system that doesn't rely on external platforms or databases. This application addresses that need by offering a portable and easy-to-use solution entirely within Java.

3.4.6. Business Objective

- Deliver a user-friendly quiz game using only Java and Eclipse.
- Enable functionality similar to TV quiz shows by including lifelines.
- Use object-oriented programming to separate logic cleanly per question.

3.4.7. Constraints

- No external tools, databases, or frameworks used.
- GUI and logic are built only using **Java and Swing** inside Eclipse.
- Lifelines can only be used once per game session.
- Each question is handled in its own class, with consistent structure.

3.4.8. Data Representation (Question Classes)

- The project uses an **interface** to standardize each question's structure.
- Each question class provides:
    - A question string
    - Four answer options
    - One correct answer

This modular structure makes it easy to add, remove, or modify questions.

3.4.9. GUI and Application Flow

The GUI is built using **Java cmd** and includes the following components:

- A label to display the question
- Four radio buttons for answer options
- Three buttons for:
    - **Next** – Submit and move to the next question
    - **50-50 Lifeline** – Hides two wrong options
    - **Skip Lifeline** – Skips the current question
- Once all questions are answered, the final score is shown via a popup dialog.

Navigation is strictly sequential — users cannot go back to a previous question.

3.4.10. Lifeline Features

Lifeline	Functionality	Limitation
50-50	Hides 2 incorrect options from view	Usable once only
Skip	Automatically skips the current question	Usable once only

3.4.11. Evaluation and Testing

The application has been tested manually by:

- Answering questions with and without lifelines
- Verifying correct answer scoring
- Ensuring lifelines work only once
- Confirming GUI updates as expected across questions
- Validating modular loading of each question class

3.4.12. Deployment

The application is designed to run locally through the Eclipse IDE:

- Executed using **Run As → Java Application**
- No installation or internet access required
- Can optionally be exported as a **runnable JAR file** using Eclipse Export wizard

3.4.13 Code Implementation

```
import java.util.*;
public class QuizApplication {
    static Scanner sc = new Scanner(System.in);
    static int totalEarnings = 0;
    static boolean fiftyFiftyUsed = false, audienceUsed = false, skipUsed = false;
    static boolean fiftyFiftyActive = false;
    static boolean skipActive = false;
    static int[] reducedIndexes = new int[2];
    public static class ConsoleColors {
        public static final String RESET = "\u001B[0m";
        public static final String BOLD = "\u001B[1m";
        public static final String RED = "\u001B[31m";
        public static final String GREEN = "\u001B[32m";
        public static final String YELLOW = "\u001B[33m";
        public static final String BLUE = "\u001B[34m";
        public static final String CYAN = "\u001B[36m";
        public static final String PURPLE = "\u001B[35m";
    }
    public static void main(String[] args) {
        System.out.println(ConsoleColors.CYAN + "Welcome to the Quiz Application!"
+ ConsoleColors.RESET);
        System.out.print("Please enter your name: ");
        String playerName = sc.nextLine();
        System.out.println(ConsoleColors.CYAN +"Hello, " + playerName + "! Are you
ready to start the quiz?"+ConsoleColors.RESET);
        System.out.println(ConsoleColors.RED+"Rules:"+ConsoleColors.RESET+"
Answer correctly to earn money. Use lifelines wisely. Each lifeline can only be used
once.");
```

```
        System.out.print("Do you agree to the rules? "+ConsoleColors.RED+"(yes/no):
"+ConsoleColors.RESET);
        if (!sc.nextLine().equalsIgnoreCase("yes")) {
            System.out.println(ConsoleColors.CYAN+"Thank you! Come back
again."+ConsoleColors.RESET);
            return;
        }

        startQuiz();
    }

    public static void startQuiz() {
        String[][] questions = {
            {"What is the time complexity of binary search?", "O(n)", "O(log n)",
"O(n^2)", "O(1)", "2"},
            {"Which sorting algorithm has the best average case time complexity?",
"Bubble Sort", "Insertion Sort", "Merge Sort", "Selection Sort", "3"},
            {"What is the default value of an int variable in Java?", "0", "1", "null",
"undefined", "1"},
            {"What is the output of 5 & 3 in Java?", "1", "2", "3", "5", "1"},
            {"What is the derivative of sin(x)?", "cos(x)", "tan(x)", "-cos(x)", "-sin(x)",
"1"}
        };

        for (int i = 0; i < questions.length; i++) {
            fiftyFiftyActive = false;
            reducedIndexes = new int[2];
            skipActive = false;
            System.out.println("\n" + ConsoleColors.YELLOW + "Question " + (i + 1) + ":
" + questions[i][0] + ConsoleColors.RESET);
            System.out.println(ConsoleColors.PURPLE + "1. " + questions[i][1] +
ConsoleColors.RESET);
            System.out.println(ConsoleColors.PURPLE + "2. " + questions[i][2] +
ConsoleColors.RESET);
            System.out.println(ConsoleColors.PURPLE + "3. " + questions[i][3] +
```

```
ConsoleColors.RESET);
        System.out.println(ConsoleColors.PURPLE + "4. " + questions[i][4] +
ConsoleColors.RESET);
        boolean answered = false;
        while (!answered) {
            System.out.print("Enter your answer Option Number" +
(areLifelinesAvailable() ? " or type 'lifeline' to use a lifeline: " : ": "));
            String input = sc.nextLine();
            if (isValidOption(input)) {
                if (input.equals(questions[i][5])) {
                    if (!skipActive) {
                        System.out.println(ConsoleColors.GREEN + ConsoleColors.BOLD +
" ✅Correct! You earned ₹1000." + ConsoleColors.RESET);
                        totalEarnings += 1000;
                    } else {
                        System.out.println(ConsoleColors.BLUE + "Correct! (No earnings
since you used Skip Lifeline)." + ConsoleColors.RESET);
                    }
                } else {
                    System.out.println(ConsoleColors.RED + ConsoleColors.BOLD + "+
Wrong answer. The correct answer was: " +
questions[i][Integer.parseInt(questions[i][5])] + ConsoleColors.RESET);
                    if (!skipActive) {
                        System.out.println(ConsoleColors.RED+"Game Over."+
ConsoleColors.GREEN+ "Your total earnings are: ₹" +
totalEarnings+ConsoleColors.RESET);
                        return;
                    } else {
                        System.out.println(ConsoleColors.RED+"Wrong answer! (No
penalty since you used Skip Lifeline)."+ConsoleColors.RESET);
                    }
                }
                answered = true;
            } else if (input.equalsIgnoreCase("lifeline") && areLifelinesAvailable()) {
                if (!showAvailableLifelines(questions[i])) {
```

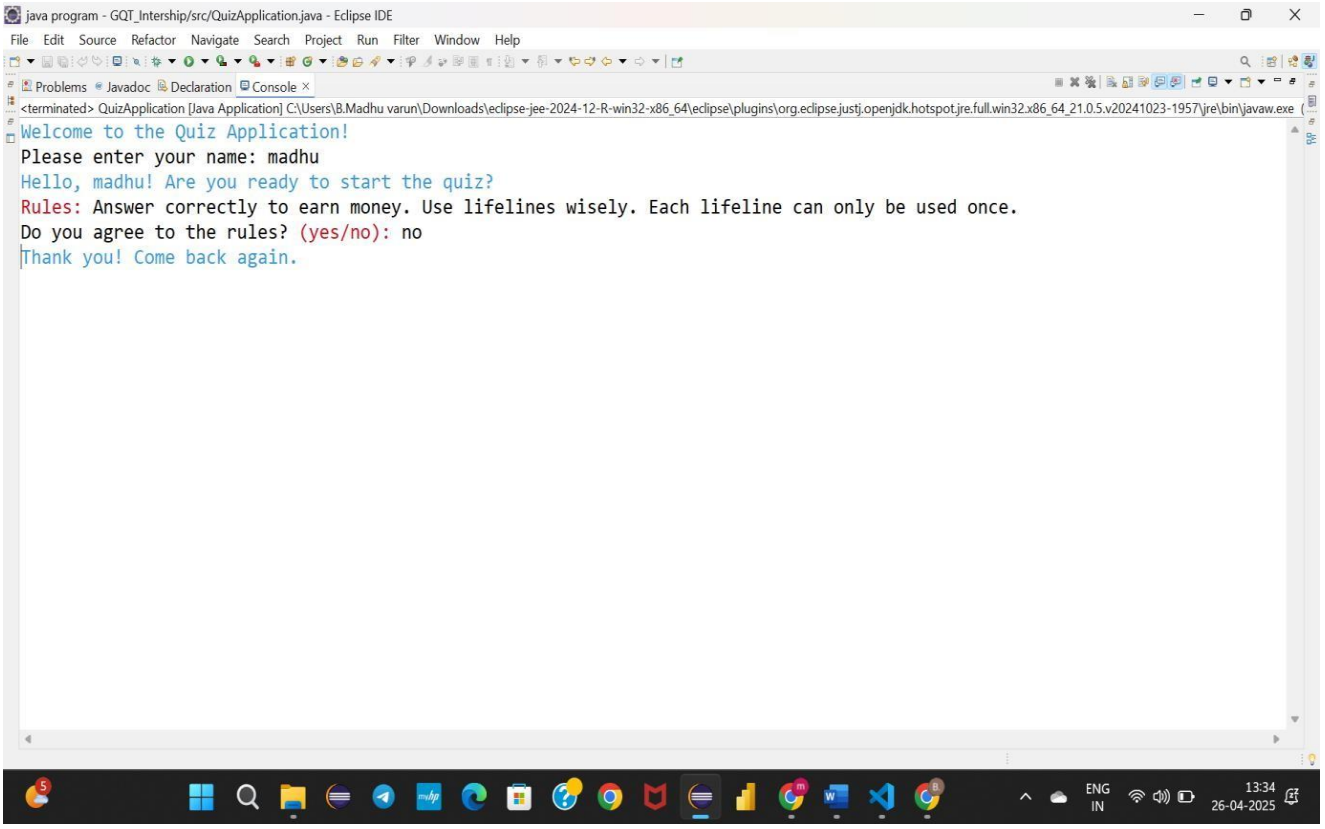
```
                System.out.println(ConsoleColors.RED+"No lifelines are
available!"+ConsoleColors.RESET);
                }
            } else {
                System.out.println(ConsoleColors.RED+"Invalid input! Please enter a
number between 1-4 or 'lifeline'."+ConsoleColors.RESET);
            }
        }
        System.out.print("Do you want to quit? (yes/no): ");
        if (sc.nextLine().equalsIgnoreCase("yes")) {
            System.out.println(ConsoleColors.GREEN+"You decided to quit. Your total
earnings are: ₹" + totalEarnings+ConsoleColors.RESET);
            return;
        }
    }
    System.out.println(ConsoleColors.GREEN+"\nQuiz completed! Your total
earnings are: ₹" + totalEarnings+ConsoleColors.RESET);
  }
  public static boolean showAvailableLifelines(String[] question) {
    boolean lifelineUsed = false;
    System.out.println(ConsoleColors.YELLOW+"\nAvailable
Lifelines:"+ConsoleColors.RESET);
    if (!fiftyFiftyUsed) System.out.println(ConsoleColors.BLUE+"1. 50-
50"+ConsoleColors.RESET);
    if (!audienceUsed) System.out.println(ConsoleColors.BLUE+"2. Audience
Poll"+ConsoleColors.RESET);
    if (!skipUsed) System.out.println(ConsoleColors.BLUE+"3. Skip
Question"+ConsoleColors.RESET);
    System.out.print("Choose a lifeline (enter the number): ");
    String choice = sc.nextLine();
    switch (choice) {
      case "1":
        if (!fiftyFiftyUsed) {
          fiftyFiftyUsed = true;
          fiftyFiftyActive = true;
```

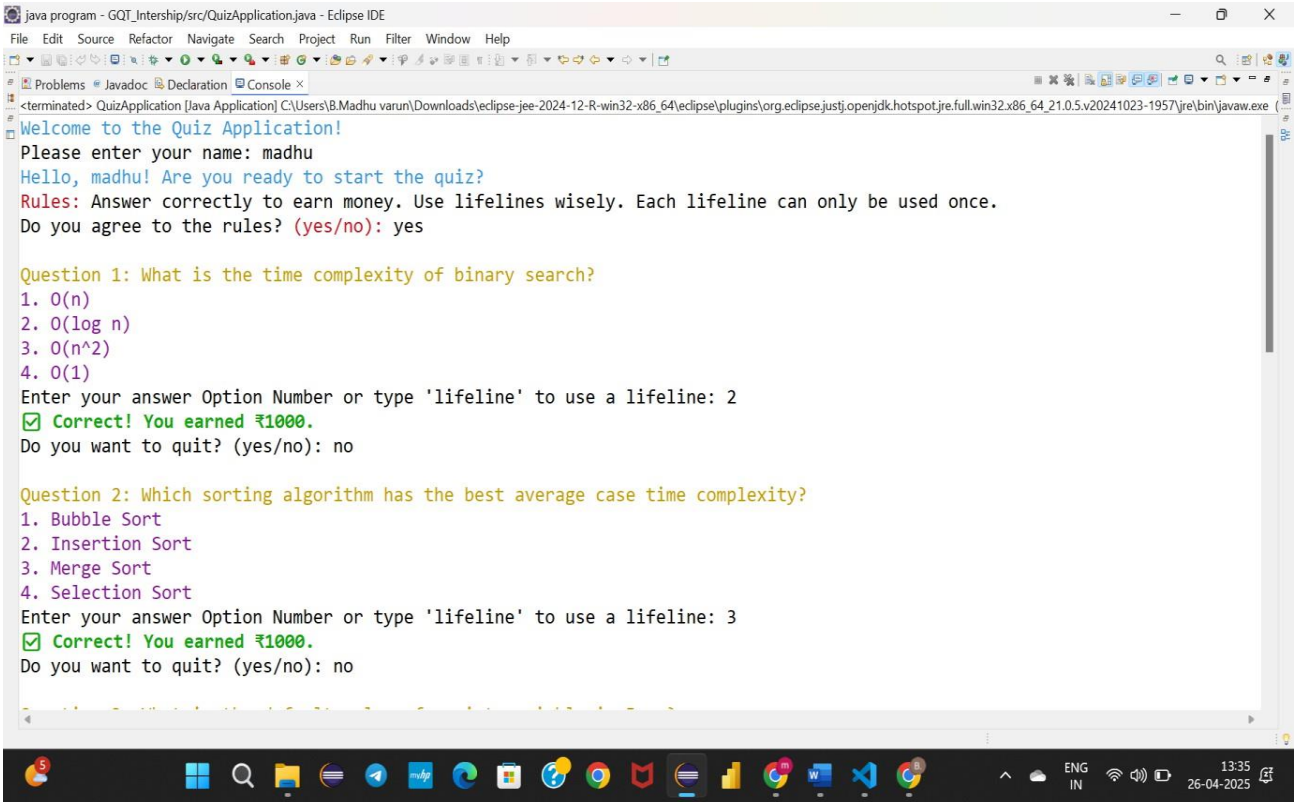
```
                useFiftyFifty(question);
                lifelineUsed = true;
            }
            break;
        case "2":
            if (!audienceUsed) {
                audienceUsed = true;
                useAudiencePoll(question);
                lifelineUsed = true;
            }
            break;
        case "3":
            if (!skipUsed) {
                skipUsed = true;
                skipActive = true;
                System.out.println("You skipped the question, but you still need to answer
it.");
                lifelineUsed = true;
            }
            break;
        default:
            System.out.println(ConsoleColors.RED+"Invalid choice. No lifeline
used."+ConsoleColors.RESET);
            break;
    }
    return lifelineUsed;
  }
  public static void useFiftyFifty(String[] question) {
    System.out.println(ConsoleColors.GREEN+"🔹 50-50 Lifeline Activated! Two
incorrect options are removed."+ConsoleColors.RESET);
    Random rand = new Random();
    int correctOption = Integer.parseInt(question[5]);
    int otherOption;
    do {
      otherOption = rand.nextInt(4) + 1;
```

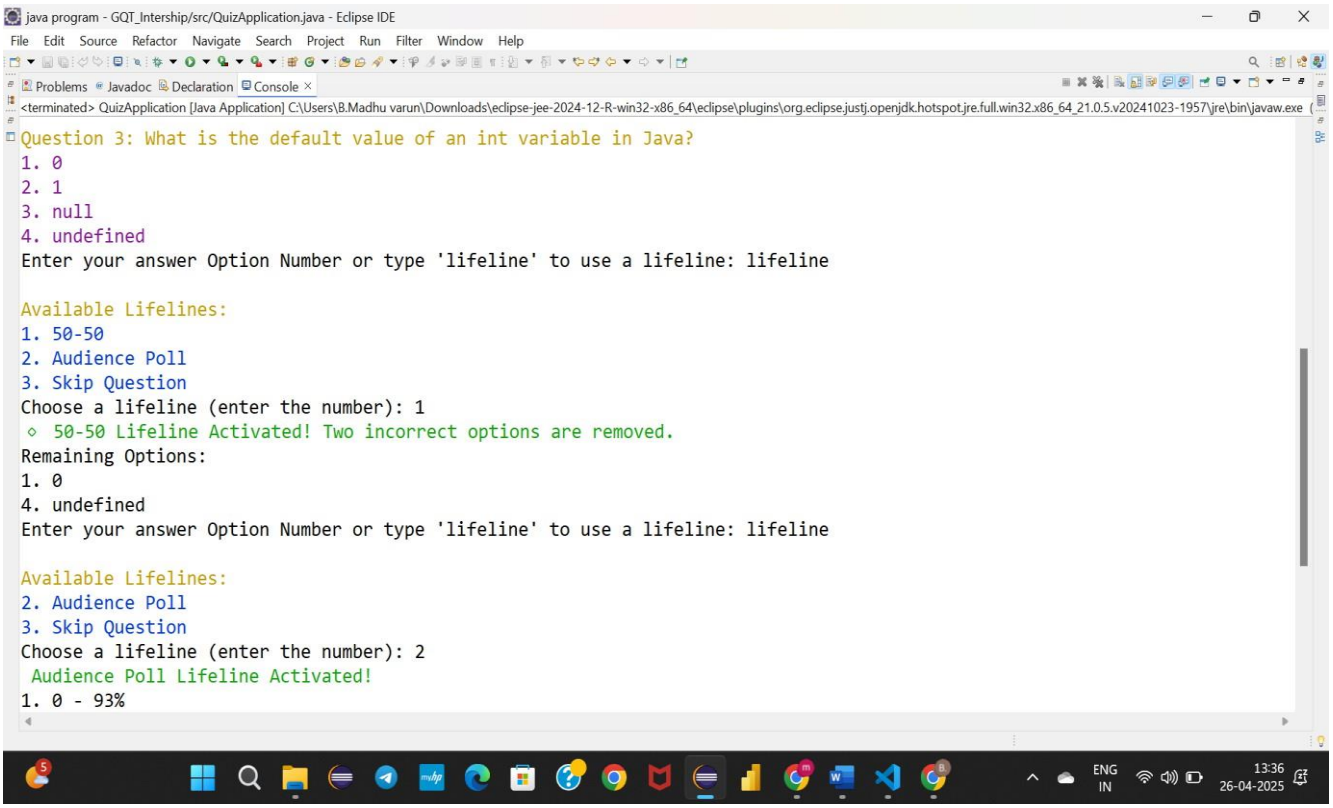
```
        } while (otherOption == correctOption);
        reducedIndexes[0] = correctOption;
        reducedIndexes[1] = otherOption;
        System.out.println("Remaining Options:");
        System.out.println(reducedIndexes[0] + ". " + question[reducedIndexes[0]]);
        System.out.println(reducedIndexes[1] + ". " + question[reducedIndexes[1]]);
    }
    public static void useAudiencePoll(String[] question) {
        System.out.println(ConsoleColors.GREEN+" Audience Poll Lifeline
Activated!"+ConsoleColors.RESET);
        Random rand = new Random();
        int correctOption = Integer.parseInt(question[5]);
        int correctPercentage = rand.nextInt(41) + 60;
        int otherPercentage = 100 - correctPercentage;
        if (fiftyFiftyActive) {
            System.out.println(reducedIndexes[0] + ". " + question[reducedIndexes[0]] + "
- " + correctPercentage + "%");
            System.out.println(reducedIndexes[1] + ". " + question[reducedIndexes[1]] + "
- " + otherPercentage + "%");
        } else {
            int[] percentages = new int[4];
            percentages[correctOption - 1] = correctPercentage;
            int remainingPercentage = 100 - correctPercentage;
            for (int i = 0; i < 4; i++) {
                if (i != correctOption - 1) {
                    percentages[i] = rand.nextInt(remainingPercentage + 1);
                    remainingPercentage -= percentages[i];
                }
            }
            for (int i = 0; i < 4; i++) {
                System.out.println((i + 1) + ". " + question[i + 1] + " - " + percentages[i] +
"%");
            }
        }
    }
```

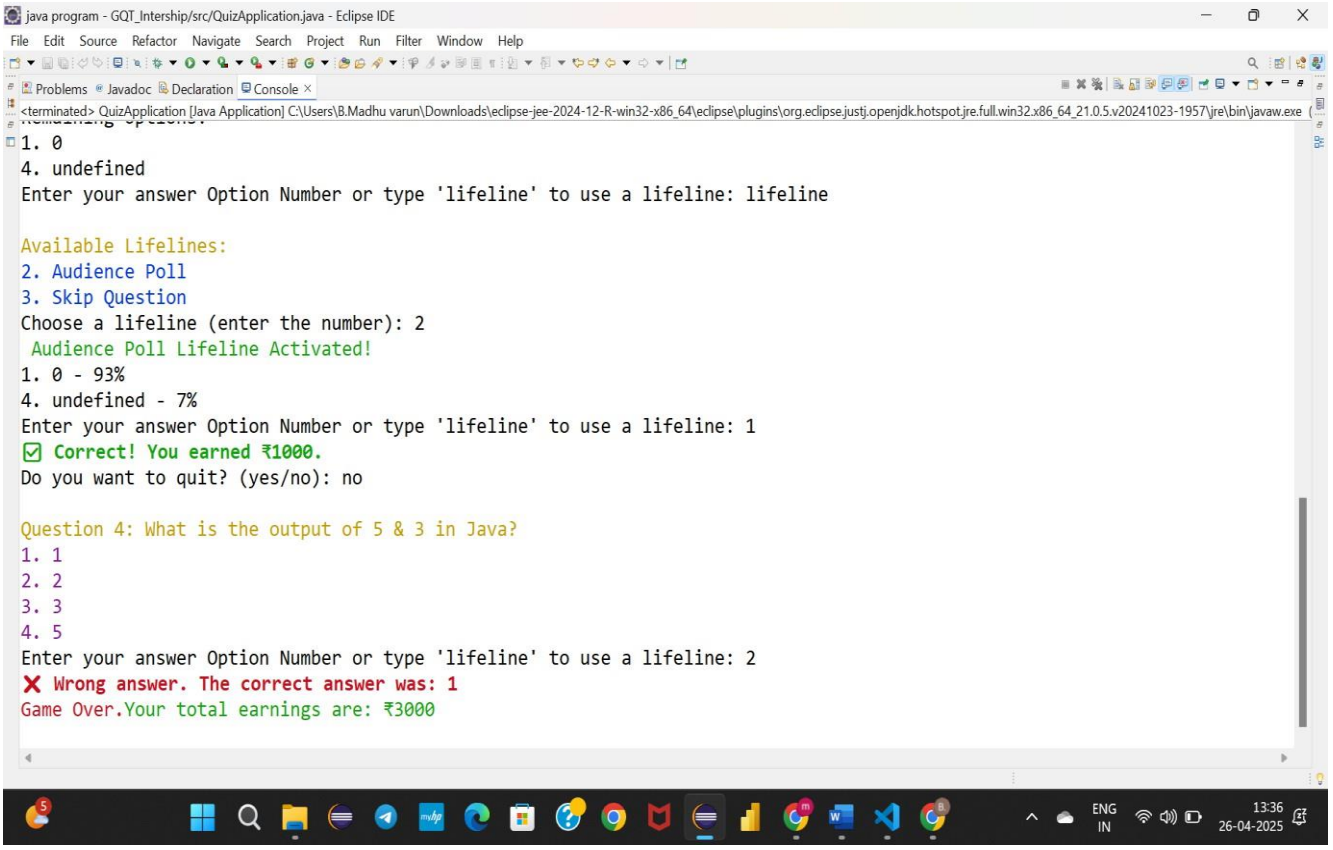
```
    public static boolean areLifelinesAvailable() {
        return !fiftyFiftyUsed || !audienceUsed || !skipUsed;
    }
    public static boolean isValidOption(String input) {
        return input.matches("[1-4]");
    }
}
```

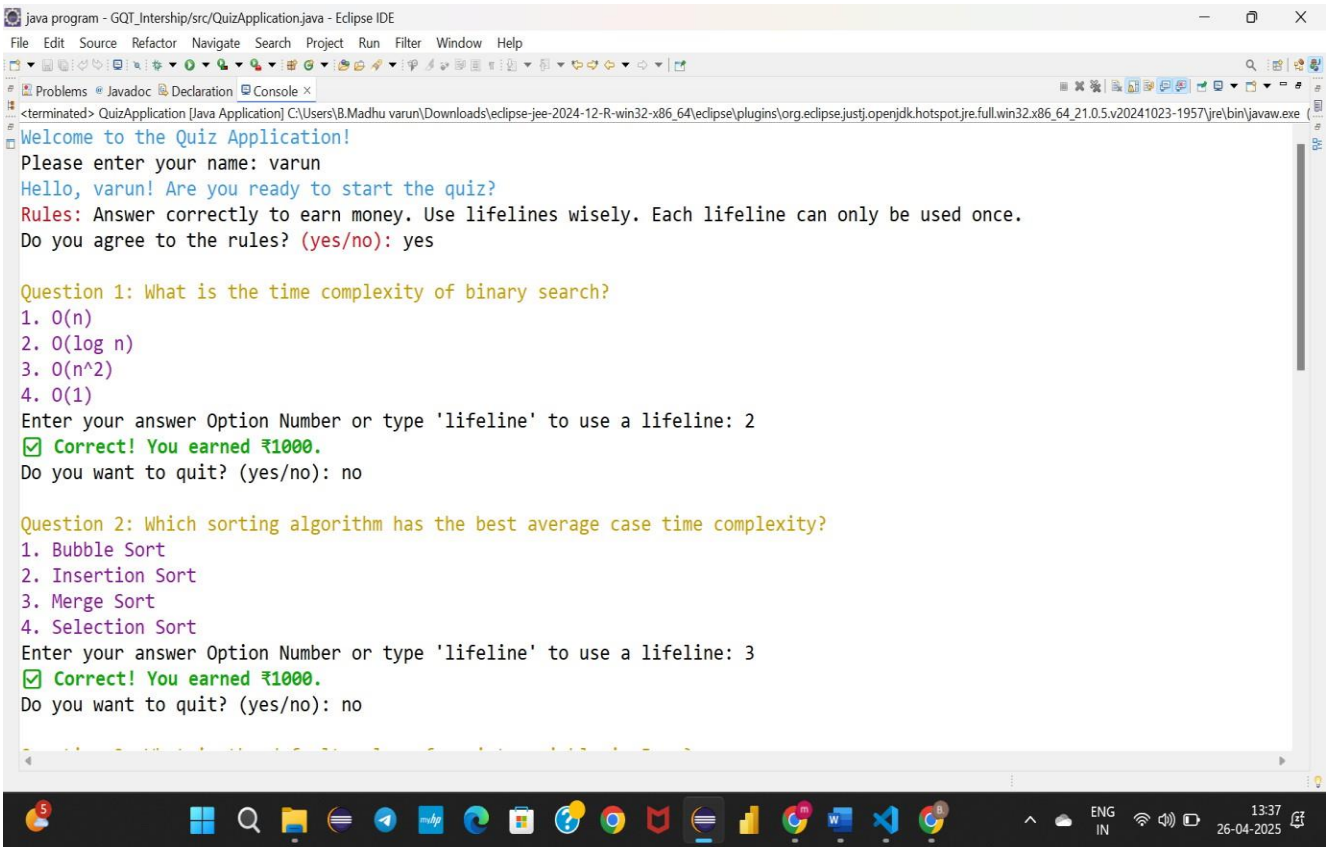
3.4.14. Output /Results Screen

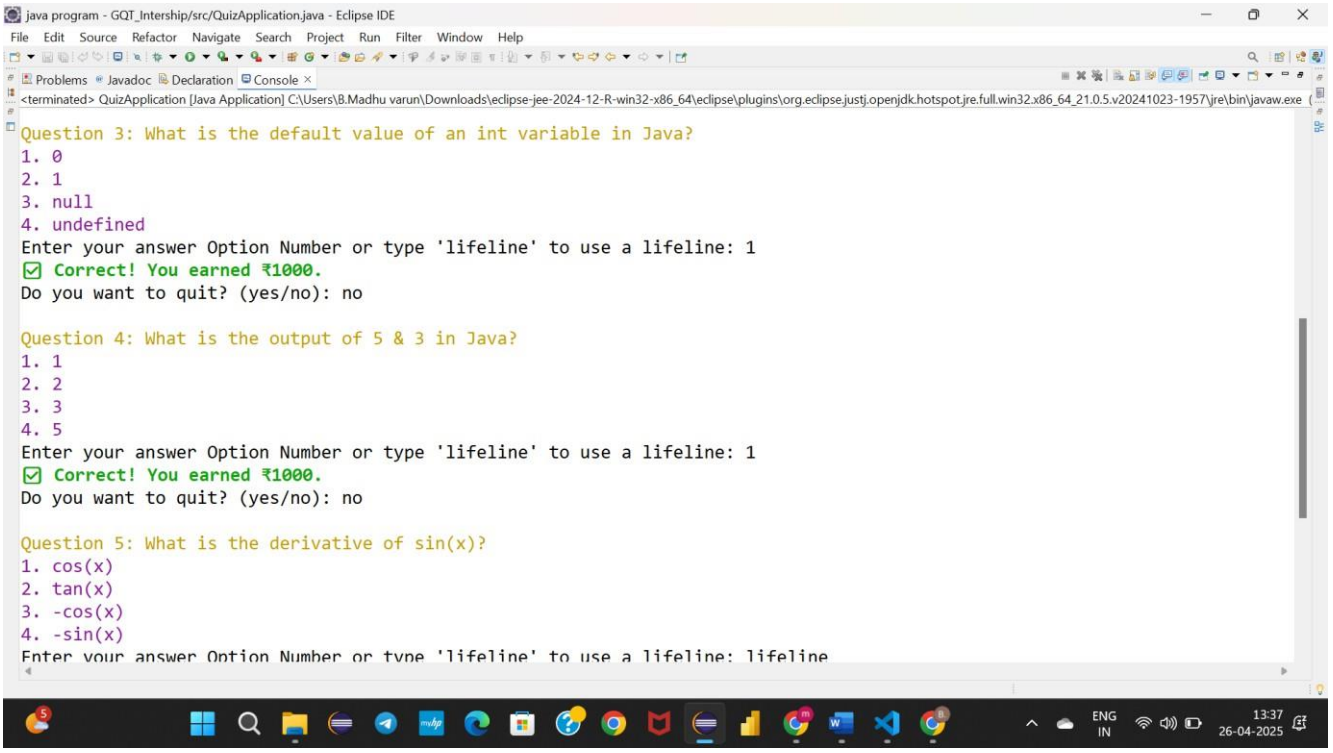


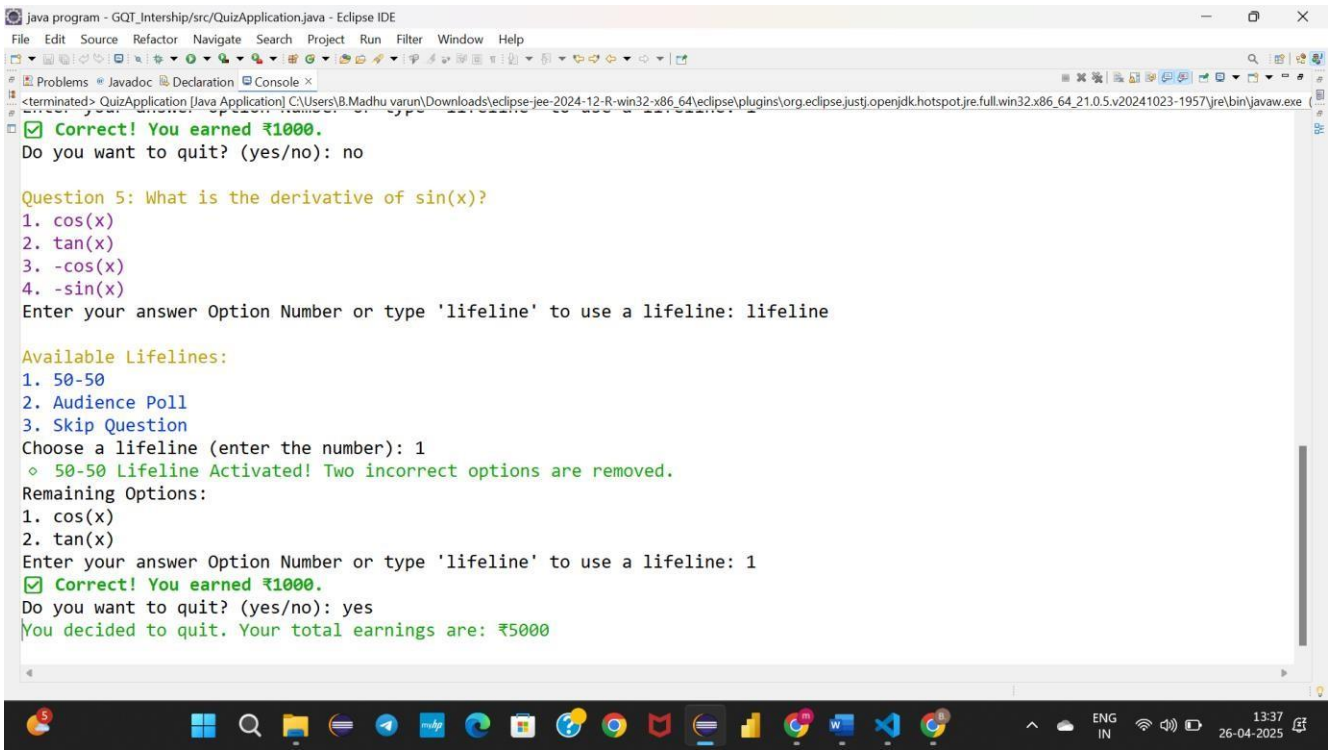



java program - GQT_Intership/src/QuizApplication.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Filter Window Help
Problems Javadoc Declaration Console
<terminated> QuizApplication [Java Application] C:\Users\B.Madhu varun\Downloads\eclipse-jee-2024-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v20241023-1957\jre\bin\javaw.exe
Question 3: What is the default value of an int variable in Java?
1. 0
2. 1
3. null
4. undefined
Enter your answer Option Number or type 'lifeline' to use a lifeline: lifeline
Available Lifelines:
1. 50-50
2. Audience Poll
3. Skip Question
Choose a lifeline (enter the number): 1
◇ 50-50 Lifeline Activated! Two incorrect options are removed.
Remaining Options:
1. 0
4. undefined
Enter your answer Option Number or type 'lifeline' to use a lifeline: lifeline
Available Lifelines:
2. Audience Poll
3. Skip Question
Choose a lifeline (enter the number): 2
Audience Poll Lifeline Activated!
1. 0 - 93%
ENG IN
13:36 26-04-2025


java program - GQT_Intership/src/QuizApplication.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Filter Window Help
Problems Javadoc Declaration Console
<terminated> QuizApplication [Java Application] C:\Users\B.Madhu varun\Downloads\eclipse-jee-2024-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v20241023-1957\jre\bin\javaw.exe
1. 0
4. undefined
Enter your answer Option Number or type 'lifeline' to use a lifeline: lifeline
Available Lifelines:
2. Audience Poll
3. Skip Question
Choose a lifeline (enter the number): 2
Audience Poll Lifeline Activated!
1. 0 - 93%
4. undefined - 7%
Enter your answer Option Number or type 'lifeline' to use a lifeline: 1
☑ Correct! You earned ₹1000.
Do you want to quit? (yes/no): no
Question 4: What is the output of 5 & 3 in Java?
1. 1
2. 2
3. 3
4. 5
Enter your answer Option Number or type 'lifeline' to use a lifeline: 2
✗ Wrong answer. The correct answer was: 1
Game Over.Your total earnings are: ₹3000
ENG IN
13:36 26-04-2025


java program - GQT_Intership/src/QuizApplication.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Filter Window Help
Problems Javadoc Declaration Console
<terminated> QuizApplication [Java Application] C:\Users\B.Madhu varun\Downloads\eclipse-jee-2024-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v20241023-1957\jre\bin\javaw.exe
Welcome to the Quiz Application!
Please enter your name: varun
Hello, varun! Are you ready to start the quiz?
Rules: Answer correctly to earn money. Use lifelines wisely. Each lifeline can only be used once.
Do you agree to the rules? (yes/no): yes
Question 1: What is the time complexity of binary search?
1. O(n)
2. O(log n)
3. O(n^2)
4. O(1)
Enter your answer Option Number or type 'lifeline' to use a lifeline: 2
Correct! You earned ₹1000.
Do you want to quit? (yes/no): no
Question 2: Which sorting algorithm has the best average case time complexity?
1. Bubble Sort
2. Insertion Sort
3. Merge Sort
4. Selection Sort
Enter your answer Option Number or type 'lifeline' to use a lifeline: 3
Correct! You earned ₹1000.
Do you want to quit? (yes/no): no
ENG IN
13:37
26-04-2025


java program - GQT_Intership/src/QuizApplication.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Filter Window Help
Problems Javadoc Declaration Console
<terminated> QuizApplication [Java Application] C:\Users\B.Madhu varun\Downloads\eclipse-jee-2024-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v20241023-1957\jre\bin\javaw.exe
Question 3: What is the default value of an int variable in Java?
1. 0
2. 1
3. null
4. undefined
Enter your answer Option Number or type 'lifeline' to use a lifeline: 1
Correct! You earned ₹1000.
Do you want to quit? (yes/no): no
Question 4: What is the output of 5 & 3 in Java?
1. 1
2. 2
3. 3
4. 5
Enter your answer Option Number or type 'lifeline' to use a lifeline: 1
Correct! You earned ₹1000.
Do you want to quit? (yes/no): no
Question 5: What is the derivative of sin(x)?
1. cos(x)
2. tan(x)
3. -cos(x)
4. -sin(x)
Enter your answer Option Number or type 'lifeline' to use a lifeline: lifeline
ENG IN
13:37
26-04-2025



ABOUT MAJOR PROJECT: Inventory Management System.

3.5. Title of the Project

Inventory Management System (Using Spring Tool Suite And Vs Code.)

3.5.1. Contents

- Business Objective
- Business Constraints
- Project Architecture
- Functionalities
- Data Collection and Details
- Data Preprocessing
- UI/UX Design
- Modeling & Architecture
- Evaluation
- Deployment

3.5.2. Project Overview and Scope

This **IMS** is designed entirely in **Java**, using **Spring & Vs Code** for development. This Inventory Management System is a software solution designed to efficiently manage a business's stock, orders, and related data . Admins can manage the product catalog, inventory, and view orders. It's crucial for maintaining optimal inventory levels, reducing costs, and improving order fulfillment.

The application uses a local database (e.g., MySQL or SQLite) for storing product and user data as well as **JDBC (Java Database Connectivity)** to interact with the database.

Scope:

- **User-side**: Product browsing, cart management, order placement, and user authentication.
- **Admin-side**: Managing products (adding, updating, deleting), viewing orders.
- **Database**: A local MySQL or SQLite database stores all application data, including products, user information, and orders.

3.5.3. Project Architecture

(i) Client & Business Understanding:

The business aim is to create a simplified e-commerce system that is user-friendly, secure, and capable of handling basic e-commerce tasks such as viewing products, adding items to the shopping cart, and placing orders.

(ii) Data Collection:

Data for this project will be managed in a local database, using **JDBC** to communicate between the Java application and the database. The database includes the following tables:

1. **Products**: Contains details like product name, description, price, quantity, and image path.
2. **Users**: Stores user login credentials and personal information.
3. **Orders**: Tracks the user's order history.
4. **Order_Items**: Contains product details of the items included in each order.

(iii) Operational Platform:

- **Frontend**: Vs Code (for GUI)
- **Backend**: RESTful API (Java/Spring Boot)
- **Database**: MySQL or SQLite (local storage)

(iv) User Interface (UI/UX):

The user interface is developed using **Vs Code**, which allows for creating GUI components such as buttons, text fields, and tables. The interface includes:

- A login page.
- Product browsing page with categories.
- Shopping cart page.
- Admin panel to manage products and orders.

3.5.4. Business Problem

IMS often struggle with providing a simple and secure platform to manage products and process orders efficiently. The project solves this problem by providing a basic Inventory platform where users can:

- Browse products.
- Add items to a cart.
- Place orders securely.

Admin users can easily manage product listings and track orders from customers.

3.5.5. Business Objective

The primary objectives of this project are:

1. Enable users to view and purchase products through a simple and interactive interface.
2. Allow admin users to manage product listings, including adding and removing products.
3. Ensure secure transaction processing and efficient order management.

3.5.6. Constraints

- **Business Constraints**:
  - The application is designed as a simple, standalone desktop application.
  - **Cost**: Focus on using open-source technologies (Java, MySQL/SQL) for cost minimization.
- **Technical Constraints**:
  - **JDBC** is used for database connectivity, so the database must be locally hosted (e.g., MySQL or SQLite).
  - **Vs Code** is used for the GUI, meaning the application is built for desktop use, not web- based.
- **Success Criteria**:
  - **Business Success**: A working e-commerce platform with product browsing, cart management, and order placement.
  - **Technical Success**: The application should work with basic product listings, cart handling, and secure order processing.
  - **Economic Success**: A working solution with minimal resources, without additional operational costs.

3.5.7. Functionalities & Features

1. **User Authentication**:
   - **Login**: Users can log in with their credentials (username and password).
   - **Registration**: New users can register by providing basic information (username, password, email).
2. **Product Management (Admin only)**:
   - Admins can **add**, **update**, or **delete** products from the catalog.
   - Each product has details like **name**, **description**, **price**, **quantity**, and **image**.
3. **Product Browsing (User)**:

- Users can browse products, filter by category, and view product details.
- Products are listed in a table format with name, description, and price.

4. **Stock & Supplier Management** :
   - Tracks inventory levels, manage stock updates, and provide alerts.
   - The Maintain supplier information and track supplier performance.
5. **Checkout and Order Placement**:
   - Admins can proceed to checkout, confirm order details, and place an order.
   - Order details are stored in the database, and product quantities are updated accordingly.
6. **Admin Panel**:
   - Admins can view orders placed by customers, and process them.
   - Admins can update product information like price, quantity, and description.
7. **Database Operations**:
   - **JDBC** is used to interact with the MySQL/SQLite database.
   - Database tables store users, products, and orders data.

3.5.8. Data Collection & Understanding

Data used in this project is primarily product-related and user-related, stored in the following tables in the database:

1. **Products Table**:
   - Product Name
   - Product Description
   - Product Price
   - Product Quantity
2. **Users Table**:
   - Username
3. **Orders Table**:
   - Order Total
4. **Order_Items Table**:
   - Product Name
   - Quantity
   - Price per item

3.5.9. Data Preprocessing

1. **User Data**:
   - Ensuring passwords are stored in a **hashed** format using secure hashing algorithms (e.g., SHA-256).
2. **Product Data**:

- Regular updates of product quantities and prices in the database.

3. **Order Data**:
   - When an order is placed, the system updates the **Orders** and **Order_Items** tables.

3.5.10. UI/UX Design

Swing GUI includes basic components like:

- **Text Fields** for input (e.g., username, password, product details).
- list products and orders.
- **Dialogs** for confirmation (e.g., "Order placed successfully").

The UI is designed to be intuitive, with a simple flow for users to navigate through product browsing, cart management, and order placement.

3.5.11. Evaluation

The system is evaluated based on:

1. **Functionality**: Does the system allow users to browse products, manage a cart, and place orders? Can admins add/update/delete products and view orders?
2. **Usability**: Is the user interface intuitive and easy to navigate?
3. **Performance**: Does the system work efficiently with multiple users interacting with the system simultaneously?

3.5.12. Deployment

- The application is **deployed locally** and can be run on any machine with **Java Runtime Environment (JRE)** installed.
- **Deployment steps**:
  1. Compile and package the project using **Spring Tool Suite & Vs Code** .
  2. Install MySQL or SQLite and set up the database tables.
  3. Connect the Java application with the database using **JDBC**.
  4. Run the application and test all functionalities.

3.5.13 Code Implementation

InventoryManagementSystemApplication.java:

```
package com.cts.inventorymanagementsystem;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class InventoryManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(InventoryManagementSystemApplication.class, args);  }

 }
```

ModelMapperConfig.java

```
package com.cts.inventorymanagementsystem.config;
import org.modelmapper.ModelMapper;
import org.modelmapper.convention.MatchingStrategies;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class ModelMapperConfig {
@Bean
public ModelMapper modelMapper(){
ModelMapper modelMapper = new ModelMapper();
modelMapper.getConfiguration()
.setFieldMatchingEnabled(true)
.setFieldAccessLevel(org.modelmapper.config.Configuration.AccessLevel.PRIVATE)
.setMatchingStrategy(MatchingStrategies.STANDARD);
return modelMapper;
}
}
```

Entities:

Product.java

```
package com.cts.inventorymanagementsystem.entity;
import jakarta.persistence.*;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Positive;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.Base64;
import java.util.List;
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table(name = "products")
public class Product {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
@NotBlank(message = "Name is required")
private String name;
@NotBlank(message = "Sku is required")
@Column(unique = true)
private String sku;
@Positive(message = "Product price msut be a positive value")
private BigDecimal price;
@Min(value = 0, message = "Stock quantity cannot be lesser than zero")
private Integer stockQuantity;
private String description;

@Lob
@Column(name = "image_data", columnDefinition = "LONGBLOB")
private byte[] imageData;
private String imageName;
private String imageType;
private LocalDateTime expiryDate;
private  LocalDateTime updatedAt;
private final LocalDateTime createdAt = LocalDateTime.now();
@ManyToOne
@JoinColumn(name = "category_id")
private Category category;
@OneToMany(mappedBy = "product", cascade = CascadeType.ALL, orphanRemoval = true)
private List<Transaction> transactions;
public String getImageUrl() {
if (this.imageData != null && this.imageType != null) {
return "data:" + this.imageType + ";base64," + Base64.getEncoder().encodeToString(this.imageData);
}
return null; // Or a default image URL
}
@Override
public String toString() {
return "Product{" +
"id=" + id +
", name='" + name + '\'' +
", sku='" + sku + '\'' +
", price=" + price +
", stockQuantity=" + stockQuantity +
```

```
", description='" + description + '\'' +
", imageName='" + imageName + '\'' + // Include image name
", imageType='" + imageType + '\'' + // Include image type
", expiryDate=" + expiryDate +
", updatedAt=" + updatedAt +
", createdAt=" + createdAt +
'}';
}
}
```

Category.java

```
package com.cts.inventorymanagementsystem.entity;
import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table(name = "categories")
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;
    @NotBlank(message = "Name is required")
    @Column(unique = true)
    private String name;
    @OneToMany(mappedBy = "category", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Product> products;
    @Override
    public String toString() {
        return "Category{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

Supplier.java

```
package com.cts.inventorymanagementsystem.entity;
import java.util.List;
import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
```

```
import lombok.NoArgsConstructor;
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table(name = "suppliers")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotBlank(message = "Name is required")
    private String name;
    private String address;
    @OneToMany(mappedBy = "supplier", fetch = FetchType.LAZY, cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<Transaction> transaction;
}
```

Transaction.java

```
package com.cts.inventorymanagementsystem.entity;
import com.cts.inventorymanagementsystem.enums.TransactionStatus;
import com.cts.inventorymanagementsystem.enums.TransactionType;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.math.BigDecimal;
import java.time.LocalDateTime;
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table(name = "transactions")
public class Transaction {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Integer totalProducts;
    private BigDecimal totalPrice;
    @Enumerated(EnumType.STRING)
    private TransactionType transactionType;
    @Enumerated(EnumType.STRING)
    private TransactionStatus status;

    private String description;
    private LocalDateTime updatedAt;
    private final LocalDateTime createdAt = LocalDateTime.now();
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private User user;
    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "product_id")
    private Product product;
    @ManyToOne(fetch = FetchType.LAZY)
```

```
    @JoinColumn(name = "supplier_id")
    private Supplier supplier;
    @Override
    public String toString() {
        return "Transaction{" +
                "id=" + id +
                ", totalProducts=" + totalProducts +
                ", totalPrice=" + totalPrice +
                ", transactionType=" + transactionType +
                ", status=" + status +
                ", description='" + description + '\'' +
                ", updatedAt=" + updatedAt +
                ", createdAt=" + createdAt +
                '}';
    }
}
```

DTO Classes:

ProductDTO.java

```
package com.cts.inventorymanagementsystem.dto;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonIgnoreProperties(ignoreUnknown = true)
@Builder
public class ProductDTO {
    private Long id;
    private Long productId;
    private Long categoryId;
    private Long supplierId;
    private String name;
    private String sku;
    private BigDecimal price;
    private Integer stockQuantity;
    private String description;
    private String imageName;
    private String imageType;
private byte[] imageData;
private LocalDateTime expiryDate;
private  LocalDateTime updatedAt;
private LocalDateTime createdAt;
}
```

CategoryDTO.java

```
package com.cts.inventorymanagementsystem.dto;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonInclude;
import jakarta.validation.constraints.NotBlank;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonIgnoreProperties(ignoreUnknown = true)
public class CategoryDTO {
   private Long id;
   @NotBlank(message = "Name is required")
   private String name;
}
```

SupplierDTO.java

```
package com.cts.inventorymanagementsystem.dto;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonInclude;
import jakarta.validation.constraints.NotBlank;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonIgnoreProperties(ignoreUnknown = true)
public class SupplierDTO {
   private Long id;
   @NotBlank(message = "Name is required")
   private String name;
   private String address;
}
```

TransactionDTO.java

```
package com.cts.inventorymanagementsystem.dto;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import com.cts.inventorymanagementsystem.enums.TransactionStatus;
import com.cts.inventorymanagementsystem.enums.TransactionType;
import com.fasterxml.jackson.annotation.
JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonIgnoreProperties(ignoreUnknown = true)
public class TransactionDTO {
   private Long id;
   private Integer totalProducts;
   private BigDecimal totalPrice;
   private TransactionType transactionType;
   private TransactionStatus status;
```

```
    private String description;
    private LocalDateTime updatedAt;
    private LocalDateTime createdAt;
    private UserDTO user;
    private ProductDTO product;
    private SupplierDTO supplier;
}
```

TransactionRequest.java

```
package com.cts.inventorymanagementsystem.dto;
import com.fasterxml.jackson.annotation.
JsonIgnoreProperties;
import jakarta.validation.constraints.Positive;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@AllArgsConstructor
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class TransactionRequest {
    @Positive(message = "Product id is required")
    private Long productId;
    @Positive(message = "Quantity id is required")
    private Integer quantity;
    private Long supplierId;
    private String  description;
}
```

Enums:

TransactionStatus.java

```
package com.cts.inventorymanagementsystem.enums;
public enum TransactionStatus {
   PENDING, PROCESSING, COMPLETED, CANCELED
}
```

TransactionType.java

```
package com.cts.inventorymanagementsystem.
enums;
public enum TransactionType {
   PURCHASE, SALE, RETURN_TO_SUPPLIER }
```

Controller Classes:

ProductController.java

```
package com.cts.inventorymanagementsystem.controller;
import com.cts.inventorymanagementsystem.dto.ProductDTO;
import com.cts.inventorymanaemntsystem.dto.Response;
import com.cts.inventorymanagementsystem.service.ProductService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import java.math.BigDecimal;
@RestController
```

```
@RequestMapping("/api/products")
@RequiredArgsConstructor
@Slf4j
@CrossOrigin(origins = "*")
public class ProductController {
   private final ProductService productService;
   @PostMapping("/add")
   @PreAuthorize("hasAuthority('ADMIN')")
   public ResponseEntity<Response> saveProduct(
           @RequestParam("imageFile") MultipartFile imageFile,
           @RequestParam("name") String  name,
           @RequestParam("sku") String  sku,
           @RequestParam("price") BigDecimal price,
           @RequestParam("stockQuantity") Integer  stockQuantity,
           @RequestParam("categoryId") Long  categoryId,
           @RequestParam(value = "description", required = false) String  description
           ) {
           log.info("Received request to save product: name = {}, price = {}, stock = {}",
name,price,stockQuantity);
           ProductDTO productDTO = new ProductDTO();
       productDTO.setName(name);
       productDTO.setSku(sku);
       productDTO.setPrice(price);
    productDTO.setStockQuantity(stockQuantity);
       productDTO.setCategoryId(categoryId);
       productDTO.setDescription(description);
       System.out.println(productDTO);
      Response savedProduct = productService.saveProduct(productDTO, imageFile);
      log.info("Product saved successfully");
       return ResponseEntity.ok(savedProduct);
   }
   @PutMapping("/update")
   @PreAuthorize("hasAuthority('ADMIN')")
   public ResponseEntity<Response> updateProduct(
           @RequestParam(value = "imageFile", required=false) MultipartFile imageFile,
           @RequestParam(value = "name",required = false) String  name,
           @RequestParam(value = "sku",required = false) String  sku,
           @RequestParam(value = "price",required = false) BigDecimal price,
           @RequestParam(value = "stockQuantity",required = false) Integer  stockQuantity,
           @RequestParam(value = "productId",required = true) Long  productId,
           @RequestParam(value = "categoryId",required = false) Long  categoryId,
           @RequestParam(value = "description", required = false) String  description
   ) {
           log.info("Received request to update product ID: {}", productId);
       ProductDTO productDTO = new ProductDTO();
   productDTO.setName(name);
   productDTO.setSku(sku);
   productDTO.setPrice(price);
   productDTO.setStockQuantity(stockQuantity);
   productDTO.setCategoryId(categoryId);
   productDTO.setProductId(productId);
   productDTO.setDescription(description);
```

```
        Response updatedProduct = productService.updateProduct(productDTO, imageFile);
        log.info("Product updated successfully for product with ID: {}",productId);
        return ResponseEntity.ok(updatedProduct);
    }
    @GetMapping("/all")
    public ResponseEntity<Response> getAllProducts() {
            log.info("Fetching all products");
            Response products = productService.getAllProducts();
            log.info("Products retrieved successfully");
        return ResponseEntity.ok(products);
    }
    @GetMapping("/{id}")
    public ResponseEntity<Response> getProductById(@PathVariable Long id) {
            log.info("Fetching product by ID: {}",id);
            Response product = productService.getProductById(id);
            log.info("Product retrieved successfully");
            return ResponseEntity.ok(product);
    }
    @DeleteMapping("/delete/{id}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> deleteProduct(@PathVariable Long id) {
            log.info("Received request to delete product ID: {}", id);
            Response deletedProduct =  productService.deleteProduct(id);
            log.info("Product deleted successfully with ID: {}",id);
        return ResponseEntity.ok(deletedProduct);
    }
}
```

CategoryController.java

```
package com.cts.inventorymanagementsystem.controller;

import com.cts.inventorymanagementsystem.dto.CategoryDTO;
import com.cts.inventorymanagementsystem.dto.Response;
import com.cts.inventorymanagementsystem.service.CategoryService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/categories")
@RequiredArgsConstructor
public class CategoryController {

    private final CategoryService categoryService;


    @PostMapping("/add")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> createCategory(@RequestBody @Valid CategoryDTO categoryDTO) {
        return ResponseEntity.ok(categoryService.createCategory(categoryDTO));
    }

    @GetMapping("/all")
```

```
    public ResponseEntity<Response> getAllCategories() {
        return ResponseEntity.ok(categoryService.getAllCategories());
    }
    @GetMapping("/{id}")
    public ResponseEntity<Response> getCategoryById(@PathVariable Long id) {
        return ResponseEntity.ok(categoryService.getCategoryById(id));
    }
    @PutMapping("/update/{id}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> updateCategory(@PathVariable Long id, @RequestBody @Valid
CategoryDTO categoryDTO) {
        return ResponseEntity.ok(categoryService.updateCategory(id, categoryDTO));
    }

    @DeleteMapping("/delete/{id}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> deleteCategory(@PathVariable Long id) {
        return ResponseEntity.ok(categoryService.deleteCategory(id));
    }

}
```

SupplierController.java

```
package com.cts.inventorymanagementsystem.controller;
import com.cts.inventorymanagementsystem.dto.Response;
import com.cts.inventorymanagementstem.dto.SupplierDTO;
import com.cts.inventorymanagementsystem.service.SupplierService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/suppliers")
@RequiredArgsConstructor
public class SupplierController {
    private final SupplierService supplierService;
    @PostMapping("/add")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> addSupplier(@RequestBody @Valid SupplierDTO supplierDTO) {
        return ResponseEntity.ok(supplierService.addSupplier(supplierDTO));
    }

    @GetMapping("/all")
    public ResponseEntity<Response> getAllSuppliers() {
        return ResponseEntity.ok(supplierService.getAllSuppliers());
    }
    @GetMapping("/{id}")
    public ResponseEntity<Response> getSupplierById(@PathVariable Long id) {
        return ResponseEntity.ok(supplierService.getSupplierById(id));
    }
    @PutMapping("/update/{id}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> updateSupplier(@PathVariable Long id, @RequestBody @Valid
SupplierDTO supplierDTO) {
        return ResponseEntity.ok(supplierService.updateSupplier(id, supplierDTO));
```

```
    }

    @DeleteMapping("/delete/{id}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> deleteSupplier(@PathVariable Long id) {
        return ResponseEntity.ok(supplierSevice.deleteSupplier(id));
    }
}
```

TransactionController.java

```
package com.cts.inventorymanagementsystem.controller;
import com.cts.inventorymanagementsystem.dto.Response;
import com.cts.inventorymanagementsystem.dto.TransactionRequest;
import com.cts.inventorymanagementsystem.enums.TransactionStatus;
import com.cts.inventorymanagementsystem.service.TransactionService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/transactions")
@RequiredArgsConstructor
public class TransactionController {

    private final TransactionService transactionService;


    @PostMapping("/purchase")
    public ResponseEntity<Response> restockInventory(@RequestBody @Valid TransactionRequest
transactionRequest) {
        return ResponseEntity.ok(transactionService.restockInventory(transactionRequest));
    }
    @PostMapping("/sell")
    public ResponseEntity<Response> sell(@RequestBody @Valid TransactionRequest transactionRequest) {
        return ResponseEntity.ok(transactionService.sell(transactionRequest));
    }
    @PostMapping("/return")
    public ResponseEntity<Response> returnToSupplier(@RequestBody @Valid TransactionRequest
transactionRequest) {
        return ResponseEntity.ok(transactionService.returnToSupplier(transactionRequest));
    }

    @GetMapping("/all")
    public ResponseEntity<Response> getAllTransactions(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "1000") int size,
            @RequestParam(required = false) String searchText
    ) {
        return ResponseEntity.ok(transactionService.getAllTransactions(page, size, searchText));
    }

    @GetMapping("/{id}")
    public ResponseEntity<Response> getTransactionById(@PathVariable Long id) {
        return ResponseEntity.ok(transactionService.getTransactionById(id));
    }
```

```
    @GetMapping("/by-month-year")
    public ResponseEntity<Response> getAllTransactionByMonthAndYear(
            @RequestParam int month,
            @RequestParam int year
    ) {
        return ResponseEntity.ok(transactionService.getAllTransactionByMonthAndYear(month, year));
    }

    @PutMapping("/update/{transactionId}")
    public ResponseEntity<Response> updateTransactionStatus(
            @PathVariable Long transactionId,
            @RequestBody @Valid TransactionStatus status) {
        System.out.println("ID IS: " + transactionId);
        System.out.println("Status IS: " + status);
        return ResponseEntity.ok(transactionService.updateTransactionStatus(transactionId, status));
    }
}
```

Service Classes:

ProductServiceImpl.java

```
package com.cts.inventorymanagementsystem.service.impl;
s
import com.cts.inventorymanagementsystem.dto.ProductDTO;

import com.cts.inventorymanagementsystem.dto.Response;
import com.cts.inventorymanagementsystem.entity.Category;
import com.cts.inventorymanagementsystem.entity.Product;
import com.cts.inventorymanagementsystem.exceptions.NotFoundException;
import com.cts.inventorymanagementsystem.repository.CategoryRepository;
import com.cts.inventorymanagementsystem.repository.ProductRepository;
import com.cts.inventorymanagementsystem.service.ProductService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.modelmapper.ModelMapper;
import org.modelmapper.TypeToken;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.File;
import java.io.IOException;
import java.math.BigDecimal;
import java.util.List;
import java.util.UUID;
import java.util.stream.Collectors;

@Service
@Slf4j
@RequiredArgsConstructor
public class ProductServiceImpl implements ProductService {

    private final ProductRepository productRepository;
    private final ModelMapper modelMapper;
    private final CategoryRepository categoryRepository;

//    private static final String IMAGE_DIRECTORY = System.getProperty("user.dir") + "/product-image/";
```

```
   //AFTER YOUR FROTEND IS SET UP WROTE THIS SO THE IMAGE IS SAVED IN YOUR FRONTEND PUBLIC FOLDER
//    private static final String IMAGE_DIRECTOR_FRONTEND = "C:/Users/2389403/OneDrive - Cognizant/Desktop/IMS/InventoryManagementSystemFrontend/public/products/";


   @Override
   public Response saveProduct(ProductDTO productDTO, MultipartFile imageFile) {

      Category category = categoryRepository.findById(productDTO.getCategoryId())
            .orElseThrow(()-> new NotFoundException("Category Not Found"));

      //map out product dto to product entity
      Product productToSave = Product.builder()
            .name(productDTO.getName())
            .sku(productDTO.getSku())
            .price(productDTO.getPrice())
            .stockQuantity(productDTO.getStockQuantity())
            .description(productDTO.getDescription())
            .category(category)
            .build();

      if (imageFile != null && !imageFile.isEmpty()) {
          try {
                    byte[] imageData = imageFile.getBytes();
                   log.info("Image data size: {} bytes", imageData.length);  // Add this
                   productToSave.setImageData(imageData);
                   productToSave.setImageName(imageFile.getOriginalFilename());
                   String imageType = imageFile.getContentType();
                   log.info("Image content type: {}", imageType);       // Add this
                   productToSave.setImageType(imageType);
      } catch (IOException e) {
         log.error("Error processing image", e);  // Log the exception
         throw new IllegalArgumentException("Error occurred while processing image: " + e.getMessage());
      }
      }

      productRepository.save(productToSave);
      return Response.builder()
            .status(200)
            .message("Product successfully saved")
            .build();
   }

   @Override
   public Response updateProduct(ProductDTO productDTO, MultipartFile imageFile) {
      Product existingProduct = productRepository.findById(productDTO.getProductId())
            .orElseThrow(() -> new NotFoundException("Product Not Found"));

      //check if image is associated with the update request
      if (imageFile != null && !imageFile.isEmpty()) {
         try {
            byte[] imageData = imageFile.getBytes();
            existingProduct.setImageData(imageData);
            existingProduct.setImageName(imageFile.getOriginalFilename());
            existingProduct.setImageType(imageFile.getContentType());
         } catch (IOException e) {
```

```
                throw new IllegalArgumentException("Error occurred while processing the image" + e.getMessage());
            }
        }
        //Check if category is to be changed for the product
        if (productDTO.getCategoryId() != null && productDTO.getCategoryId() > 0) {
            Category category = categoryRepository.findById(productDTO.getCategoryId())
                    .orElseThrow(() -> new NotFoundException("Category Not Found"));
            existingProduct.setCategory(category);
        }

        //check and update fields
        if (productDTO.getName() != null && !productDTO.getName().isBlank()) {
            existingProduct.setName(productDTO.getName());
        }

        if (productDTO.getSku() != null && !productDTO.getSku().isBlank()) {
            existingProduct.setSku(productDTO.getSku());
        }

        if (productDTO.getDescription() != null && !productDTO.getDescription().isBlank()) {
            existingProduct.setDescription(productDTO.getDescription());
        }

        if (productDTO.getPrice() != null && productDTO.getPrice().compareTo(BigDecimal.ZERO) >= 0) {
            existingProduct.setPrice(productDTO.getPrice());
        }

        if (productDTO.getStockQuantity() != null && productDTO.getStockQuantity() >= 0) {
            existingProduct.setStockQuantity(productDTO.getStockQuantity());
        }

        //Update the product
        productRepository.save(existingProduct);
        return Response.builder()
                .status(200)
                .message("Product successfully Updated")
                .build();

    }

    @Override
    public Response getAllProducts() {
        List<Product> products = productRepository.findAll(Sort.by(Sort.Direction.DESC, "id"));
        //  Map the Product entities to ProductDTOs.  Crucially, include image data.
        List<ProductDTO> productDTOS = products.stream()
                .map(product -> {
                    ProductDTO dto = modelMapper.map(product, ProductDTO.class);
                    dto.setImageName(product.getImageName());  // Map image name
                    dto.setImageType(product.getImageType());  // Map image type
                    dto.setImageData(product.getImageData());
                    return dto;
                })
                .collect(Collectors.toList());
//        System.out.println(productDTOS);

        return Response.builder()
                .status(200)
```

```
                .message("success")
                .products(productDTOS)
                .build();
    }

    @Override
    public Response getProductById(Long id) {
        Product product = productRepository.findById(id)
                .orElseThrow(() -> new NotFoundException("Product Not Found"));

        ProductDTO productDTO = modelMapper.map(product, ProductDTO.class);
        productDTO.setImageName(product.getImageName()); // set image name
        productDTO.setImageType(product.getImageType());// set image type
        productDTO.setImageData(product.getImageData());
        return Response.builder()
                .status(200)
                .message("success")
                .product(productDTO)
                .build();
    }

    @Override
    public Response deleteProduct(Long id) {
        Product product = productRepository.findById(id)
                .orElseThrow(() -> new NotFoundException("Product Not Found"));
        System.out.println(product.getTransactions());
        productRepository.deleteById(id);

        return Response.builder()
                .status(200)
                .message("Product successfully deleted")
                .build();
    }
}
```

CategoryServiceImpl.java

```
package com.cts.inventorymanagementsystem.service.impl;
import com.cts.inventorymagementsystem.dto.CategoryDTO;
import com.cts.inventorymanagementsystem.dto.Response;
import com.cts.inventorymanagementsystem.entity.Category;
import com.cts.inventorymanagementsystem.exceptions.NotFoundException;
import com.cts.inventorymanagementsystem.repository.CategoryRepository;
import com.cts.inventorymanagementsystem.service.CategoryService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.modelmapper.ModelMapper;
import org.modelmapper.TypeToken;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class CategoryServiceImpl implements CategoryService {
```

```
    private final CategoryRepository categoryRepository;
    private final ModelMapper modelMapper;

    @Override
    public Response createCategory(CategoryDTO categoryDTO) {
        Category categoryToSave = modelMapper.map(categoryDTO, Category.class);
        categoryRepository.save(categoryToSave);

        return Response.builder()
                .status(200)
                .message("Category created successfully")
                .build();
    }

    @Override
    public Response getAllCategories() {

        List<Category> categories = categoryRepository.findAll(Sort.by(Sort.Direction.DESC, "id"));

        List<CategoryDTO> categoryDTOS = modelMapper.map(categories, new
TypeToken<List<CategoryDTO>>() {}.getType());

        return Response.builder()
                .status(200)
                .message("success")
                .categories(categoryDTOS)
                .build();
    }

    @Override
    public Response getCategoryById(Long id) {

        Category category = categoryRepository.findById(id)
                .orElseThrow(()-> new NotFoundException("Category Not Found"));
        CategoryDTO categoryDTO = modelMapper.map(category, CategoryDTO.class);

        return Response.builder()
                .status(200)
                .message("success")
                .category(categoryDTO)
                .build();
    }

    @Override
    public Response updateCategory(Long id, CategoryDTO categoryDTO) {

        Category existingCategory = categoryRepository.findById(id)
                .orElseThrow(()-> new NotFoundException("Category Not Found"));

        existingCategory.setName(categoryDTO.getName());
        categoryRepository.save(existingCategory);

        return Response.builder()
                .status(200)
                .message("Category Successfully Updated")
                .build();
```

```
    }

    @Override
    public Response deleteCategory(Long id) {

        categoryRepository.findById(id)
                .orElseThrow(()-> new NotFoundException("Category Not Found"));

        categoryRepository.deleteById(id);

        return Response.builder()
                .status(200)
                .message("Category Successfully Deleted")
                .build();
    }
}
```

SupplierServiceImpl.java

```
package com.cts.inventorymanagementsystem.service.impl;


import com.cts.inventorymanagementsystem.dto.Response;
import com.cts.inventorymanagementsystem.dto.SupplierDTO;
import com.cts.inventorymanagementsystem.entity.Supplier;
import com.cts.inventorymanagementsystem.exceptions.NotFoundException;
import com.cts.inventorymanagementsystem.repository.SupplierRepository;
import com.cts.inventorymanagementsystem.service.SupplierService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.modelmapper.ModelMapper;
import org.modelmapper.TypeToken;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class SupplierServiceImpl implements SupplierService {

    private final SupplierRepository supplierRepository;
    private final ModelMapper modelMapper;

    @Override
    public Response addSupplier(SupplierDTO supplierDTO) {
        Supplier supplierToSave = modelMapper.map(supplierDTO, Supplier.class);
        supplierRepository.save(supplierToSave);

        return Response.builder()
                .status(200)
                .message("Supplier added successfully")
                .build();
    }

    @Override
    public Response updateSupplier(Long id, SupplierDTO supplierDTO) {
```

```
        Supplier existingSupplier = supplierRepository.findById(id)
                .orElseThrow(()-> new NotFoundException("Supplier Not Found"));

        if (supplierDTO.getName() != null) existingSupplier.setName(supplierDTO.getName());
        if (supplierDTO.getAddress() != null) existingSupplier.setAddress(supplierDTO.getAddress());

        supplierRepository.save(existingSupplier);

        return Response.builder()
                .status(200)
                .message("Supplier Successfully Updated")
                .build();
    }

    @Override
    public Response getAllSuppliers() {

        List<Supplier> categories = supplierRepository.findAll(Sort.by(Sort.Direction.DESC, "id"));

        List<SupplierDTO> supplierDTOS = modelMapper.map(categories, new TypeToken<List<SupplierDTO>>()
{}.getType());

        return Response.builder()
                .status(200)
                .message("success")
                .suppliers(supplierDTOS)
                .build();
    }

    @Override
    public Response getSupplierById(Long id) {

        Supplier supplier = supplierRepository.findById(id)
                .orElseThrow(()-> new NotFoundException("Supplier Not Found"));

        SupplierDTO supplierDTO = modelMapper.map(supplier, SupplierDTO.class);

        return Response.builder()
                .status(200)
                .message("success")
                .supplier(supplierDTO)
                .build();
    }

    @Override
    public Response deleteSupplier(Long id) {

        supplierRepository.findById(id)
                .orElseThrow(()-> new NotFoundException("Supplier Not Found"));

        supplierRepository.deleteById(id);

        return Response.builder()
                .status(200)
                .message("Supplier Successfully Deleted")
                .build();
```

```
    }
}
```

TransactionServiceImpl.java

```
package com.cts.inventorymanagementsystem.service.impl;

import com.cts.inventorymanagementsystem.dto.Response;
import com.cts.inventorymanagementsystem.dto.TransactionDTO;
import com.cts.inventorymanagementsystem.dto.TransactionRequest;
import com.cts.inventorymanagementsystem.entity.Product;
import com.cts.inventorymanagementsystem.entity.Supplier;
import com.cts.inventorymanagementsystem.entity.Transaction;
import com.cts.inventorymanagementsystem.entity.User;
import com.cts.inventorymanagementsystem.enums.TransactionStatus;
import com.cts.inventorymanagementsystem.enums.TransactionType;
import com.cts.inventorymanagementsystem.exceptions.NameValueRequiredException;
import com.cts.inventorymanagementsystem.exceptions.NotFoundException;
import com.cts.inventorymanagementsystem.repository.ProductRepository;
import com.cts.inventorymanagementsystem.repository.SupplierRepository;
import com.cts.inventorymanagementsystem.repository.TransactionRepository;
import com.cts.inventorymanagementsystem.service.TransactionService;
import com.cts.inventorymanagementsystem.service.UserService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.modelmapper.ModelMapper;
import org.modelmapper.TypeToken;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class TransactionServiceImpl implements TransactionService {

    private final TransactionRepository transactionRepository;
    private final ModelMapper modelMapper;
    private final SupplierRepository supplierRepository;
    private final UserService userService;
    private final ProductRepository productRepository;



    @Override
    public Response restockInventory(TransactionRequest transactionRequest) {

        Long productId = transactionRequest.getProductId();
        Long supplierId = transactionRequest.getSupplierId();
        Integer quantity = transactionRequest.getQuantity();

        if (supplierId == null) throw new NameValueRequiredException("Supplier Id id Required");
```

```
    Product product = productRepository.findById(productId)
            .orElseThrow(()-> new NotFoundException("Product Not Found"));

    Supplier supplier = supplierRepository.findById(supplierId)
            .orElseThrow(()-> new NotFoundException("Supplier Not Found"));

    User user = userService.getCurrentLoggedInUser();

    //update the stock quantity and re-save
    product.setStockQuantity(product.getStockQuantity() + quantity);
    productRepository.save(product);

    //create a transaction
    Transaction transaction = Transaction.builder()
            .transactionType(TransactionType.PURCHASE)
            .status(TransactionStatus.COMPLETED)
            .product(product)
            .user(user)
            .supplier(supplier)
            .totalProducts(quantity)
            .totalPrice(product.getPrice().multiply(BigDecimal.valueOf(quantity)))
            .description(transactionRequest.getDescription())
            .build();

    transactionRepository.save(transaction);

    return Response.builder()
            .status(200)
            .message("Transaction Made Successfully")
            .build();
}

@Override
public Response sell(TransactionRequest transactionRequest) {

    Long productId = transactionRequest.getProductId();
    Integer quantity = transactionRequest.getQuantity();


    Product product = productRepository.findById(productId)
            .orElseThrow(()-> new NotFoundException("Product Not Found"));


    User user = userService.getCurrentLoggedInUser();

    //update the stock quantity and re-save
    product.setStockQuantity(product.getStockQuantity() - quantity);
    productRepository.save(product);

    //create a transaction
    Transaction transaction = Transaction.builder()
            .transactionType(TransactionType.SALE)
            .status(TransactionStatus.COMPLETED)
            .product(product)
            .user(user)
            .totalProducts(quantity)
```

```
            .totalPrice(product.getPrice().multiply(BigDecimal.valueOf(quantity)))
            .description(transactionRequest.getDescription())
            .build();

    transactionRepository.save(transaction);

    return Response.builder()
            .status(200)
            .message("Transaction Sold Successfully")
            .build();
}

@Override
public Response returnToSupplier(TransactionRequest transactionRequest) {

    Long productId = transactionRequest.getProductId();
    Long supplierId = transactionRequest.getSupplierId();
    Integer quantity = transactionRequest.getQuantity();

    if (supplierId == null) throw new NameValueRequiredException("Supplier Id id Required");

    Product product = productRepository.findById(productId)
            .orElseThrow(()-> new NotFoundException("Product Not Found"));

    Supplier supplier = supplierRepository.findById(supplierId)
            .orElseThrow(()-> new NotFoundException("Supplier Not Found"));

    User user = userService.getCurrentLoggedInUser();

    //update the stock quantity and re-save
    product.setStockQuantity(product.getStockQuantity() - quantity);
    productRepository.save(product);

    //create a transaction
    Transaction transaction = Transaction.builder()
            .transactionType(TransactionType.RETURN_TO_SUPPLIER)
            .status(TransactionStatus.PROCESSING)
            .product(product)
            .user(user)
            .supplier(supplier)
            .totalProducts(quantity)
            .totalPrice(BigDecimal.ZERO)
            .description(transactionRequest.getDescription())
            .build();

    transactionRepository.save(transaction);

    return Response.builder()
            .status(200)
            .message("Transaction Returned Successfully Initialized")
            .build();
}

@Override
public Response getAllTransactions(int page, int size, String searchText) {

    Pageable pageable = PageRequest.of(page, size, Sort.by(Sort.Direction.DESC, "id"));
```

```
    Page<Transaction> transactionPage = transactionRepository.searchTransactions(searchText, pageable);

    List<TransactionDTO> transactionDTOS = modelMapper
            .map(transactionPage.getContent(), new TypeToken<List<TransactionDTO>>() {}.getType());

    transactionDTOS.forEach(transactionDTOItem -> {
        transactionDTOItem.setUser(null);
        transactionDTOItem.setProduct(null);
        transactionDTOItem.setSupplier(null);
    });


    return Response.builder()
            .status(200)
            .message("success")
            .transactions(transactionDTOS)
            .build();
}

@Override
public Response getTransactionById(Long id) {
    Transaction transaction = transactionRepository.findById(id)
            .orElseThrow(()-> new NotFoundException("Transaction Not Found"));

    TransactionDTO transactionDTO = modelMapper.map(transaction, TransactionDTO.class);

    transactionDTO.getUser().setTransactions(null); //removing the user trnasaction list

    return Response.builder()
            .status(200)
            .message("success")
            .transaction(transactionDTO)
            .build();

}

@Override
public Response getAllTransactionByMonthAndYear(int month, int year) {

   List<Transaction> transactions = transactionRepository.findAllByMonthAndYear(month, year);

    List<TransactionDTO> transactionDTOS = modelMapper
            .map(transactions, new TypeToken<List<TransactionDTO>>() {}.getType());

    transactionDTOS.forEach(transactionDTOItem -> {
        transactionDTOItem.setUser(null);
        transactionDTOItem.setProduct(null);
        transactionDTOItem.setSupplier(null);
    });


    return Response.builder()
            .status(200)
            .message("success")
            .transactions(transactionDTOS)
            .build();
}
```

```
    @Override
    public Response updateTransactionStatus(Long transactionId, TransactionStatus transactionStatus) {

        Transaction existingTransaction = transactionRepository.findById(transactionId)
                .orElseThrow(()-> new NotFoundException("Transaction Not Found"));

        existingTransaction.setStatus(transactionStatus);
        existingTransaction.setUpdatedAt(LocalDateTime.now());

        transactionRepository.save(existingTransaction);

        return Response.builder()
                .status(200)
                .message("Transaction Status Successfully Updated")
                .build();
    }
}
```

Repository Class:

ProductRepository.java

```
package com.cts.inventorymanagementsystem.repository;
import com.cts.inventorymanagementsystem.entity.Product;
import org.springframework.data.jpa.repsitory.JpaRepository;
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

CategoryRepository.java

```
package com.cts.inventorymanagementsystem.repository;
import com.cts.inventorymanagementsystem.entity.Category;
import org.springframework.data.jpa.resitory.JpaRepository;
public interface CategoryRepository extends JpaRepository<Category, Long> {
}
```

SupplierRepository.java

```
package com.cts.inventorymanagementsystem.repository;
import com.cts.inventorymanagementsystem.entity.Supplier;
import org.springframework.data.jpa.repository.JpaRepository;
public interface SupplierRepository extends JpaRepository<Supplier, Long> {
}
```

TransactionRepository.java

```
package com.cts.inventorymanagementsystem.repository;
import com.cts.inventorymanagementsystem.entity.Transaction;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;
public interface TransactionRepository extends JpaRepository<Transaction, Long> {

    @Query("SELECT t FROM Transaction t " +
            "WHERE YEAR(t.createdAt) = :year AND MONTH(t.createdAt) = :month")
    List<Transaction> findAllByMonthAndYear(@Param("month") int month, @Param("year") int year);
    @Query("SELECT t FROM Transaction t " +
```

```
        "LEFT JOIN t.product p " +
        "WHERE (:searchText IS NULL OR " +
        "LOWER(t.description) LIKE LOWER(CONCAT('%', :searchText, '%')) OR " +
        "LOWER(t.status) LIKE LOWER(CONCAT('%', :searchText, '%')) OR " +
        "LOWER(p.name) LIKE LOWER(CONCAT('%', :searchText, '%')) OR " +
        "LOWER(p.sku) LIKE LOWER(CONCAT('%', :searchText, '%')))")
    Page<Transaction> searchTransactions(@Param("searchText") String searchText, Pageable pageable);
}
```

aspplications.properties

```
spring.application.name=InventoryManagementSystem
server.port=5050

#MYSQL CONNECTION
spring.datasource.url=jdbc:mysql://localhost:3306/inventory_db
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update


#THE MAXIMUM SIGN OF IMAGE THAT CAN BE UPLOADED
spring.servlet.multipart.max-file-size=2GB
spring.servlet.multipart.max-request-size=2GB

#Secret key for the server
secreteJwtString=ctsproject123456789ctsproject123456789
```

FrontEnd Logic Code:

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>IMS</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

app.component.html

```
<div class="dashboard-layout">
```

```
  <!-- SIDEBAR -->
  <div class="sidebar">
    <h1 class="ims">IMS</h1>
    <ul class="nav-links">
      <li *ngIf="isAuth()"><a routerLink="/dashboard">Dashboard</a></li>
      <li *ngIf="isAuth()"><a routerLink="/transaction">Transactions</a></li>
      <li *ngIf="isAdmin()"><a routerLink="/category">Category</a></li>
      <li *ngIf="isAdmin()"><a routerLink="/product">Product</a></li>
      <li *ngIf="isAdmin()"><a routerLink="/supplier">Supplier</a></li>
      <li *ngIf="isAuth()"><a routerLink="/purchase">Purchase</a></li>
      <li *ngIf="isAuth()"><a routerLink="/sell">Sell</a></li>
      <li *ngIf="isAuth()"><a routerLink="/profile">Profile</a></li>
      <li *ngIf="!isAuth()"><a routerLink="/login">Login</a></li>
      <li *ngIf="!isAuth()"><a routerLink="/register">Register</a></li>
      <li *ngIf="isAuth()"><a (click)="logOut()" >Logout</a></li>
    </ul>
  </div>

  <!-- MAIN CONTENT Area-->
   <div class="main-content">
      <router-outlet></router-outlet>
   </div>
</div>
```

app.component.ts

```
import { CommonModule } from '@angular/common';
import { ChangeDetectorRef, Component } from '@angular/core';
import { Router, RouterLink, RouterOutlet } from '@angular/router';
import { ApiService } from './service/api.service';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, RouterLink, CommonModule],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css',
})

export class AppComponent {
  title = 'ims';
  constructor(
    private apiService: ApiService,
    private router: Router,
    private cdr: ChangeDetectorRef
  ) {}

isAuth():boolean{
  return this.apiService.isAuthenticated();
}

isAdmin():boolean{
  return this.apiService.isAdmin();
}

logOut():void{
  this.apiService.logout();
  this.router.navigate(["/login"])
```

```
  this.cdr.detectChanges();
}
}
```

app.config.ts

```
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';
import { routes } from './app.routes';
import { provideHttpClient } from '@angular/common/http';

export const appConfig: ApplicationConfig = {
  providers: [provideZoneChangeDetection({ eventCoalescing: true }), provideRouter(routes), provideHttpClient()]
};
```

app.routes.ts

```
import { Routes } from '@angular/router';
import { GuardService } from './service/guard.service';
import { LoginComponent } from './login/login.component';
import { RegisterComponent } from './register/register.component';
import { CategoryComponent } from './category/category.component';
import { SupplierComponent } from './supplier/supplier.component';
import { AddEditSupplierComponent } from './add-edit-supplier/add-edit-supplier.component';
import { ProductComponent } from './product/product.component';
import { AddEditProductComponent } from './add-edit-product/add-edit-product.component';
import { PurchaseComponent } from './purchase/purchase.component';
import { SellComponent } from './sell/sell.component';
import { TransactionComponent } from './transaction/transaction.component';
import { TransactionDetailsComponent } from './transaction-details/transaction-details.component';
import { ProfileComponent } from './profile/profile.component';
import { DashboardComponent } from './dashboard/dashboard.component';

export const routes: Routes = [

  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },

  { path: 'category', component: CategoryComponent, canActivate:[GuardService], data: {requiresAdmin: true} },

  { path: 'supplier', component: SupplierComponent, canActivate:[GuardService], data: {requiresAdmin: true} },
  { path: 'edit-supplier/:supplierId', component: AddEditSupplierComponent, canActivate:[GuardService], data:
{requiresAdmin: true} },
  { path: 'add-supplier', component: AddEditSupplierComponent, canActivate:[GuardService], data:
{requiresAdmin: true} },

  { path: 'product', component: ProductComponent, canActivate:[GuardService], data: {requiresAdmin: true} },
  { path: 'edit-product/:productId', component: AddEditProductComponent, canActivate:[GuardService], data:
{requiresAdmin: true} },
  { path: 'add-product', component: AddEditProductComponent, canActivate:[GuardService], data:
{requiresAdmin: true} },

  { path: 'purchase', component: PurchaseComponent, canActivate:[GuardService] },
  { path: 'sell', component: SellComponent, canActivate:[GuardService] },

  { path: 'transaction', component: TransactionComponent, canActivate:[GuardService] },
  { path: 'transaction/:transactionId', component: TransactionDetailsComponent, canActivate:[GuardService] },

  { path: 'profile', component: ProfileComponent, canActivate:[GuardService] },
```

```
  { path: 'dashboard', component: DashboardComponent, canActivate:[GuardService] },

    {path: "", redirectTo: "/login", pathMatch: 'full'},


];
```

Supplier Component:

supplier.component.html

```
<div class="supplier-page">
  <p *ngIf="message" class="message">{{ message }}</p>

  <!-- SUPPLIER HEADER -->
  <div class="supplier-header">
    <h1>Supplier</h1>
    <div class="add-sup">
      <button (click)="navigateToAddSupplierPage()">Add Supplier</button>
    </div>
  </div>

  <!-- SUPPLIER LIST -->
  <ul class="supplier-list" *ngIf="suppliers.length > 0">
    <li
      *ngFor="let supplier of suppliers"
      class="supplier-item"
      [attr.key]="supplier.id"
    >
      <span>{{ supplier.name }}</span>

      <div class="supplier-actions">
          <button (click)="navigateToEditSupplierPage(supplier.id)" >Edit</button>
          <button (click)="handleDeleteSupplier(supplier.id)">Delete</button>
      </div>
    </li>
  </ul>
</div>
```

login.component.html

```
<div class="auth-container">
    <h2>Login</h2>
    <p *ngIf="message" class="message">{{ message }}</p>
    <form (ngSubmit)="handleSubmit()">
      <input
        type="email"
        placeholder="Email ..."
        [(ngModel)]="formData.email"
        name="email"
        required
      />
      <input
        type="password"
        placeholder="Password ..."
```

```
      [(ngModel)]="formData.password"
      name="password"
      required
     />
     <button type="submit">Login</button>
   </form>
   <p>Don't have an account?<a routerLink="/register" routerLinkActive="router-link-active" >Register</a></p>
  </div>
```

login.component.ts

```
import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { Router, RouterLink } from '@angular/router';
import { ApiService } from '../service/api.service';
import { firstValueFrom } from 'rxjs';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [FormsModule, CommonModule, RouterLink],
  templateUrl: './login.component.html',
  styleUrl: './login.component.css'
})
export class LoginComponent {
  constructor(private apiService:ApiService, private router:Router){}

  formData: any = {
    email: '',
    password: ''
  };

  message:string | null = null;

  async handleSubmit(){
    if(
      !this.formData.email ||
      !this.formData.password
    ){
      this.showMessage("All fields are required");
      return;
    }

    try {
      const response: any = await firstValueFrom(
        this.apiService.loginUser(this.formData)
      );
      if (response.status === 200) {
        this.apiService.encryptAndSaveToStorage('token', response.token);
        this.apiService.encryptAndSaveToStorage('role', response.role);
        this.router.navigate(["/dashboard"]);
      }
    } catch (error:any) {
      console.log(error)
      this.showMessage(error?.error?.message || error?.message || "Unable to Login a user" + error)

    }
```

```
  }

  showMessage(message:string){
    this.message = message;
    setTimeout(() =>{
      this.message = null
    }, 4000)
  }

}
```

Dashboard Component:

dashboard.component.html

```
<div class="dashboard-container">
    <h1>Transaction Dashboard</h1>


    <!-- Transaction Selection -->
    <div style="margin-bottom: 50px; margin-top: 30px;">
        <label>Select Month:</label>
        <select [(ngModel)]="selectedMonth">
          <option *ngFor="let month of months" [value]="month.value">{{ month.name }}</option>
        </select>

        <label>Select Year:</label>
        <select [(ngModel)]="selectedYear">
          <option *ngFor="let year of years" [value]="year">{{ year }}</option>
        </select>

        <button (click)="loadMonthlyData()">Show Monthly Data</button>
      </div>


    <!-- Transaction Type Count Bar Chart -->
    <div style="margin-bottom: 30px;">
      <h2>Transaction Counts by Type</h2>
      <ngx-charts-bar-vertical
        [view]="view"
        [scheme]="'vivid'"
        [results]="transactionTypeData"
        [gradient]="false"
        [xAxis]="true"
        [yAxis]="true"
        [legend]="showLegend"
        [showXAxisLabel]="true"
        [showYAxisLabel]="true"
        xAxisLabel="Transaction Type"
        yAxisLabel="Count"
        [animations]="animations">
      </ngx-charts-bar-vertical>
    </div>
```

```
    <!-- Transaction Amount by Type Pie Chart -->
    <div style="margin-bottom: 30px;">
      <h2>Total Transaction Amount by Type</h2>
      <ngx-charts-pie-chart
        [view]="view"
        [scheme]="'cool'"
        [results]="transactionAmountData"
        [legend]="showLegend"
        [labels]="showLabels"
        [doughnut]="false"
        [animations]="animations">
      </ngx-charts-pie-chart>
    </div>


    <!-- Monthly Transaction Bar Chart -->
    <h2 style="margin-bottom: 50px;">Monthly Transaction Chart</h2>

    <div *ngIf="monthlyTransactionData.length">
      <ngx-charts-bar-vertical
        [view]="view"
        [scheme]="'flame'"
        [results]="monthlyTransactionData"
        [gradient]="false"
        [xAxis]="true"
        [yAxis]="true"
        [legend]="showLegend"
        [showXAxisLabel]="true"
        [showYAxisLabel]="true"
        xAxisLabel="Day of Month"
        yAxisLabel="Total Price"
        [animations]="animations">
      </ngx-charts-bar-vertical>
    </div>
  </div>
```

dashboard.component.ts

```
  // Import necessary Angular modules and services
import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import { NgxChartsModule } from '@swimlane/ngx-charts';  // Module for charts
import { ApiService } from '../service/api.service'; // Service to interact with API
import { FormsModule } from '@angular/forms'; // Forms module for two-way binding

// Define the component metadata
@Component({
  selector: 'app-dashboard', // The component selector
  standalone: true, // Marks this component as standalone (no need for NgModule)
  imports: [CommonModule, NgxChartsModule, FormsModule], // Import other modules required for this
component
  templateUrl: './dashboard.component.html', // HTML template
  styleUrl: './dashboard.component.css', // CSS styles for the component
```

```
})

export class DashboardComponent {
  // Define the properties for storing transaction data and chart data
  transactions: any[] = []; // Array to hold all transactions
  transactionTypeData: any[] = []; // Data for the chart showing count of transactions by type
  transactionAmountData: any[] = []; // Data for the chart showing total amount by transaction type
  monthlyTransactionData: any[] = []; // Data for the chart showing daily totals for the selected month

  // List of months, used for selecting a month
  months = [
    { name: 'January', value: '01' },
    { name: 'February', value: '02' },
    { name: 'March', value: '03' },
    { name: 'April', value: '04' },
    { name: 'May', value: '05' },
    { name: 'June', value: '06' },
    { name: 'July', value: '07' },
    { name: 'August', value: '08' },
    { name: 'September', value: '09' },
    { name: 'October', value: '10' },
    { name: 'November', value: '11' },
    { name: 'December', value: '12' },
  ];

  // Array to store the years (last 10 years from current year)
  years = Array.from({ length: 10 }, (_, i) => new Date().getFullYear() - i);

  // Selected month and year for filtering monthly data
  selectedMonth = '';
  selectedYear = '';

  // Chart view dimensions, legend, and animations settings
  view: [number, number] = [700, 400];  // Chart size: width x height
  showLegend = true;  // Display chart legend
  showLabels = true;  // Display labels on chart
  animations = true;  // Enable chart animations

  // Constructor to inject ApiService for API calls
  constructor(private apiService: ApiService) {}

  // ngOnInit lifecycle hook, called when the component initializes
  ngOnInit(): void {
    this.loadTransactions(); // Load transactions when the component initializes
  }

  // Method to fetch all transactions from the API
  loadTransactions(): void {
    this.apiService.getAllTransactions('').subscribe((data) => {
      this.transactions = data.transactions; // Store transactions data
      this.processChartData(); // Process data to generate charts
    });
```

```
}

// Method to process transaction data for type-based and amount-based charts
processChartData(): void {
  // Object to count the number of transactions by type
  const typeCounts: { [key: string]: number } = {};

  // Object to sum the transaction amounts by type
  const amountByType: { [key: string]: number } = {};

  // Loop through each transaction to calculate totals by type
  this.transactions.forEach((transaction) => {
    const type = transaction.transactionType; // Get the transaction type
    typeCounts[type] = (typeCounts[type] || 0) + 1; // Count transactions by type
    amountByType[type] = (amountByType[type] || 0) + transaction.totalPrice; // Sum amounts by type
  });

  // Prepare data for chart displaying number of transactions by type
  this.transactionTypeData = Object.keys(typeCounts).map((type) => ({
    name: type,
    value: typeCounts[type],
  }));

  // Prepare data for chart displaying total transaction amount by type
  this.transactionAmountData = Object.keys(amountByType).map((type) => ({
    name: type,
    value: amountByType[type],
  }));
}

// Method to load transaction data for a specific month and year
loadMonthlyData(): void {
  // If no month or year is selected, exit the function
  if (!this.selectedMonth || !this.selectedYear) {
    return;
  }

  // Call API to get transactions for the selected month and year
  this.apiService
    .getTransactionsByMonthAndYear(
      Number.parseInt(this.selectedMonth), // Convert month string to number
      Number.parseInt(this.selectedYear) // Convert year string to number
    )
    .subscribe((data) => {
      this.transactions = data.transactions; // Store transactions for the selected month
      this.processChartData(); // Process the overall data for charts
      this.processMonthlyData(data.transactions); // Process the data for the daily chart
    });
}

// Method to process daily transaction data for the selected month
processMonthlyData(transactions: any[]): void {
```

```
    // Object to store daily total amounts (key = day, value = total amount)
    const dailyTotals: { [key: string]: number } = {};

    // Loop through each transaction and accumulate totals for each day
    transactions.forEach((transaction) => {
      const date = new Date(transaction.createdAt).getDate().toString(); // Get the day from transaction date
      dailyTotals[date] = (dailyTotals[date] || 0) + transaction.totalPrice; // Sum daily totals
    });

    // Prepare data for chart displaying daily totals for the selected month
    this.monthlyTransactionData = Object.keys(dailyTotals).map((day) => ({
      name: `Day ${day}`,
      value: dailyTotals[day],
    }));
  }
}
```

Pagination Component

pagination.component.html

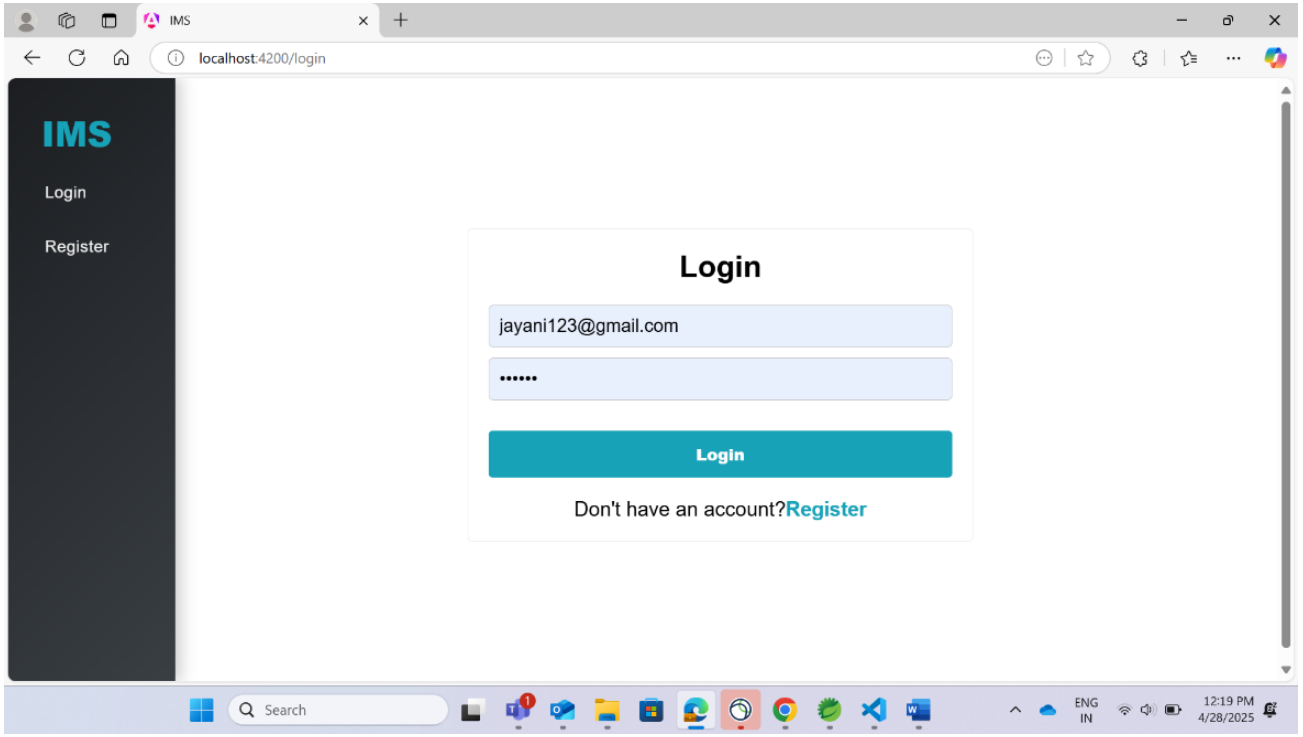
```
<div class="pagination-container">
    <!-- PREVIOIUS BUTTON -->
  <button
    class="pagination-button"
    [disabled]="currentPage === 1"
    (click)="onPageChange(currentPage - 1)"
  >
    &laquo; Prev
  </button>

    <!-- Page numbers BUTTONs -->
  <button
    *ngFor="let number of pageNumbers"
    class="pagination-button"
    [class.active]="currentPage === number"
    (click)="onPageChange(number)"
  >
    {{ number }}
  </button>

    <!-- NEXT BUTTON -->
  <button
    class="pagination-button"
    [disabled]="currentPage === totalPages"
    (click)="onPageChange(currentPage + 1)"
  >
    Next &raquo;
  </button>
</div>
```

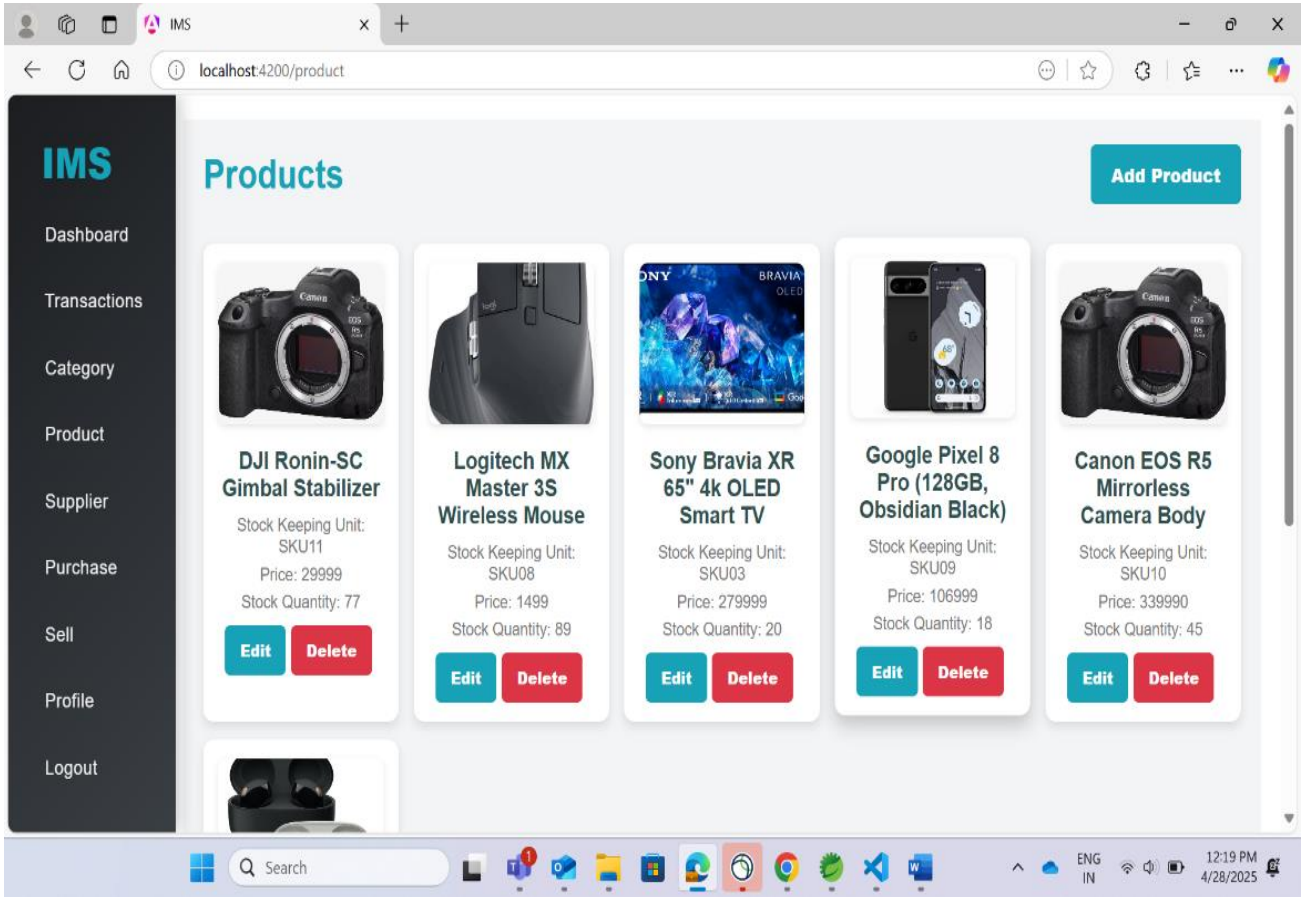
pagination.component.ts

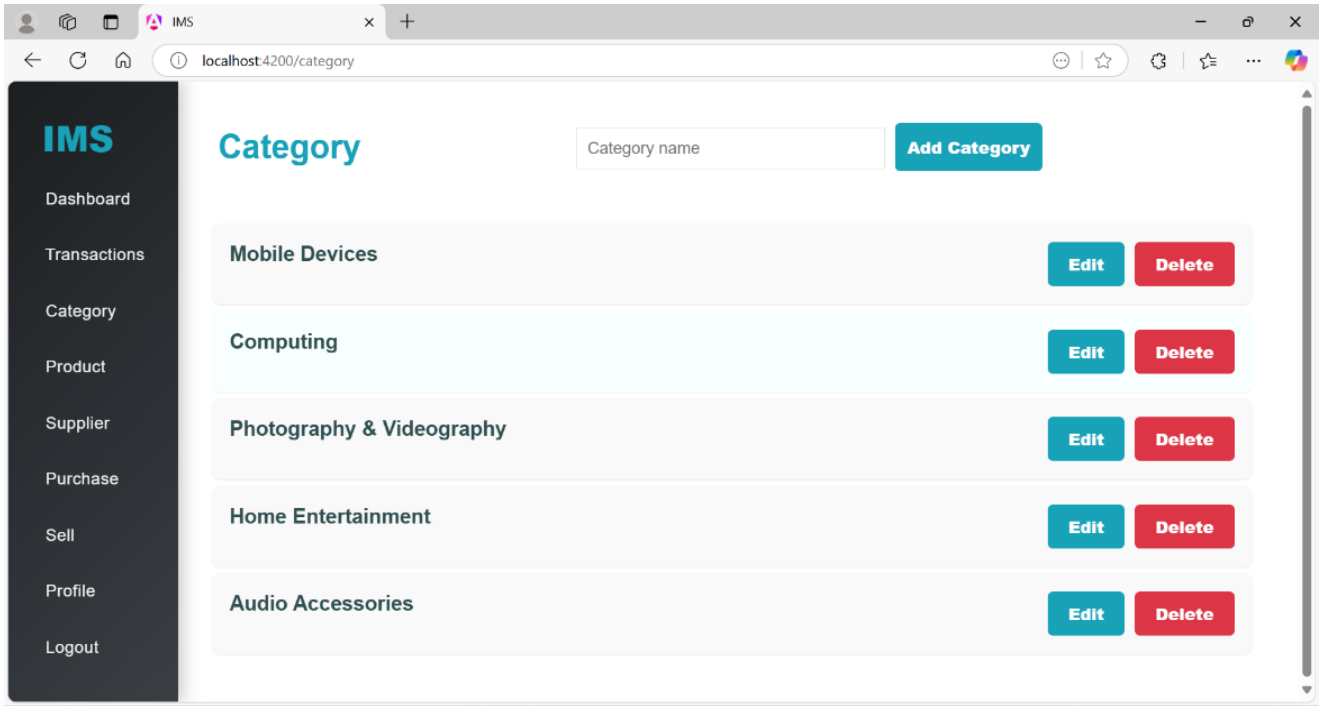
```
import { CommonModule } from '@angular/common';
```

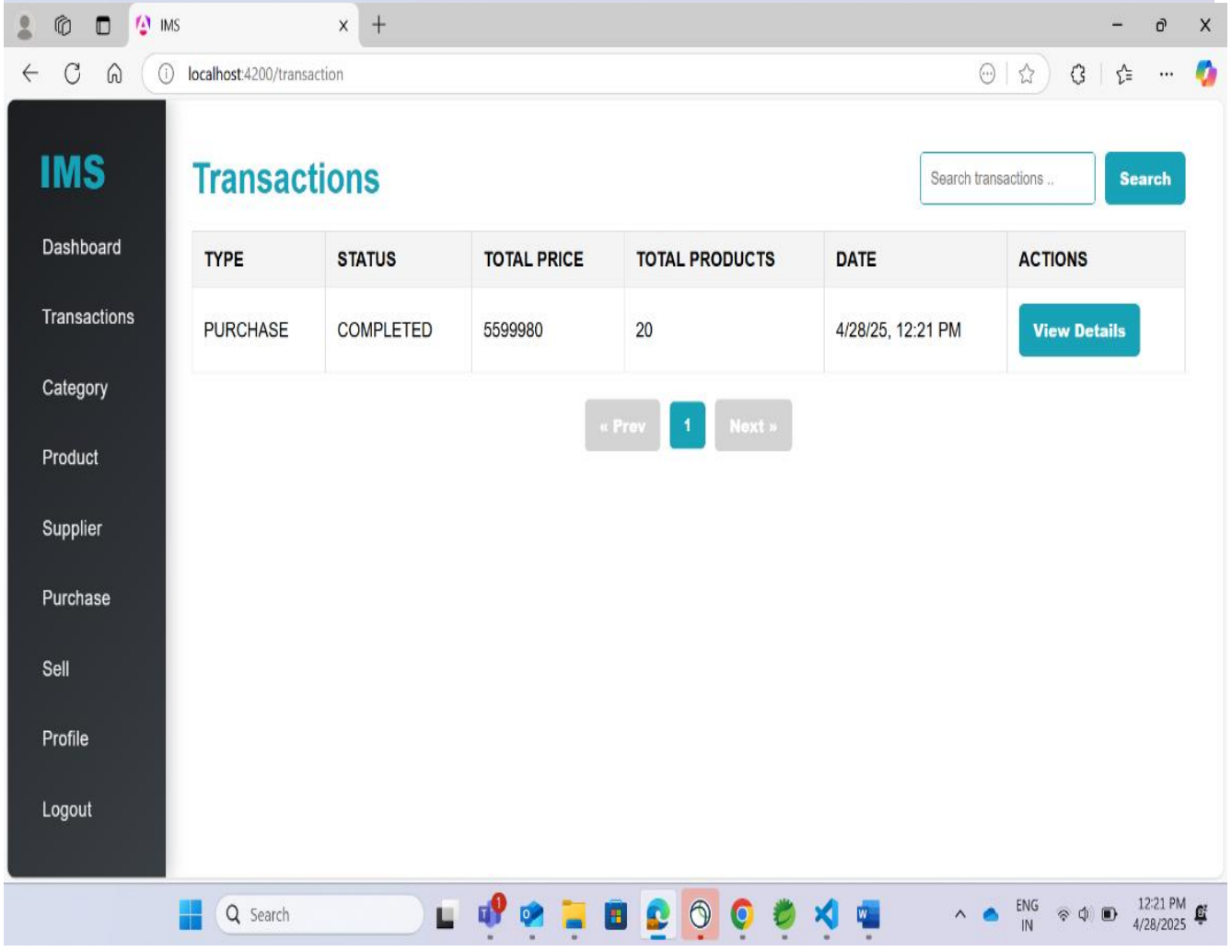
```
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'app-pagination',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './pagination.component.html',
  styleUrl: './pagination.component.css'
})
export class PaginationComponent {

  @Input() currentPage: number = 1;
  @Input() totalPages: number = 1;
  @Output() pageChange = new EventEmitter<number>;

  //method to generate page numbers
  get pageNumbers(){
    return Array.from({length: this.totalPages}, (_, i)=> i+1);
  }

  //method to handle page change
  onPageChange(page: number):void{
    if (page >= 1 && page <= this.totalPages) {
      this.pageChange.emit(page)
    }
  }

}
```

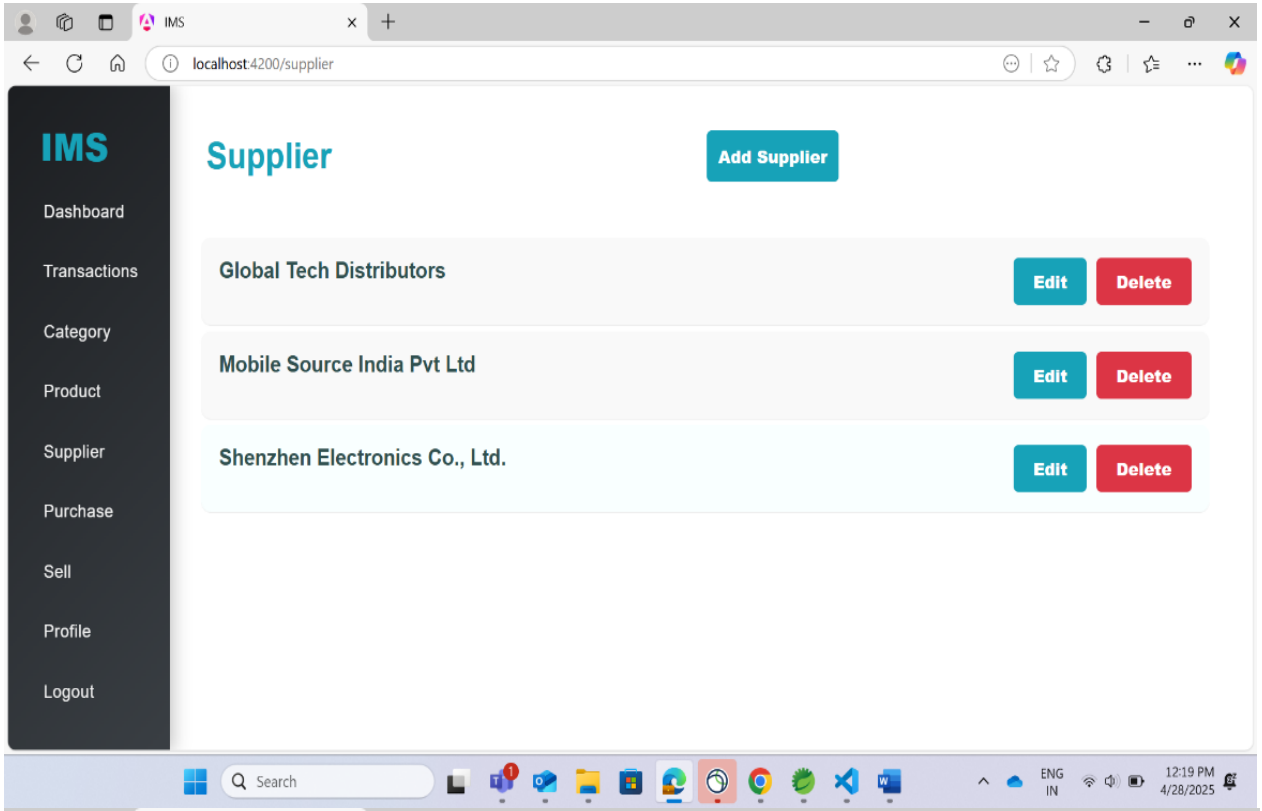
3.5.15 Output/result

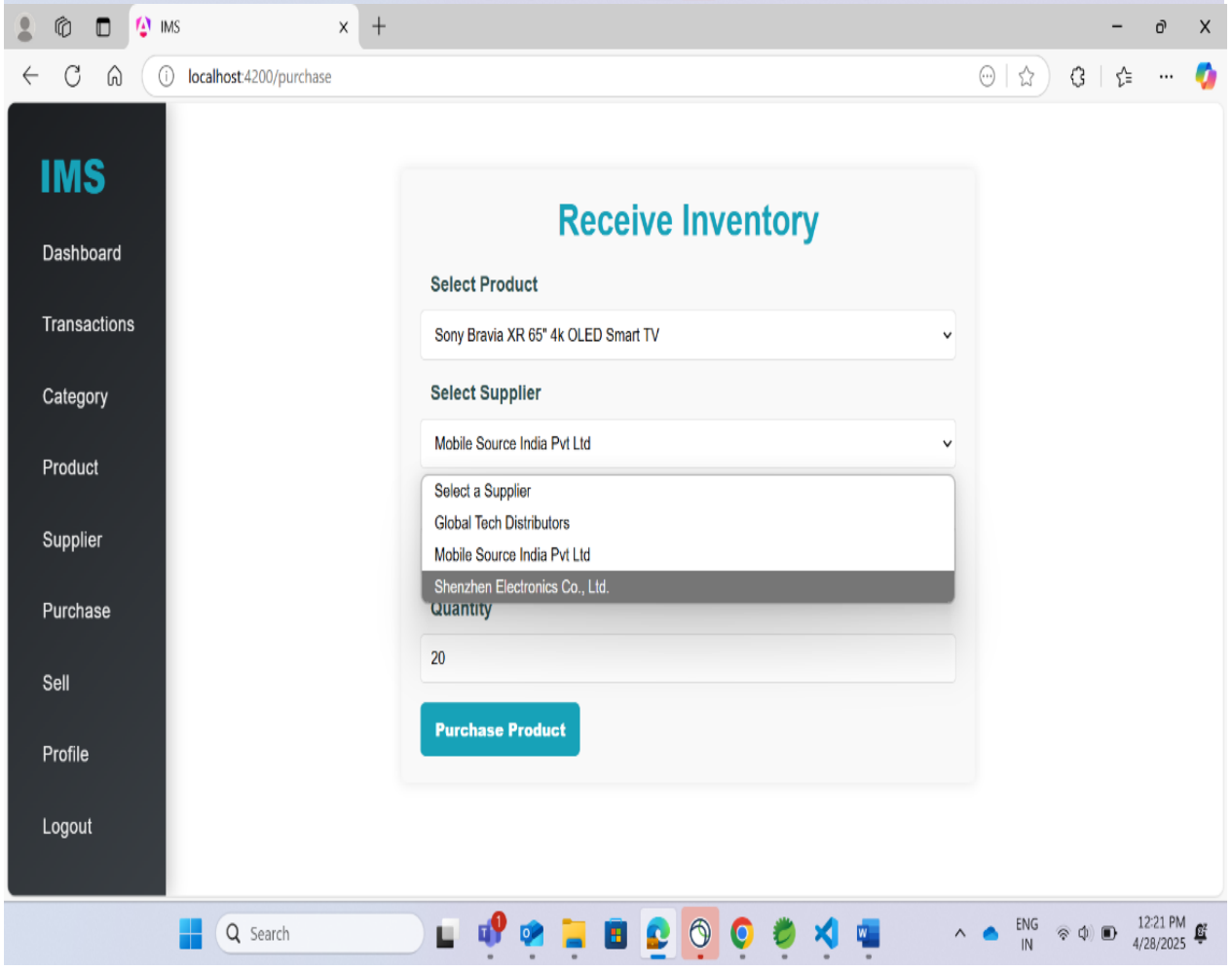


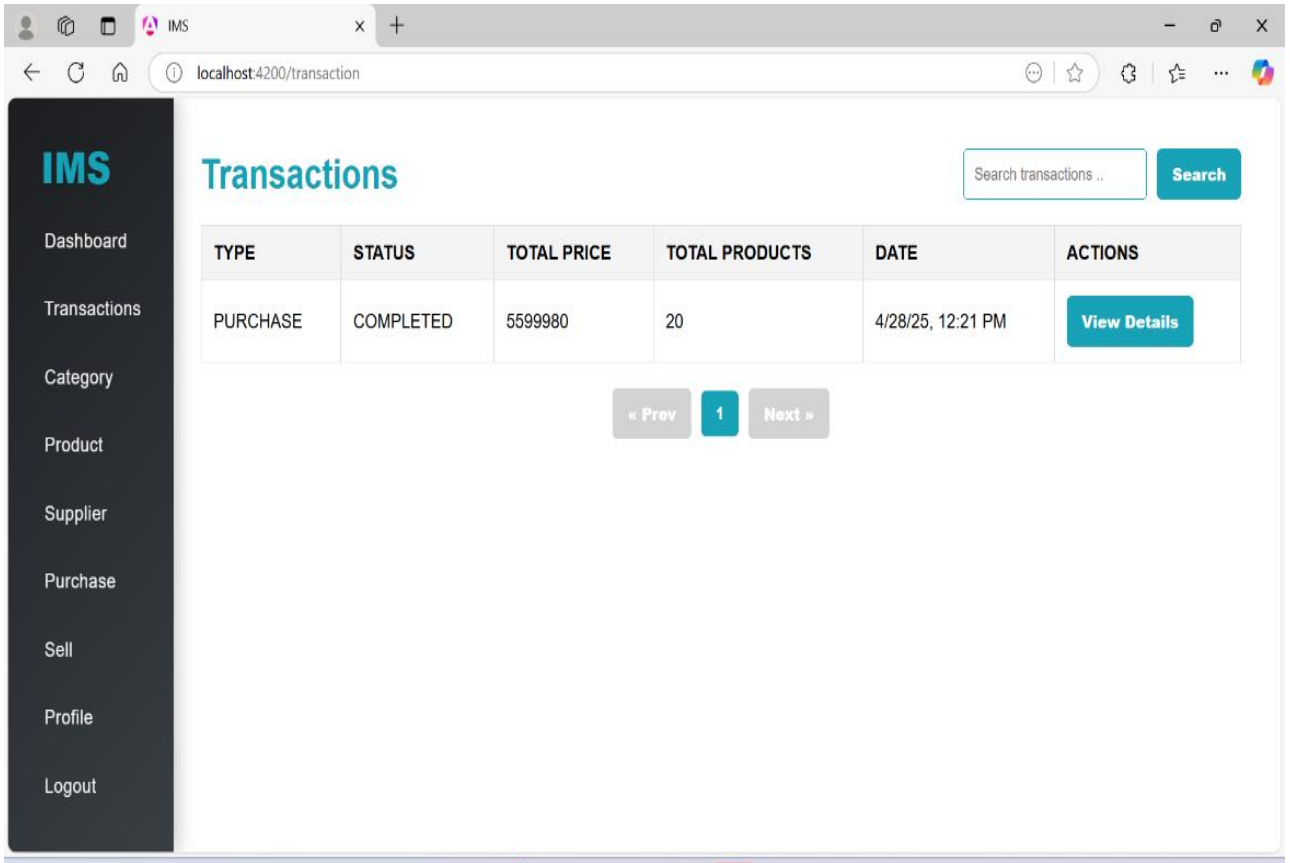


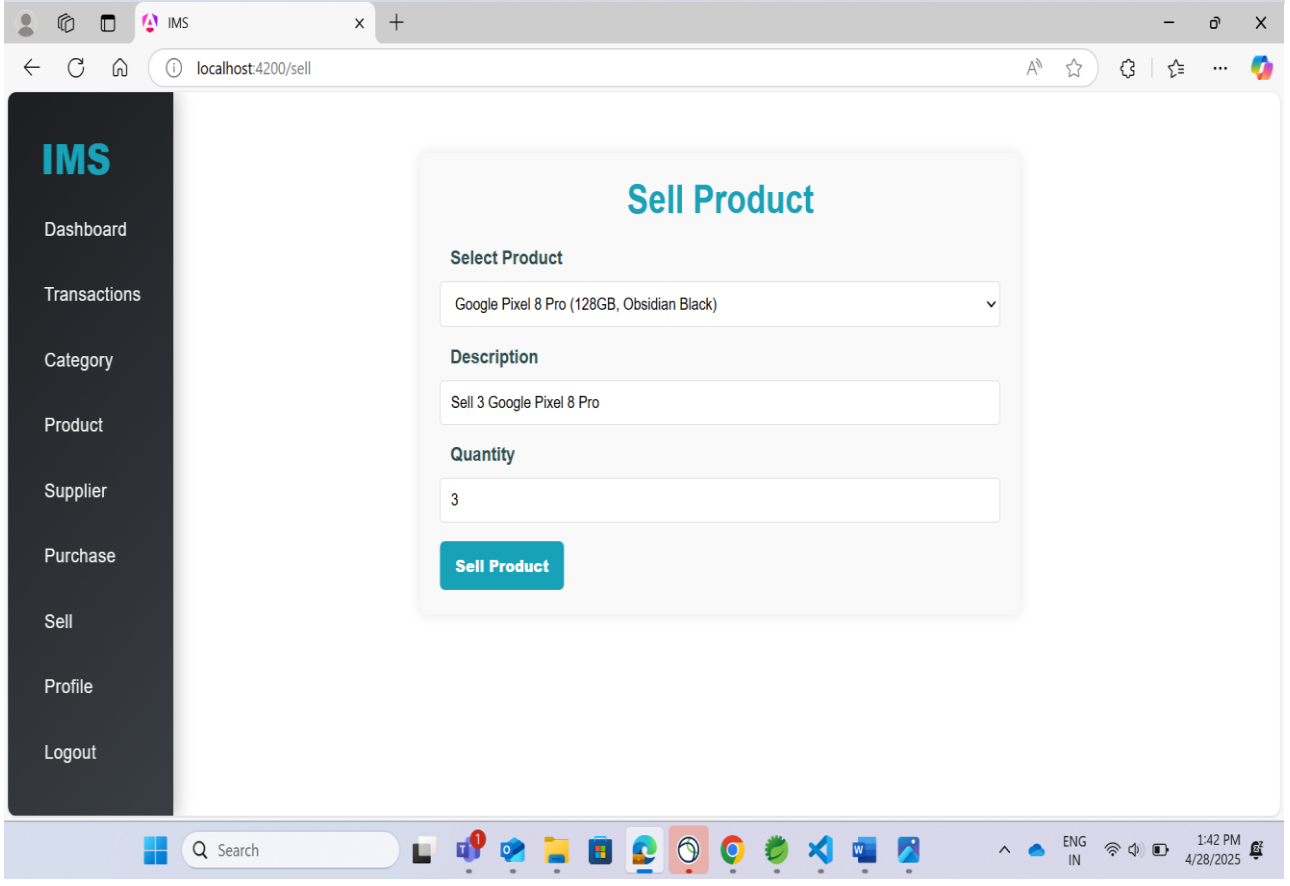

IMS
localhost:4200/category
IMS
Dashboard
Transactions
Category
Product
Supplier
Purchase
Sell
Profile
Logout
Category
Category name
Add Category
Mobile Devices
Edit
Delete
Computing
Edit
Delete
Photography & Videography
Edit
Delete
Home Entertainment
Edit
Delete
Audio Accessories
Edit
Delete
Search
ENG
IN
12:19 PM
4/28/2025

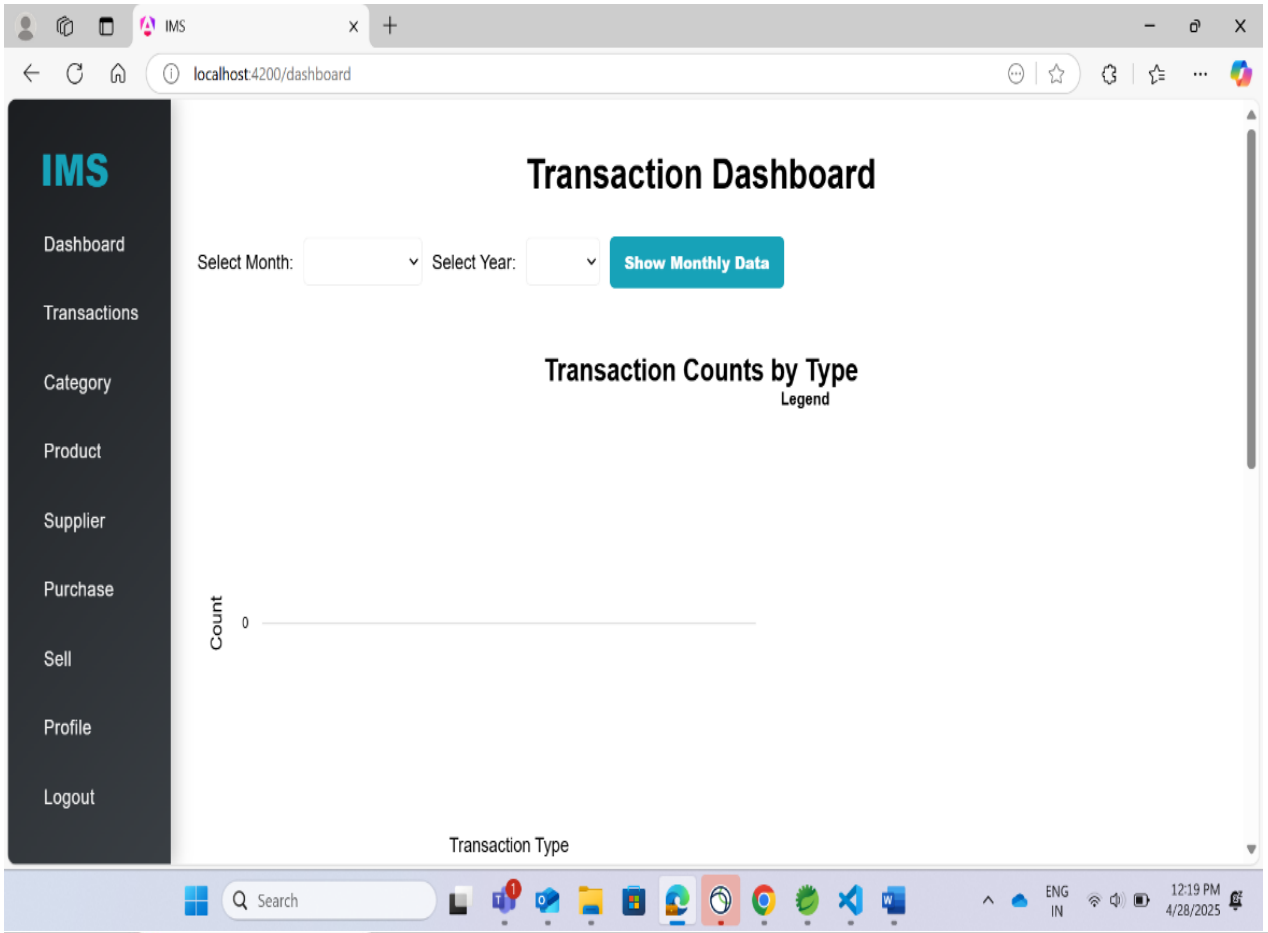

IMS
localhost:4200/transaction
IMS
Dashboard
Transactions
Category
Product
Supplier
Purchase
Sell
Profile
Logout
Transactions
Search transactions ..
Search
TYPE
STATUS
TOTAL PRICE
TOTAL PRODUCTS
DATE
ACTIONS
PURCHASE
COMPLETED
5599980
20
4/28/25, 12:21 PM
View Details
« Prev
1
Next »
Search
ENG
IN
12:21 PM
4/28/2025

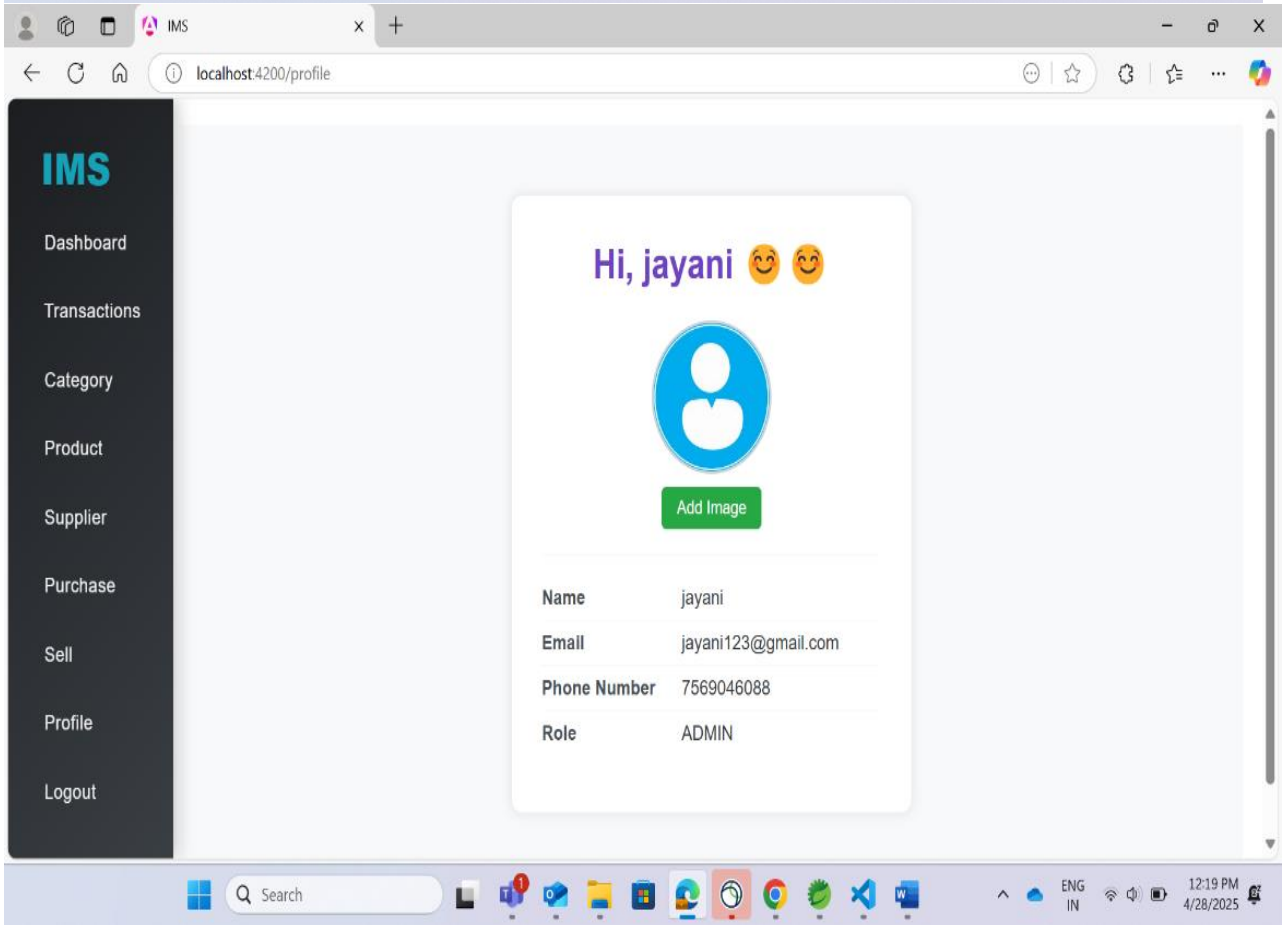

IMS
localhost:4200/supplier
IMS
Dashboard
Transactions
Category
Product
Supplier
Purchase
Sell
Profile
Logout
Supplier
Add Supplier
Global Tech Distributors
Edit
Delete
Mobile Source India Pvt Ltd
Edit
Delete
Shenzhen Electronics Co., Ltd.
Edit
Delete
Search
ENG
IN
12:19 PM
4/28/2025


IMS
localhost:4200/purchase
IMS
Dashboard
Transactions
Category
Product
Supplier
Purchase
Sell
Profile
Logout
Receive Inventory
Select Product
Sony Bravia XR 65" 4k OLED Smart TV
Select Supplier
Mobile Source India Pvt Ltd
Select a Supplier
Global Tech Distributors
Mobile Source India Pvt Ltd
Shenzhen Electronics Co., Ltd.
Quantity
20
Purchase Product
Search
ENG
IN
12:21 PM
4/28/2025


IMS
localhost:4200/transaction
IMS
Dashboard
Transactions
Category
Product
Supplier
Purchase
Sell
Profile
Logout
Transactions
Search transactions ..
Search
TYPE
STATUS
TOTAL PRICE
TOTAL PRODUCTS
DATE
ACTIONS
PURCHASE
COMPLETED
5599980
20
4/28/25, 12:21 PM
View Details
« Prev
1
Next »
Search
ENG
IN
12:21 PM
4/28/2025


IMS
localhost:4200/sell
IMS
Dashboard
Transactions
Category
Product
Supplier
Purchase
Sell
Profile
Logout
Sell Product
Select Product
Google Pixel 8 Pro (128GB, Obsidian Black)
Description
Sell 3 Google Pixel 8 Pro
Quantity
3
Sell Product
Search
ENG
IN
1:42 PM
4/28/2025


IMS
localhost:4200/dashboard
IMS
Dashboard
Transactions
Category
Product
Supplier
Purchase
Sell
Profile
Logout
Transaction Dashboard
Select Month:
Select Year:
Show Monthly Data
Transaction Counts by Type
Legend
Count
0
Transaction Type
Search
ENG
IN
12:19 PM
4/28/2025


IMS
localhost:4200/profile
IMS
Dashboard
Transactions
Category
Product
Supplier
Purchase
Sell
Profile
Logout
Hi, jayani
Add Image
Name
jayani
Email
jayani123@gmail.com
Phone Number
7569046088
Role
ADMIN
Search
ENG
IN
12:19 PM
4/28/2025

Conclusion :

This Inventory Management System provides a comprehensive and foundational structure for an e- commerce system, specifically tailored for desktop environments. The entire application is developed using Java programming language, leveraging the capabilities of the Spring Tool Suite for streamlined development and management. For database operations, JDBC (Java Database Connectivity) is utilized, ensuring efficient, secure, and reliable interaction with the underlying relational database. To offer users a seamless and intuitive graphical experience, the application incorporates Java Swing for building the graphical user interface (GUI).

The system encompasses all the essential features expected from a basic e-commerce platform. Core functionalities include product management, where administrators can add, update, or delete products from the inventory. Each product is represented with necessary attributes such as name, price, description, and availability status, ensuring that the catalog remains organized and user-friendly.

In addition, the application supports a robust cart functionality that allows users to browse through products, select desired items, and add them to a virtual shopping cart. Users can manage their cart contents by updating quantities or removing products before finalizing their orders. This ensures a flexible and user-centered shopping experience.

Another crucial aspect is the order placement feature, where users can review their selected products and confirm their purchases. The system handles order processing, updates inventory levels accordingly, and ensures that all transactional data is securely stored and managed within the database.

Security and data integrity are key considerations throughout the application. Secure transactions are prioritized, with careful handling of sensitive customer information and transaction details. Error handling mechanisms are integrated to ensure that any disruptions during database communication or application runtime are gracefully managed, maintaining the application's stability.

This project serves as an excellent learning tool for individuals aspiring to deepen their understanding of building desktop-based e-commerce systems. It provides practical insights into Java programming, database integration with JDBC, GUI development using Swing, and system architecture design. By working through the project, learners can acquire essential skills in structuring scalable applications, managing backend operations, and creating user-friendly interfaces, all of which are critical for real-world software development.

Overall, this E-Commerce Application lays a strong foundation for further expansion, allowing future enhancements such as implementing user authentication, integrating payment gateways, applying advanced search features, or even migrating towards more modern technologies like JavaFX or web-based platforms.

4. ACTIVITY LOG

ACTIVITY LOG FOR THE WEEK

weeks	Description of the weekly activity	Learning Outcome	Person In-Charge Signature
Week–1	Java fundamentals: JVM architecture, data types, operators, control statements	Understood Java platform independence, wrote 10+ basic programs	
Week-2	OOPs concepts: Classes/Objects, Inheritance, Polymorphism (Method Overloading & Overriding)	Built Shape hierarchy (Circle, Rectangle) demonstrating polymorphism	
Week-3	Exception Handling: Checked/Unchecked exceptions, try-with-resources	Created file parser with custom "InvalidFormat Exception"	
Week-4	Collections Framework: ArrayList vs LinkedList, HashMap, Comparable/Comparator	Developed student gradebook using TreeMap with custom sorting	
Week-5	Multithreading: Thread lifecycle, synchronized blocks, volatile keyword	Implemented producer-consumer problem solution	
Week-6	File I/O: BufferedReader/Writer, Object Serialization using Serializable	Built employee record system with persistent storage	

Week-7	Java 8 Features: Lambda expressions, Stream API, Functional Interfaces	Refactored collections code using streams (filter, map, reduce)	

Week-8	Project Planning: Quiz application requirements, question bank design (100+ MCQs)	Finalized UML class diagram and project timeline	
Week-9	Core Engine Development: Question loader, scoring system, timer implementation	Completed base version with command-line interface	
Week-10	Advanced Features: 50-50 lifeline (Random class), skip question, difficulty-based scoring	Successfully implemented all game mechanics	
Week-11	Testing: JUnit test cases for question validation, exception scenarios	Achieved 85% code coverage, fixed boundary condition bugs	
Week-12	Deployment: Created executable JAR, user documentation, demo presentation	Delivered final product with complete source code and manual	

5. CONCLUSION

1. CONCLUSION

Reflection on the Internship Experience

My internship at Global Quest Technologies has been an enriching and transformative experience, providing me with invaluable exposure to real-world Java development, software engineering practices, and industry-standard technologies. Over the course of this internship, I had the opportunity to work on challenging projects, collaborate with experienced professionals, and enhance my technical and problem-solving skills.The internship began with an orientation and training phase, where I familiarized myself with the company's development environment, version control systems (such as Git), and Agile methodologies. This foundational knowledge was crucial in helping me adapt quickly to the team's workflow.

Key Learnings and Skills Acquired

During my internship, I worked on several Java-based projects, which helped me strengthen my understanding of:

Core Java Concepts – Object-Oriented Programming (OOP), Collections, Multithreading, Exception Handling, and File I/O operations.

Frameworks & Technologies – Introduction to Spring Boot, Hibernate, and RESTful APIs for backend development.

Database Management – Hands-on experience with MySQL for database design, querying, and optimization.

Development Tools – Proficiency in Eclipse/IntelliJ IDEA, Maven, and Postman for API testing.

Version Control & Collaboration – Effective use of Git and GitHub for team collaboration and code management.

Debugging & Problem-Solving – Learned how to troubleshoot errors, optimize code, and improve application performance.

Project Contributions

One of the major highlights of my internship was contributing to a real-world project,

Challenges Faced & Overcoming Them

While the internship was highly rewarding, I encountered several challenges, such as:

Understanding complex codebases – Initially, it was difficult to grasp large projects, but with

mentorship and consistent effort, I became more comfortable navigating them.

Debugging runtime errors – Learning to use debugging tools and logs effectively helped me resolve issues faster.

Adapting to Agile workflows – Daily stand-ups and sprint planning were new to me, but they improved my time management and collaboration skills.

Professional Growth & Soft Skills Development

Beyond technical skills, this internship helped me develop essential soft skills, including:

Communication – Interacting with team members, presenting my work, and documenting processes.

Time Management – Balancing multiple tasks and meeting deadlines in an Agile environment.

Teamwork – Collaborating with developers, testers, and project managers to deliver high-quality software.

Gratitude & Acknowledgement

I am deeply grateful to Global Quest Technologies for providing me with this incredible learning opportunity. Special thanks to my mentors and colleagues who guided me, shared their expertise, and helped me grow as a developer. The supportive work environment and constructive feedback have been instrumental in shaping my technical and professional abilities.

6. REFERENCES

2. REFERENCES

1. https://docs.oracle.com/en/java/
2. https://www.javatpoint.com/java-tutorial
3. https://spring.io/projects/spring-framework
4. https://hibernate.org/orm/documentation/
5. https://www.atlassian.com/git/tutorials
6. https://dev.mysql.com/doc/
7. https://www.baeldung.com/rest-api-best-practices
8. https://maven.apache.org/guides/getting-started/
9. https://www.geeksforgeeks.org/multithreading-in-java/
10. https://www.scrum.org/resources/scrum-guide