RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

ONLINE ELECTRONICS SHOPPING (E-COMMERCE WEBSITE)

A MINI PROJECT REPORT

Submitted by

KAVYASHRI Y	**231501076**
KISHORE KANNAN R	**231501078**
NISHA S	**231501112**

In partial fulfillment for the award of the degree of

BACHELOR OF

TECHNOLOGY IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105
2024 - 2025

BONAFIDE CERTIFICATE

Certified that this project report "**ONLINE ELECTRONICS SHOPPING**" is the Bonafide work of "**KAVYASHRI Y (231501076), KISHORE KANNAN R (231501078), NISHA S (231501112)"** who carried out the project work under my supervision**.**

Submitted for the Practical Examination held on____________________

SIGNATURE

Mr. U. Kumaran,

Assistant Professor (SS)
AIML,
Rajalakshmi Engineering College,
(Autonomous),
Thandalam, Chennai - 602 105

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The business-to-consumer aspect of electronic commerce (e-commerce) is the most visible business use of the World Wide Web. The primary goal of an e-commerce site is to sell goods and services online. This project deals with developing an e-commerce website 'Online Electronic Shopping'. It provides the user with a catalog of different electronic items available for purchase in the store. In order to facilitate online purchases, a shopping cart is provided to the user. After selection of the items, the user is forwarded to the Payment Gateway process. The system is implemented using a 3-tier approach, with a backend database, a middle tier consisting of Java, JDBC, Servlets, and a web browser as the front-end client. To develop an e-commerce website, a variety of technologies must be studied and applied. These include a multi-tiered architecture, server and client-side scripting techniques, implementation technologies such as Servlets, programming languages and tools such as Java , HTML, CSS, JavaScript, and Bootstrap, as well as relational databases like MySQL. This project was chosen with the objective of developing a basic e-commerce website where a consumer is provided with a shopping cart application and to explore the technologies used to develop such an application.

TABLE OF CONTENTS

I. INTRODUCTION

1.1 INTRODUCTION

Online Electronic shopping is the process whereby consumers directly buy goods, services etc. from a seller interactively in real-time without an intermediary service over the internet. Online shopping is the process of buying goods and services from merchants who sell on the Internet. Since the emergence of the World Wide Web, merchants have sought to sell their products to people who surf the Internet. Shoppers can visit web stores from the comfort of their homes and shop as they sit in front of the computer. Consumers buy a variety of items from online stores.

In this projects a user can visit the websites, registers and login to the website. They can check all the products available for shopping, filter and search item based on different categories, and then add to cart. They can add multiple item to the cart and also plus or minus the quantity in the cart. Once the cart is updated, the user can proceed to checkout and click the credit card payment details to proceed. Once the payment is success the orders will be placed and users will be able to see the orders details in the orders section along with the shipping status of the product.

The admin also plays an important role for this project as the admin is the one responsible for adding any product to the store, updating the items, removing the item from the store as well as managing the inventory. The admin can see all the product orders placed and also can mark them as shipped or delivered based on the conditions.

One of the best functionality that the projects include is mailing the customers, so once a user registers to the website, they will recieve a mail for the successful registration to the website, and along with that whenever a user orders any product or the product got shipped from the store, then the user will also receive the email for its confirmation. Sometimes, if the user tried to add any item which is out of stock, them they will get an email one the item is available again the stock.

1.2 OBJECTIVES

- **Primary Objectives**
    1. Create an Intuitive Shopping Platform
    2. Streamline the Checkout Process
    3. Efficient Order and Shipping Management
    4. Inventory Management and Product Availability

- **Business Objectives**
    1. Increase Sales and Revenue
    2. Enhance Customer Satisfaction and Retention
    3. Boost Operational Efficiency
    4. Ensure Data Security and Compliance

1.3 MODULES

Admin Module

- Login & Dashboard
- Product Management
    - Launch New Products
    - Fine-Tune Product Details
    - Retire Products Gracefully
    - Price Adjustments on the Fly
- Category Management
- Inventory Management
    - Stock product
    - Efficient Shipping Coordination
    - Sales Monitoring
- Email Maestro: Seamless Communication

User Experience Hub

- Login & Registration
- Product Browsing
    - Vibrant Product Displays
    - Smart Search & Quick Finds
    - Filter & Sort
- Shopping Cart
    - Add & Remove with Ease
    - Adjust Quantities Effortlessly
- Purchase processing
    - One-Click Checkout System
    - Payment Processing
    - Instant Order Receipts

Data Module

- User Vault: Managing User Information Securely
- Inventory Chronicles: Comprehensive Product Records
- Transaction Ledger: Tracking Every Sale
- Shipment Tracker: Delivering with Precision

Security Module

- Guarded Entry: User Authentication & Safety
- Session Safehouse: Manage User Activity with Care
- Fortified Data Highway: Secure Transmission
- Access Authority: Admin & User Control

These headings add a dynamic and engaging touch to your report while clearly describing each module's purpose and function.

II. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

ECLIPSE

Eclipse is written mostly in Java and its primary use is for developing Java applications,Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment. It is the second most-popular IDE for Java development, and, until 2016, was the most popular. Eclipse is written mostly in Java and its primary use is for developing Java applications.

Eclipse Enterprise Edition (EE) is a package for developers who work with Java and web applications. It includes tools for:

- Java
- JavaScript
- TypeScript
- JavaServer Pages and Faces
- Tomcat server
- Apache Maven
- Git

Eclipse EE is a version of Eclipse that comes with tools to make it easier to write server code. For example, you can compile and run a server by pressing the play button.

2.2 LANGUAGES

2.2.1 MySQL

MySQL is an open-source relational database management system (RDBMS). A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. SQL is a language that programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

2.2.2 JAVA

Java is a set of computer software and specifications that provides a software platform for developing application software and deploying it in a cross-platform computing environment. Java is used in a wide variety of computing platforms from embedded devices and mobile phones to enterprise servers and supercomputers. Java applets, which are less common than standalone Java applications, were commonly run in secure, sandboxed environments to provide many features of native applications through being embedded in HTML pages.

2.2.3 HTML

HTML, or Hypertext Markup Language, is the standard language used to create web pages. It defines the structure and content of web documents using tags and attributes to format text, embed images, create links, and build interactive elements. HTML facilitates communication between web browsers and servers, making it a crucial skill for web developers. HTML was invented by Tim Berners-Lee, a physicist at CERN, in 1990. His goal was to create a simple way to share and access documents over the Internet. Since its inception, HTML has evolved significantly, becoming the foundation of web development. When working with HTML, you use a simple code structure that includes tags and attributes to build the layout of a webpage.

2.2.4 CSS

CSS, which stands for Cascading Style Sheets, is a language in web development that enhances the presentation of HTML elements. By applying styles like color, layout, and spacing, CSS makes web pages visually appealing and responsive to various screen sizes. CSS is designed to enable the separation of content and presentation, including layout, colors, and fonts. This separation can improve content accessibility, since the content can be written without concern for its presentation; provide more flexibility and control in the specification of presentation characteristics; enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, which reduces complexity and repetition in the structural content; and enable the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

2.2.5 JAVASCRIPT

JavaScript, often abbreviated as JS, is a programming language and core technology of the Web, alongside HTML and CSS. 99% of websites use JavaScript on the client side for webpage behavior. JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard.

III. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

User Requirements

The system requirement in the online electronics shopping focuses on the ability to search product by the customer. It also includes user account management, viewing product details, and making purchases.

System Requirements

There should be a database backup of the online bookstore system. The operating system should be Windows 10 or a higher version of Windows.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

Software Requirements

- Operating System: Windows 10
- Front End: HTML, CSS, JavaScript
- Back End: Java, MySQL

Hardware Requirements

- Desktop PC or Laptop
- Printer (optional)
- Operating System: Windows 10
- Intel® Core™ i3-6006U CPU @ 2.00GHz or higher
- 4.00 GB RAM or higher
- 64-bit operating system, x64 based processor
- Monitor Resolution: 1024 x 768 or higher
- Keyboard and Mouse

3.3 DATA DICTIONARY

Product table

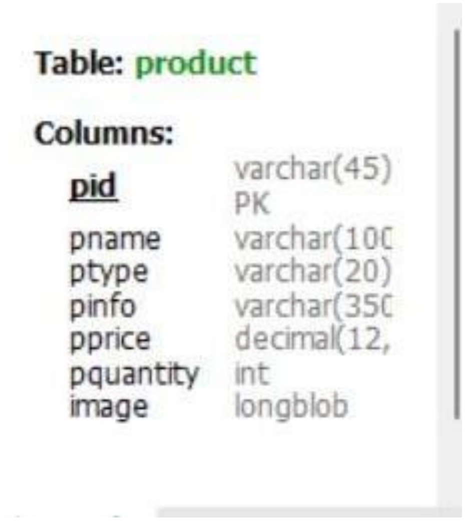


Table: product

Columns:

pid	varchar(45) PK
pname	varchar(10
ptype	varchar(20)
pinfo	varchar(35
pprice	decimal(12,
pquantity	int
image	longblob

User_demand table

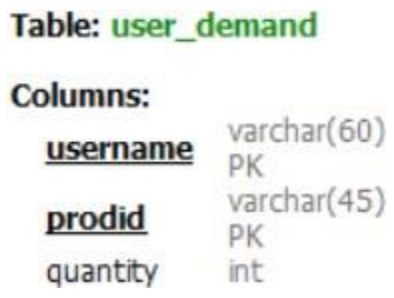


Table: user_demand

Columns:

username	varchar(60) PK
prodid	varchar(45) PK
quantity	int

Usercart table

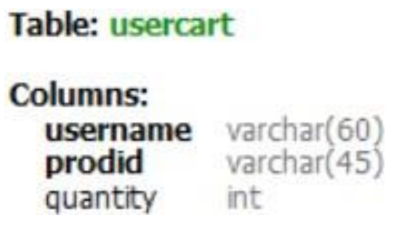
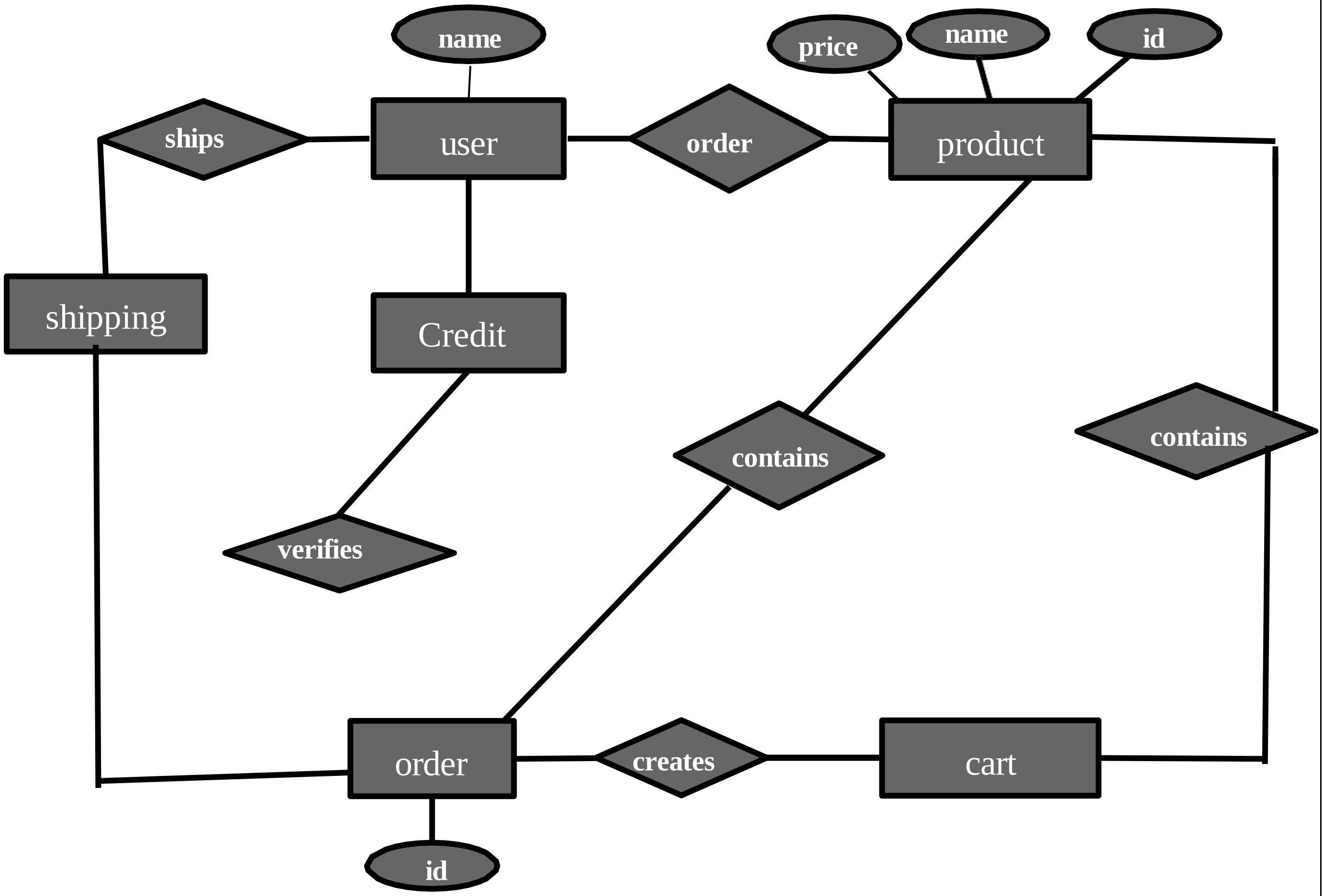


Table: usercart

Columns:

username	varchar(60)
prodid	varchar(45)
quantity	int

3.4 ER DIAGRAM



IV. PROGRAM CODE

DATABASE

```
CREATE TABLE IF NOT EXISTS `shopping-cart`.`product` (
  `pid` VARCHAR(45) NOT NULL,
  `pname` VARCHAR(100) NULL DEFAULT NULL,
  `ptype` VARCHAR(20) NULL DEFAULT NULL,
  `pinfo` VARCHAR(350) NULL DEFAULT NULL,
  `pprice` DECIMAL(12,2) NULL DEFAULT NULL,
  `pquantity` INT NULL DEFAULT NULL,
  `image` LONGBLOB NULL DEFAULT NULL,
  PRIMARY KEY (`pid`))


CREATE TABLE IF NOT EXISTS `shopping-cart`.`user` (
  `email` VARCHAR(60) NOT NULL,
  `name` VARCHAR(30) NULL DEFAULT NULL,
  `mobile` BIGINT NULL DEFAULT NULL,
  `address` VARCHAR(250) NULL DEFAULT NULL,
  `pincode` INT NULL DEFAULT NULL,
  `password` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`email`))


CREATE TABLE IF NOT EXISTS `shopping-cart`.`orders` (
  `orderid` VARCHAR(45) NOT NULL,
  `prodid` VARCHAR(45) NOT NULL,
  `quantity` INT NULL DEFAULT NULL,
  `amount` DECIMAL(10,2) NULL DEFAULT NULL,
  `shipped` INT NOT NULL DEFAULT 0,
  PRIMARY KEY (`orderid`, `prodid`),
  INDEX `productid_idx` (`prodid` ASC) VISIBLE,
  CONSTRAINT `productid`
    FOREIGN KEY (`prodid`)
    REFERENCES `shopping-cart`.`product` (`pid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

APPLICATION PROPERTIES

```
db.driverName = com.mysql.cj.jdbc.Driver
db.connectionString = jdbc:mysql://localhost:3306/shopping-cart
db.username = root
db.password = root

#Mailer Credentials
mailer.email=your_email
mailer.password=your_app_password_generated_from_email
```

ADD PRODUCT

```
package com.shashi.beans;

import java.io.InputStream;
import java.io.Serializable;

@SuppressWarnings("serial")
public class ProductBean implements Serializable {
	public ProductBean() {
	}

	private String prodId;
	private String prodName;
	private String prodType;
	private String prodInfo;
	private double prodPrice;
	private int prodQuantity;
	private InputStream prodImage;

	public ProductBean(String prodId, String prodName, String prodType, String prodInfo,
double prodPrice,
			int prodQuantity, InputStream prodImage) {
		super();
		this.prodId = prodId;
		this.prodName = prodName;
		this.prodType = prodType;
		this.prodInfo = prodInfo;
		this.prodPrice = prodPrice;
		this.prodQuantity = prodQuantity;
		this.prodImage = prodImage;
	}

	public String getProdId() {
		return prodId;
	}

	public void setProdId(String prodId) {
		this.prodId = prodId;
	}
```

```
public String getProdName() {
        return prodName;
}

public void setProdName(String prodName) {
        this.prodName = prodName;
}

public String getProdType() {
        return prodType;
}

public void setProdType(String prodType) {
        this.prodType = prodType;
}

public String getProdInfo() {
        return prodInfo;
}

public void setProdInfo(String prodInfo) {
        this.prodInfo = prodInfo;
}

public double getProdPrice() {
        return prodPrice;
}

public void setProdPrice(double prodPrice) {
        this.prodPrice = prodPrice;
}
```

```
	public int getProdQuantity() {
		return prodQuantity;
	}

	public void setProdQuantity(int prodQuantity) {
		this.prodQuantity = prodQuantity;
	}

	public InputStream getProdImage() {
		return prodImage;
	}

	public void setProdImage(InputStream prodImage) {
		this.prodImage = prodImage;
	}

}
```

CART DETAILS

```
package com.shashi.beans;

import java.io.Serializable;

@SuppressWarnings("serial")
public class CartBean implements Serializable {

	public CartBean() {
	}
```

```
public String userId;

public String prodId;

public int quantity;

public String getUserId() {
	return userId;
}

public void setUserId(String userId) {
	this.userId = userId;
}

public String getProdId() {
	return prodId;
}

public void setProdId(String prodId) {
	this.prodId = prodId;
}

public int getQuantity() {
	return quantity;
}

public void setQuantity(int quantity) {
	this.quantity = quantity;
}

public CartBean(String userId, String prodId, int quantity) {
		super();
		this.userId = userId;
		this.prodId = prodId;
		this.quantity = quantity;
	}

}
```

ORDER DETAILS

```
package com.shashi.beans;

import java.io.InputStream;
import java.io.Serializable;
import java.sql.Timestamp;

public class OrderDetails implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	private String orderId;
	private String productId;
	private String prodName;
	private String qty;
	private String amount;
	private int shipped;
	private Timestamp time;
private InputStream prodImage;

public String getOrderId() {
	return orderId;
}

public void setOrderId(String orderId) {
	this.orderId = orderId;
}

public String getProdName() {
	return prodName;
}

public void setProdName(String prodName) {
	this.prodName = prodName;
}

public String getQty() {
	return qty;
}

public void setQty(String qty) {
	this.qty = qty;
}

public String getAmount() {
	return amount;
}

public void setAmount(String amount) {
	this.amount = amount;
	}

	public Timestamp getTime() {
		return time;
	}

	public void setTime(Timestamp time) {
		this.time = time;
	}

	public InputStream getProdImage() {
		return prodImage;
	}

	public void setProdImage(InputStream prodImage) {
		this.prodImage = prodImage;
	}

	public String getProductId() {
		return productId;
	}

	public void setProductId(String productId) {
		this.productId = productId;
	}

	public int getShipped() {
		return shipped;
	}

	public void setShipped(int shipped) {
		this.shipped = shipped;
	}

}
```

PRODUCT DETAILS

```
package com.shashi.beans;

import java.io.InputStream;
import java.io.Serializable;

@SuppressWarnings("serial")
public class ProductBean implements Serializable {

	public ProductBean() {
	}

	private String prodId;
	private String prodName;
	private String prodType;
	private String prodInfo;
	private double prodPrice;
	private int prodQuantity;
	private InputStream prodImage;

	public ProductBean(String prodId, String prodName, String prodType, String prodInfo,
double prodPrice,
				int prodQuantity, InputStream prodImage) {
		super();
}
        this.prodId = prodId;
        this.prodName = prodName;
        this.prodType = prodType;
        this.prodInfo = prodInfo;
        this.prodPrice = prodPrice;
        this.prodQuantity = prodQuantity;
        this.prodImage = prodImage;
}

public String getProdId() {
        return prodId;
}

public void setProdId(String prodId) {
        this.prodId = prodId;
}

public String getProdName() {
        return prodName;
}

public void setProdName(String prodName) {
        this.prodName = prodName;
}

public String getProdType() {
        return prodType;
}

public void setProdType(String prodType) {
        this.prodType = prodType;
}
```

```
}

	public String getProdInfo() {
		return prodInfo;
	}

	public void setProdInfo(String prodInfo) {
		this.prodInfo = prodInfo;
	}

	public double getProdPrice() {
		return prodPrice;
	}

	public void setProdPrice(double prodPrice) {
		this.prodPrice = prodPrice;
	}

	public int getProdQuantity() {
		return prodQuantity;
	}

	public void setProdQuantity(int prodQuantity) {
		this.prodQuantity = prodQuantity;
	}

	public InputStream getProdImage() {
		return prodImage;
	}

	public void setProdImage(InputStream prodImage) {
		this.prodImage = prodImage;
	}
```

```
}
```

TRANSACTION DETAILS

```
package com.shashi.beans;

import java.io.Serializable;
import java.sql.Timestamp;
import java.text.SimpleDateFormat;

import com.shashi.utility.IDUtil;

@SuppressWarnings("serial")
public class TransactionBean implements Serializable {

	private String transactionId;

	private String userName;

	private Timestamp transDateTime;

	private double transAmount;

	public TransactionBean() {
		super();
		this.transactionId = IDUtil.generateTransId();

		SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-DD hh:mm:ss");
		Timestamp timestamp = new Timestamp(System.currentTimeMillis());

		sdf.format(timestamp);

		this.transDateTime = timestamp;
	}

	public TransactionBean(String userName, double transAmount) {
		super();
		this.userName = userName;
		this.transAmount = transAmount;

		this.transactionId = IDUtil.generateTransId();

		SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-DD hh:mm:ss");

		Timestamp timestamp = new Timestamp(System.currentTimeMillis());

		sdf.format(timestamp);

		this.transDateTime = timestamp;

	}

	public TransactionBean(String transactionId, String userName, double transAmount) {
		super();
		this.transactionId = transactionId;
		this.userName = userName;
		this.transAmount = transAmount;

		SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-DD hh:mm:ss");
		Timestamp timestamp = new Timestamp(System.currentTimeMillis());

		sdf.format(timestamp);

		this.transDateTime = timestamp;
	}

	public TransactionBean(String userName, Timestamp transDateTime, double transAmount) {
		super();
		this.userName = userName;
		this.transDateTime = transDateTime;
		this.transactionId = IDUtil.generateTransId();
		this.transAmount = transAmount;
	}

	public TransactionBean(String transactionId, String userName, Timestamp transDateTime,
double transAmount) {
		super();
		this.transactionId = transactionId;
		this.userName = userName;
		this.transDateTime = transDateTime;
		this.transAmount = transAmount;

	}

	public String getTransactionId() {
		return transactionId;
	}

	public void setTransactionId(String transactionId) {
		this.transactionId = transactionId;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}

	public Timestamp getTransDateTime() {
		return transDateTime;
	}

	public void setTransDateTime(Timestamp transDateTime) {
		this.transDateTime = transDateTime;
	}

	public double getTransAmount() {
		return transAmount;
	}

	public void setTransAmount(double transAmount) {
		this.transAmount = transAmount;
	}

}
```

USER DETAILS

```
package com.shashi.beans;

import java.io.Serializable;

@SuppressWarnings("serial")
public class UserBean implements Serializable {

	public UserBean() {
	}

	public UserBean(String userName, Long mobileNo, String emailId, String address, int
pinCode, String password) {
		super();
		this.name = userName;
		this.mobile = mobileNo;
		this.email = emailId;
		this.address = address;
		this.pinCode = pinCode;
		this.password = password;
	}

	private String name;
	private Long mobile;
	private String email;
	private String address;
	private int pinCode;
	private String password;

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

	public Long getMobile() {
		return mobile;
	}

	public void setMobile(Long mobile) {
		this.mobile = mobile;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}

	public int getPinCode() {
		return pinCode;
	}
	public void setPinCode(int pinCode) {
		this.pinCode = pinCode;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

}
```

DEMANDBEAN DETAILS

```
package com.shashi.beans;

import java.io.Serializable;

@SuppressWarnings("serial")
public class DemandBean implements Serializable {

	private String userName;
	private String prodId;
	private int demandQty;

	public DemandBean() {
		super();
	}
	}

	public DemandBean(String userName, String prodId, int demandQty) {
		super();
		this.userName = userName;
		this.prodId = prodId;
		this.demandQty = demandQty;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}

	public String getProdId() {
		return prodId;
	}

	public void setProdId(String prodId) {
		this.prodId = prodId;
	}

	public int getDemandQty() {
		return demandQty;
	}

	public void setDemandQty(int demandQty) {
		this.demandQty = demandQty;
	}

}
```

ORDERBEAN DETAILS

```
package com.shashi.beans;

import java.io.Serializable;

@SuppressWarnings("serial")
public class OrderBean implements Serializable {

	private String transactionId;
	private String productId;
	private int quantity;
	private Double amount;
	private int shipped;

	public OrderBean() {
		super();
	}

	public OrderBean(String transactionId, String productId, int quantity, Double amount) {
		super();
		this.transactionId = transactionId;
		this.productId = productId;
		this.quantity = quantity;
		this.amount = amount;
		this.shipped = 0;
	}

	public OrderBean(String transactionId, String productId, int quantity, Double amount, int
shipped) {
		super();
```

```
        this.transactionId = transactionId;
        this.productId = productId;
        this.quantity = quantity;
        this.amount = amount;
        this.shipped = shipped;
    }

    public String getTransactionId() {
        return transactionId;
    }

    public void setTransactionId(String transactionId) {
        this.transactionId = transactionId;
    }

    public String getProductId() {
        return productId;
    }

    public void setProductId(String productId) {
        this.productId = productId;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public Double getAmount() {
```

```
		return amount;
	}

	public void setAmount(Double amount) {
		this.amount = amount;
	}

	public int getShipped() {
		return shipped;
	}

	public void setShipped(int shipped) {
		this.shipped = shipped;
	}

}
```

USER CONSTANTS

```
package com.shashi.constants;

public interface IUserConstants {

	public String TABLE_USER = "user";
	public String COLUMN_NAME = "name";
	public String COLUMN_MOBILE = "mobile";
	public String COLUMN_EMAIL = "email";
	public String COLUMN_ADDRESS = "address";
	public String COLUMN_PINCODE = "pincode";
	public String COLUMN_PASSWORD = "password";
}
```

CART SERVICE IMPLEMENTATION

```
package com.shashi.service.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import com.shashi.beans.CartBean;
import com.shashi.beans.DemandBean;
import com.shashi.beans.ProductBean;
import com.shashi.service.CartService;
import com.shashi.utility.DBUtil;

public class CartServiceImpl implements CartService {

        @Override
        public String addProductToCart(String userId, String prodId, int prodQty) {
                String status = "Failed to Add into Cart";

                Connection con = DBUtil.provideConnection();

                PreparedStatement ps = null;
                PreparedStatement ps2 = null;
                ResultSet rs = null;

                try {
```

```
                ps = con.prepareStatement("select * from usercart where username=? and
prodid=?");

                ps.setString(1, userId);
                ps.setString(2, prodId);

                rs = ps.executeQuery();

                if (rs.next()) {

                    int cartQuantity = rs.getInt("quantity");

                    ProductBean product = new
ProductServiceImpl().getProductDetails(prodId);

                    int availableQty = product.getProdQuantity();

                    prodQty += cartQuantity;
                    //
                    if (availableQty < prodQty) {

                        status = updateProductToCart(userId, prodId, availableQty);

                        status = "Only " + availableQty + " no of " +
product.getProdName()
                                + " are available in the shop! So we are adding
only " + availableQty
                                + " no of that item into Your Cart" + "";

                        DemandBean demandBean = new DemandBean(userId,
product.getProdId(), prodQty - availableQty);
```

```
                                        DemandServiceImpl demand = new DemandServiceImpl();

                                        boolean flag = demand.addProduct(demandBean);

                                        if (flag)
                                                status += "<br/>Later, We Will Mail You when " +
product.getProdName()
                                                                + " will be available into the Store!";

                                } else {
                                        status = updateProductToCart(userId, prodId, prodQty);

                                }
                        }

                } catch (SQLException e) {
                        status = "Error: " + e.getMessage();
                        e.printStackTrace();
                }

                DBUtil.closeConnection(con);
                DBUtil.closeConnection(ps);
                DBUtil.closeConnection(rs);
                DBUtil.closeConnection(ps2);

                return status;
        }

        @Override
        public List<CartBean> getAllCartItems(String userId) {
                List<CartBean> items = new ArrayList<CartBean>();
```

```
Connection con = DBUtil.provideConnection();

PreparedStatement ps = null;
ResultSet rs = null;

try {

        ps = con.prepareStatement("select * from usercart where username=?");

        ps.setString(1, userId);

        rs = ps.executeQuery();

        while (rs.next()) {
                CartBean cart = new CartBean();

                cart.setUserId(rs.getString("username"));
                cart.setProdId(rs.getString("prodid"));
                cart.setQuantity(Integer.parseInt(rs.getString("quantity")));

                items.add(cart);

        }

} catch (SQLException e) {

        e.printStackTrace();
}

DBUtil.closeConnection(con);
DBUtil.closeConnection(ps);
DBUtil.closeConnection(rs);
```

```
		return items;
	}

	@Override
	public int getCartCount(String userId) {
		int count = 0;

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;

		ResultSet rs = null;

		try {
			ps = con.prepareStatement("select sum(quantity) from usercart where
username=?");

			ps.setString(1, userId);

			rs = ps.executeQuery();

			if (rs.next() && !rs.wasNull())
				count = rs.getInt(1);

		} catch (SQLException e) {

			e.printStackTrace();
		}

		DBUtil.closeConnection(con);
		DBUtil.closeConnection(ps);
		DBUtil.closeConnection(rs);

		return count;
	}

	@Override
	public String removeProductFromCart(String userId, String prodId) {
		String status = "Product Removal Failed";

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;
		PreparedStatement ps2 = null;
		ResultSet rs = null;

		try {

			ps = con.prepareStatement("select * from usercart where username=? and
prodid=?");

			ps.setString(1, userId);
			ps.setString(2, prodId);

			rs = ps.executeQuery();

			if (rs.next()) {

				int prodQuantity = rs.getInt("quantity");

				prodQuantity -= 1;

				if (prodQuantity > 0) {
```

```
                        ps2 = con.prepareStatement("update usercart set quantity=?
where username=? and prodid=?");

                        ps2.setInt(1, prodQuantity);

                        ps2.setString(2, userId);

                        ps2.setString(3, prodId);

                        int k = ps2.executeUpdate();

                        if (k > 0)
                            status = "Product Successfully removed from the Cart!";
                    } else if (prodQuantity <= 0) {

                        ps2 = con.prepareStatement("delete from usercart where
username=? and prodid=?");

                        ps2.setString(1, userId);

                        ps2.setString(2, prodId);

                        int k = ps2.executeUpdate();

                        if (k > 0)
                            status = "Product Successfully removed from the Cart!";
                    }

                } else {

                    status = "Product Not Available in the cart!";
```

```
				}

			} catch (SQLException e) {
				status = "Error: " + e.getMessage();
				e.printStackTrace();
			}

			DBUtil.closeConnection(con);
			DBUtil.closeConnection(ps);
			DBUtil.closeConnection(rs);
			DBUtil.closeConnection(ps2);

			return status;
		}

		@Override
		public boolean removeAProduct(String userId, String prodId) {
			boolean flag = false;

			Connection con = DBUtil.provideConnection();

			PreparedStatement ps = null;
			ResultSet rs = null;

			try {

				ps = con.prepareStatement("delete from usercart where username=? and
prodid=?");
				ps.setString(1, userId);
				ps.setString(2, prodId);

				int k = ps.executeUpdate();
```

```
                if (k > 0)
                    flag = true;

            } catch (SQLException e) {
                flag = false;
                e.printStackTrace();
            }

            DBUtil.closeConnection(con);
            DBUtil.closeConnection(ps);
            DBUtil.closeConnection(rs);

            return flag;
        }

        @Override
        public String updateProductToCart(String userId, String prodId, int prodQty) {

            String status = "Failed to Add into Cart";

            Connection con = DBUtil.provideConnection();

            PreparedStatement ps = null;
            PreparedStatement ps2 = null;
            ResultSet rs = null;

            try {

                ps = con.prepareStatement("select * from usercart where username=? and
prodid=?");
```

```
				ps.setString(1, userId);
				ps.setString(2, prodId);

				rs = ps.executeQuery();

				if (rs.next()) {

					if (prodQty > 0) {
						ps2 = con.prepareStatement("update usercart set quantity=?
where username=? and prodid=?");

						ps2.setInt(1, prodQty);

						ps2.setString(2, userId);

						ps2.setString(3, prodId);

						int k = ps2.executeUpdate();

						if (k > 0)
							status = "Product Successfully Updated to Cart!";
					} else if (prodQty == 0) {
						ps2 = con.prepareStatement("delete from usercart where
username=? and prodid=?");

						ps2.setString(1, userId);

						ps2.setString(2, prodId);

						int k = ps2.executeUpdate();

						if (k > 0)
```

```
                                        status = "Product Successfully Updated in Cart!";
                              }
                    } else {

                              ps2 = con.prepareStatement("insert into usercart values(?,?,?)");

                              ps2.setString(1, userId);

                              ps2.setString(2, prodId);

                              ps2.setInt(3, prodQty);

                              int k = ps2.executeUpdate();

                              if (k > 0)
                                        status = "Product Successfully Updated to Cart!";

                    }

          } catch (SQLException e) {
                    status = "Error: " + e.getMessage();
                    e.printStackTrace();
          }

          DBUtil.closeConnection(con);
          DBUtil.closeConnection(ps);
          DBUtil.closeConnection(rs);
          DBUtil.closeConnection(ps2);

          return status;
}
```

```
	public int getProductCount(String userId, String prodId) {
		int count = 0;

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;
		ResultSet rs = null;

		try {
			ps = con.prepareStatement("select sum(quantity) from usercart where 
username=? and prodid=?");
			ps.setString(1, userId);
			ps.setString(2, prodId);
			rs = ps.executeQuery();

			if (rs.next() && !rs.wasNull())
				count = rs.getInt(1);

		} catch (SQLException e) {
			e.printStackTrace();
		}

		return count;
	}

	@Override
	public int getCartItemCount(String userId, String itemId) {
		int count = 0;
		if (userId == null || itemId == null)
			return 0;
		Connection con = DBUtil.provideConnection();
		PreparedStatement ps = null;

		ResultSet rs = null;

		try {
			ps = con.prepareStatement("select quantity from usercart where username=?
and prodid=?");

			ps.setString(1, userId);
			ps.setString(2, itemId);

			rs = ps.executeQuery();

			if (rs.next() && !rs.wasNull())
				count = rs.getInt(1);

		} catch (SQLException e) {

			e.printStackTrace();
		}

		DBUtil.closeConnection(con);
		DBUtil.closeConnection(ps);
		DBUtil.closeConnection(rs);

		return count;
	}
}
```

DEMAND SERVICE IMPLEMENTATION

```
package com.shashi.service.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import com.shashi.beans.DemandBean;
import com.shashi.service.DemandService;
import com.shashi.utility.DBUtil;
//This class is to process the demand items which are
//not available at the time of purchase by any customer
//the customer will receive mail once the product is avaible
//back into the store
public class DemandServiceImpl implements DemandService {

	@Override
	public boolean addProduct(String userId, String prodId, int demandQty) {
		boolean flag = false;

		//get the database connection
		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;
		PreparedStatement ps2 = null;
		ResultSet rs = null;

		try {
			//create the prepared statement with the query
			ps = con.prepareStatement("select * from user_demand where username=? and
prodid=?");

                        ps.setString(1, userId);
                        ps.setString(2, prodId);

                        rs = ps.executeQuery();

                        if (rs.next()) {

                                flag = true;
                        } else {
                                ps2 = con.prepareStatement("insert into  user_demand values(?,?,?)");

                                ps2.setString(1, userId);

                                ps2.setString(2, prodId);

                                ps2.setInt(3, demandQty);

                                int k = ps2.executeUpdate();

                                if (k > 0)
                                        flag = true;
                        }

                } catch (SQLException e) {
                        flag = false;
                        e.printStackTrace();
                }

                DBUtil.closeConnection(con);
                DBUtil.closeConnection(ps);
```

```
		DBUtil.closeConnection(ps2);
		DBUtil.closeConnection(rs);
		//return true if the product is added into the db
		return flag;
	}

	@Override
	public boolean removeProduct(String userId, String prodId) {
		boolean flag = false;

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;
		PreparedStatement ps2 = null;
		ResultSet rs = null;

		try {
			ps = con.prepareStatement("select * from user_demand where username=? and
prodid=?");

			ps.setString(1, userId);
			ps.setString(2, prodId);

			rs = ps.executeQuery();

			// System.out.println("userId "+userId+"\nprodId: "+prodId);

			if (rs.next()) {

				// System.out.println("userId "+userId+"\nprodId: "+prodId);
				ps2 = con.prepareStatement("delete from  user_demand where
username=? and prodid=?");
```

```
                    ps2.setString(1, userId);

                    ps2.setString(2, prodId);

                    int k = ps2.executeUpdate();

                    if (k > 0)
                        flag = true;

                } else {
                    flag = true;
                }

            } catch (SQLException e) {
                flag = false;
                e.printStackTrace();
            }

            DBUtil.closeConnection(con);
            DBUtil.closeConnection(ps);
            DBUtil.closeConnection(ps2);
            DBUtil.closeConnection(rs);

            return flag;
        }

        @Override
        public boolean addProduct(DemandBean userDemandBean) {

            return addProduct(userDemandBean.getUserName(), userDemandBean.getProdId(),
    userDemandBean.getDemandQty());
```

```
	}

	@Override
	public List<DemandBean> haveDemanded(String prodId) {
		List<DemandBean> demandList = new ArrayList<DemandBean>();

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;
		ResultSet rs = null;

		try {
			ps = con.prepareStatement("select * from user_demand where prodid=?");
			ps.setString(1, prodId);
			rs = ps.executeQuery();

			while (rs.next()) {

				DemandBean demand = new DemandBean(rs.getString("username"),
rs.getString("prodid"),
						rs.getInt("quantity"));

				demandList.add(demand);

			}

		} catch (SQLException e) {

			e.printStackTrace();
		}

		return demandList;
	}

}
```

ORDER SERVICE IMPLEMENTATION

```
package com.shashi.service.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import com.shashi.beans.CartBean;
import com.shashi.beans.OrderBean;
import com.shashi.beans.OrderDetails;
import com.shashi.beans.TransactionBean;
import com.shashi.service.OrderService;
import com.shashi.utility.DBUtil;
import com.shashi.utility.MailMessage;

public class OrderServiceImpl implements OrderService {

	@Override
	public String paymentSuccess(String userName, double paidAmount) {
		String status = "Order Placement Failed!";

		List<CartBean> cartItems = new ArrayList<CartBean>();
		cartItems = new CartServiceImpl().getAllCartItems(userName);
		if (cartItems.size() == 0)
			return status;

		TransactionBean transaction = new TransactionBean(userName, paidAmount);
		boolean ordered = false;

		String transactionId = transaction.getTransactionId();

		// System.out.println("Transaction: "+transaction.getTransactionId()+"
		// "+transaction.getTransAmount()+" "+transaction.getUserName()+"
		// "+transaction.getTransDateTime());

		for (CartBean item : cartItems) {

			double amount = new ProductServiceImpl().getProductPrice(item.getProdId())
* item.getQuantity();

			OrderBean order = new OrderBean(transactionId, item.getProdId(),
item.getQuantity(), amount);

			ordered = addOrder(order);
			if (!ordered)
				break;
			else {
				ordered = new CartServiceImpl().removeAProduct(item.getUserId(),
item.getProdId());
			}

			if (!ordered)
				break;
			else
```

```
                    ordered = new ProductServiceImpl().sellNProduct(item.getProdId(),
item.getQuantity());

                if (!ordered)
                    break;
            }

            if (ordered) {
                ordered = new OrderServiceImpl().addTransaction(transaction);
                if (ordered) {

                    MailMessage.transactionSuccess(userName, new
UserServiceImpl().getFName(userName),
                            transaction.getTransactionId(),
transaction.getTransAmount());

                    status = "Order Placed Successfully!";
                }
            }

            return status;
        }

        @Override
        public boolean addOrder(OrderBean order) {
            boolean flag = false;

            Connection con = DBUtil.provideConnection();

            PreparedStatement ps = null;

            try {
```

```
            ps = con.prepareStatement("insert into orders values(?,?,?,?,?)");

            ps.setString(1, order.getTransactionId());
            ps.setString(2, order.getProductId());
            ps.setInt(3, order.getQuantity());
            ps.setDouble(4, order.getAmount());
            ps.setInt(5, 0);

            int k = ps.executeUpdate();

            if (k > 0)
                flag = true;

        } catch (SQLException e) {
            flag = false;
            e.printStackTrace();
        }

        return flag;
    }

    @Override
    public boolean addTransaction(TransactionBean transaction) {
        boolean flag = false;

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;

        try {
            ps = con.prepareStatement("insert into transactions values(?,?,?,?)");
```

```
                    ps.setString(1, transaction.getTransactionId());
                    ps.setString(2, transaction.getUserName());
                    ps.setTimestamp(3, transaction.getTransDateTime());
                    ps.setDouble(4, transaction.getTransAmount());

                    int k = ps.executeUpdate();

                    if (k > 0)
                        flag = true;

                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

                return flag;
        }

        @Override
        public int countSoldItem(String prodId) {
                int count = 0;

                Connection con = DBUtil.provideConnection();

                PreparedStatement ps = null;

                ResultSet rs = null;

                try {
                    ps = con.prepareStatement("select sum(quantity) from orders where
prodid=?");
```

```
                ps.setString(1, prodId);

                rs = ps.executeQuery();

                if (rs.next())
                        count = rs.getInt(1);

        } catch (SQLException e) {
                count = 0;
                e.printStackTrace();
        }

        DBUtil.closeConnection(con);
        DBUtil.closeConnection(ps);
        DBUtil.closeConnection(rs);

        return count;
}

@Override
public List<OrderBean> getAllOrders() {
        List<OrderBean> orderList = new ArrayList<OrderBean>();

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;
        ResultSet rs = null;

        try {

                ps = con.prepareStatement("select * from orders");
```

```
			rs = ps.executeQuery();

			while (rs.next()) {

				OrderBean order = new OrderBean(rs.getString("orderid"),
rs.getString("prodid"), rs.getInt("quantity"),
						rs.getDouble("amount"), rs.getInt("shipped"));

				orderList.add(order);

			}

		} catch (SQLException e) {

			e.printStackTrace();
		}

		return orderList;
	}

	@Override
	public List<OrderBean> getOrdersByUserId(String emailId) {
		List<OrderBean> orderList = new ArrayList<OrderBean>();

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;
		ResultSet rs = null;

		try {

			ps = con.prepareStatement(
                                        "SELECT * FROM orders o inner join transactions t on
o.orderid = t.transid where username=?");
                        ps.setString(1, emailId);
                        rs = ps.executeQuery();

                        while (rs.next()) {

                                OrderBean order = new OrderBean(rs.getString("t.transid"),
rs.getString("t.prodid"),
                                                rs.getInt("quantity"), rs.getDouble("t.amount"),
rs.getInt("shipped"));

                                orderList.add(order);

                        }

                } catch (SQLException e) {

                        e.printStackTrace();
                }

                return orderList;
        }

        @Override
        public List<OrderDetails> getAllOrderDetails(String userEmailId) {
                List<OrderDetails> orderList = new ArrayList<OrderDetails>();

                Connection con = DBUtil.provideConnection();

                PreparedStatement ps = null;
                ResultSet rs = null;
```

```
		try {

			ps = con.prepareStatement(
					"SELECT  p.pid as prodid, o.orderid as orderid, o.shipped as
shipped, p.image as image, p.pname as pname, o.quantity as qty, o.amount as amount, t.time as time
FROM orders o, product p, transactions t where o.orderid=t.transid and o.orderid = t.transid and
p.pid=o.prodid and t.username=?");
			ps.setString(1, userEmailId);
			rs = ps.executeQuery();

			while (rs.next()) {

				OrderDetails order = new OrderDetails();
				order.setOrderId(rs.getString("orderid"));
				order.setProdImage(rs.getAsciiStream("image"));
				order.setProdName(rs.getString("pname"));
				order.setQty(rs.getString("qty"));
				order.setAmount(rs.getString("amount"));
				order.setTime(rs.getTimestamp("time"));
				order.setProductId(rs.getString("prodid"));
				order.setShipped(rs.getInt("shipped"));
				orderList.add(order);

			}

		} catch (SQLException e) {

			e.printStackTrace();
		}

		return orderList;
```

```
	}

	@Override
	public String shipNow(String orderId, String prodId) {
		String status = "FAILURE";

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;

		try {
			ps = con.prepareStatement("update orders set shipped=1 where orderid=? and
prodid=? and shipped=0");

			ps.setString(1, orderId);
			ps.setString(2, prodId);

			int k = ps.executeUpdate();

			if (k > 0) {
				status = "Order Has been shipped successfully!!";
			}

		} catch (SQLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		DBUtil.closeConnection(con);
		DBUtil.closeConnection(ps);

		return status;
	}

}
```

PRODUCT SERVICE IMPLEMENTATION

```
package com.shashi.service.impl;

import java.io.InputStream;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import com.shashi.beans.DemandBean;
import com.shashi.beans.ProductBean;
import com.shashi.service.ProductService;
import com.shashi.utility.DBUtil;
import com.shashi.utility.IDUtil;
import com.shashi.utility.MailMessage;

public class ProductServiceImpl implements ProductService {

	@Override
	public String addProduct(String prodName, String prodType, String prodInfo, double
prodPrice, int prodQuantity,
			InputStream prodImage) {
		String status = null;
		String prodId = IDUtil.generateId();
		ProductBean product = new ProductBean(prodId, prodName, prodType, prodInfo,
prodPrice, prodQuantity, prodImage);

		status = addProduct(product);

		return status;
	}

	@Override
	public String addProduct(ProductBean product) {
		String status = "Product Registration Failed!";

		if (product.getProdId() == null)
			product.setProdId(IDUtil.generateId());

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;

		try {
			ps = con.prepareStatement("insert into product values(?,?,?,?,?,?,?);");
			ps.setString(1, product.getProdId());
			ps.setString(2, product.getProdName());
			ps.setString(3, product.getProdType());
			ps.setString(4, product.getProdInfo());
			ps.setDouble(5, product.getProdPrice());
			ps.setInt(6, product.getProdQuantity());
			ps.setBlob(7, product.getProdImage());

			int k = ps.executeUpdate();
```

```
				if (k > 0) {

					status = "Product Added Successfully with Product Id: " +
product.getProdId();

				} else {

					status = "Product Updation Failed!";
				}

			} catch (SQLException e) {
				status = "Error: " + e.getMessage();
				e.printStackTrace();
			}

			DBUtil.closeConnection(con);
			DBUtil.closeConnection(ps);

			return status;
		}

		@Override
		public String removeProduct(String prodId) {
			String status = "Product Removal Failed!";

			Connection con = DBUtil.provideConnection();

			PreparedStatement ps = null;
			PreparedStatement ps2 = null;

			try {
				ps = con.prepareStatement("delete from product where pid=?");
				ps.setString(1, prodId);

				int k = ps.executeUpdate();

				if (k > 0) {
					status = "Product Removed Successfully!";

					ps2 = con.prepareStatement("delete from usercart where prodid=?");

					ps2.setString(1, prodId);

					ps2.executeUpdate();

				}

			} catch (SQLException e) {
				status = "Error: " + e.getMessage();
				e.printStackTrace();
			}

			DBUtil.closeConnection(con);
			DBUtil.closeConnection(ps);
			DBUtil.closeConnection(ps2);

			return status;
		}

		@Override
		public String updateProduct(ProductBean prevProduct, ProductBean updatedProduct) {
			String status = "Product Updation Failed!";

			if (!prevProduct.getProdId().equals(updatedProduct.getProdId())) {
```

```
                status = "Both Products are Different, Updation Failed!";

                return status;
            }

            Connection con = DBUtil.provideConnection();

            PreparedStatement ps = null;

            try {
                ps = con.prepareStatement(
                        "update product set
pname=?,ptype=?,pinfo=?,pprice=?,pquantity=?,image=? where pid=?");

                ps.setString(1, updatedProduct.getProdName());
                ps.setString(2, updatedProduct.getProdType());
                ps.setString(3, updatedProduct.getProdInfo());
                ps.setDouble(4, updatedProduct.getProdPrice());
                ps.setInt(5, updatedProduct.getProdQuantity());
                ps.setBlob(6, updatedProduct.getProdImage());
                ps.setString(7, prevProduct.getProdId());

                int k = ps.executeUpdate();

                if (k > 0)
                    status = "Product Updated Successfully!";

            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
```

```
        DBUtil.closeConnection(con);
        DBUtil.closeConnection(ps);

        return status;
    }

    @Override
    public String updateProductPrice(String prodId, double updatedPrice) {
        String status = "Price Updation Failed!";

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;

        try {
            ps = con.prepareStatement("update product set pprice=? where pid=?");

            ps.setDouble(1, updatedPrice);
            ps.setString(2, prodId);

            int k = ps.executeUpdate();

            if (k > 0)
                status = "Price Updated Successfully!";
        } catch (SQLException e) {
            status = "Error: " + e.getMessage();
            e.printStackTrace();
        }

        DBUtil.closeConnection(con);
        DBUtil.closeConnection(ps);
```

```
		return status;
	}

	@Override
	public List<ProductBean> getAllProducts() {
		List<ProductBean> products = new ArrayList<ProductBean>();

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;
		ResultSet rs = null;

		try {
			ps = con.prepareStatement("select * from product");

			rs = ps.executeQuery();

			while (rs.next()) {

				ProductBean product = new ProductBean();

				product.setProdId(rs.getString(1));
				product.setProdName(rs.getString(2));
				product.setProdType(rs.getString(3));
				product.setProdInfo(rs.getString(4));
				product.setProdPrice(rs.getDouble(5));
				product.setProdQuantity(rs.getInt(6));
				product.setProdImage(rs.getAsciiStream(7));

				products.add(product);
```

```
                }

            } catch (SQLException e) {
                e.printStackTrace();
            }

            DBUtil.closeConnection(con);
            DBUtil.closeConnection(ps);
            DBUtil.closeConnection(rs);

            return products;
        }

        @Override
        public List<ProductBean> getAllProductsByType(String type) {
            List<ProductBean> products = new ArrayList<ProductBean>();

            Connection con = DBUtil.provideConnection();

            PreparedStatement ps = null;
            ResultSet rs = null;

            try {
                ps = con.prepareStatement("SELECT * FROM `shopping-cart`.product where
lower(ptype) like ?;");
                ps.setString(1, "%" + type + "%");
                rs = ps.executeQuery();

                while (rs.next()) {

                    ProductBean product = new ProductBean();
```

```
                        product.setProdId(rs.getString(1));
                        product.setProdName(rs.getString(2));
                        product.setProdType(rs.getString(3));
                        product.setProdInfo(rs.getString(4));
                        product.setProdPrice(rs.getDouble(5));
                        product.setProdQuantity(rs.getInt(6));
                        product.setProdImage(rs.getAsciiStream(7));

                        products.add(product);

                }

        } catch (SQLException e) {
                e.printStackTrace();
        }

        DBUtil.closeConnection(con);
        DBUtil.closeConnection(ps);
        DBUtil.closeConnection(rs);

        return products;
}

@Override
public List<ProductBean> searchAllProducts(String search) {
        List<ProductBean> products = new ArrayList<ProductBean>();

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;
        ResultSet rs = null;
```

```
		try {
			ps = con.prepareStatement(
					"SELECT * FROM `shopping-cart`.product where lower(ptype) 
like ? or lower(pname) like ? or lower(pinfo) like ?");
			search = "%" + search + "%";
			ps.setString(1, search);
			ps.setString(2, search);
			ps.setString(3, search);
			rs = ps.executeQuery();

			while (rs.next()) {

				ProductBean product = new ProductBean();

				product.setProdId(rs.getString(1));
				product.setProdName(rs.getString(2));
				product.setProdType(rs.getString(3));
				product.setProdInfo(rs.getString(4));
				product.setProdPrice(rs.getDouble(5));
				product.setProdQuantity(rs.getInt(6));
				product.setProdImage(rs.getAsciiStream(7));

				products.add(product);

			}

		} catch (SQLException e) {
			e.printStackTrace();
		}

		DBUtil.closeConnection(con);
		DBUtil.closeConnection(ps);
		DBUtil.closeConnection(rs);

		return products;
	}

	@Override
	public byte[] getImage(String prodId) {
		byte[] image = null;

		Connection con = DBUtil.provideConnection();

		PreparedStatement ps = null;
		ResultSet rs = null;

		try {
			ps = con.prepareStatement("select image from product where  pid=?");

			ps.setString(1, prodId);

			rs = ps.executeQuery();

			if (rs.next())
				image = rs.getBytes("image");

		} catch (SQLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		DBUtil.closeConnection(con);
		DBUtil.closeConnection(ps);
		DBUtil.closeConnection(rs);
```

```
        return image;
    }

    @Override
    public ProductBean getProductDetails(String prodId) {
        ProductBean product = null;

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
            ps = con.prepareStatement("select * from product where pid=?");

            ps.setString(1, prodId);
            rs = ps.executeQuery();

            if (rs.next()) {
                product = new ProductBean();
                product.setProdId(rs.getString(1));
                product.setProdName(rs.getString(2));
                product.setProdType(rs.getString(3));
                product.setProdInfo(rs.getString(4));
                product.setProdPrice(rs.getDouble(5));
                product.setProdQuantity(rs.getInt(6));
                product.setProdImage(rs.getAsciiStream(7));
            }

        } catch (SQLException e) {
            // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            }

            DBUtil.closeConnection(con);
            DBUtil.closeConnection(ps);

            return product;
        }

        @Override
        public String updateProductWithoutImage(String prevProductId, ProductBean
updatedProduct) {
            String status = "Product Updation Failed!";

            if (!prevProductId.equals(updatedProduct.getProdId())) {

                status = "Both Products are Different, Updation Failed!";

                return status;
            }

            int prevQuantity = new ProductServiceImpl().getProductQuantity(prevProductId);
            Connection con = DBUtil.provideConnection();

            PreparedStatement ps = null;

            try {
                ps = con.prepareStatement("update product set
pname=?,ptype=?,pinfo=?,pprice=?,pquantity=? where pid=?");

                ps.setString(1, updatedProduct.getProdName());
                ps.setString(2, updatedProduct.getProdType());
```

```
                        ps.setString(3, updatedProduct.getProdInfo());
                        ps.setDouble(4, updatedProduct.getProdPrice());
                        ps.setInt(5, updatedProduct.getProdQuantity());
                        ps.setString(6, prevProductId);

                        int k = ps.executeUpdate();
                        // System.out.println("prevQuantity: "+prevQuantity);
                        if ((k > 0) && (prevQuantity < updatedProduct.getProdQuantity())) {
                             status = "Product Updated Successfully!";
                             // System.out.println("updated!");
                             List<DemandBean> demandList = new
DemandServiceImpl().haveDemanded(prevProductId);

                             for (DemandBean demand : demandList) {

                                  String userFName = new
UserServiceImpl().getFName(demand.getUserName());
                                  try {

     MailMessage.productAvailableNow(demand.getUserName(), userFName,
updatedProduct.getProdName(),
                                                 prevProductId);
                                  } catch (Exception e) {
                                       System.out.println("Mail Sending Failed: " +
e.getMessage());
                                  }
                                  boolean flag = new
DemandServiceImpl().removeProduct(demand.getUserName(), prevProductId);

                                  if (flag)
                                       status += " And Mail Send to the customers who were
waiting for this product!";
```

```
                        }
                } else if (k > 0)
                        status = "Product Updated Successfully!";
                else
                        status = "Product Not available in the store!";

        } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

        DBUtil.closeConnection(con);
        DBUtil.closeConnection(ps);
        // System.out.println("Prod Update status : "+status);

        return status;
}

@Override
public double getProductPrice(String prodId) {
        double price = 0;

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
                ps = con.prepareStatement("select * from product where pid=?");

                ps.setString(1, prodId);
                rs = ps.executeQuery();
```

```
				if (rs.next()) {
					price = rs.getDouble("pprice");
				}

			} catch (SQLException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

			DBUtil.closeConnection(con);
			DBUtil.closeConnection(ps);

			return price;
		}

		@Override
		public boolean sellNProduct(String prodId, int n) {
			boolean flag = false;

			Connection con = DBUtil.provideConnection();

			PreparedStatement ps = null;

			try {

				ps = con.prepareStatement("update product set pquantity=(pquantity - ?) where 
pid=?");

				ps.setInt(1, n);

				ps.setString(2, prodId);
```

```
                int k = ps.executeUpdate();

                if (k > 0)
                        flag = true;
        } catch (SQLException e) {
                flag = false;
                e.printStackTrace();
        }

        DBUtil.closeConnection(con);
        DBUtil.closeConnection(ps);

        return flag;
}

@Override
public int getProductQuantity(String prodId) {

        int quantity = 0;

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
                ps = con.prepareStatement("select * from product where pid=?");

                ps.setString(1, prodId);
                rs = ps.executeQuery();
```

```
                    if (rs.next()) {
                            quantity = rs.getInt("pquantity");
                    }

            } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            }

            DBUtil.closeConnection(con);
            DBUtil.closeConnection(ps);

            return quantity;
    }

}
```

TRANSACTION SERVICE IMPLEMENTATION

```
package com.shashi.service.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.shashi.service.TransService;
import com.shashi.utility.DBUtil;

public class TransServiceImpl implements TransService {
	@Override
	public String getUserId(String transId) {
		String userId = "";

		Connection con = DBUtil.provideConnection();
		PreparedStatement ps = null;
		ResultSet rs = null;

		try {

			ps = con.prepareStatement("select username from transactions where
transid=?");

			ps.setString(1, transId);

			rs = ps.executeQuery();

			if (rs.next())
				userId = rs.getString(1);

		} catch (SQLException e) {
			e.printStackTrace();
		}

		return userId;
	}

}
```

USER SERVICE IMPLEMENTATION

```
package com.shashi.service.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.shashi.beans.UserBean;
import com.shashi.constants.IUserConstants;
import com.shashi.service.UserService;
import com.shashi.utility.DBUtil;
import com.shashi.utility.MailMessage;

public class UserServiceImpl implements UserService {

	@Override
	public String registerUser(String userName, Long mobileNo, String emailId, String address,
int pinCode,
			String password) {

		UserBean user = new UserBean(userName, mobileNo, emailId, address, pinCode,
password);

		String status = registerUser(user);

		return status;
	}

	@Override
	public String registerUser(UserBean user) {

		String status = "User Registration Failed!";
```

```
        boolean isRegtd = isRegistered(user.getEmail());

        if (isRegtd) {
                status = "Email Id Already Registered!";
                return status;
        }
        Connection conn = DBUtil.provideConnection();
        PreparedStatement ps = null;
        if (conn != null) {
                System.out.println("Connected Successfully!");
        }

        try {

                ps = conn.prepareStatement("insert into " + IUserConstants.TABLE_USER +
" values(?,?,?,?,?,?)");

                ps.setString(1, user.getEmail());
                ps.setString(2, user.getName());
                ps.setLong(3, user.getMobile());
                ps.setString(4, user.getAddress());
                ps.setInt(5, user.getPinCode());
                ps.setString(6, user.getPassword());

                int k = ps.executeUpdate();

                if (k > 0) {
                        status = "User Registered Successfully!";
                        MailMessage.registrationSuccess(user.getEmail(),
user.getName().split(" ")[0]);
                }
        } catch (SQLException e) {
            status = "Error: " + e.getMessage();
            e.printStackTrace();
        }

        DBUtil.closeConnection(ps);
        DBUtil.closeConnection(ps);

        return status;
    }

    @Override
    public boolean isRegistered(String emailId) {
        boolean flag = false;

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
            ps = con.prepareStatement("select * from user where email=?");

            ps.setString(1, emailId);

            rs = ps.executeQuery();

            if (rs.next())
                flag = true;

        } catch (SQLException e) {
```

```
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

			DBUtil.closeConnection(con);
			DBUtil.closeConnection(ps);
			DBUtil.closeConnection(rs);

			return flag;
		}

		@Override
		public String isValidCredential(String emailId, String password) {
			String status = "Login Denied! Incorrect Username or Password";

			Connection con = DBUtil.provideConnection();

			PreparedStatement ps = null;
			ResultSet rs = null;

			try {

				ps = con.prepareStatement("select * from user where email=? and
password=?");

				ps.setString(1, emailId);
				ps.setString(2, password);

				rs = ps.executeQuery();

				if (rs.next())
					status = "valid";
```

```
        } catch (SQLException e) {
            status = "Error: " + e.getMessage();
            e.printStackTrace();
        }

        DBUtil.closeConnection(con);
        DBUtil.closeConnection(ps);
        DBUtil.closeConnection(rs);
        return status;
    }

    @Override
    public UserBean getUserDetails(String emailId, String password) {

        UserBean user = null;

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
            ps = con.prepareStatement("select * from user where email=? and
password=?");
            ps.setString(1, emailId);
            ps.setString(2, password);
            rs = ps.executeQuery();

            if (rs.next()) {
                user = new UserBean();
                user.setName(rs.getString("name"));
```

```
                    user.setMobile(rs.getLong("mobile"));
                    user.setEmail(rs.getString("email"));
                    user.setAddress(rs.getString("address"));
                    user.setPinCode(rs.getInt("pincode"));
                    user.setPassword(rs.getString("password"));

                    return user;
                }

        } catch (SQLException e) {
                e.printStackTrace();
        }

        DBUtil.closeConnection(con);
        DBUtil.closeConnection(ps);
        DBUtil.closeConnection(rs);

        return user;
    }

    @Override
    public String getFName(String emailId) {
        String fname = "";

        Connection con = DBUtil.provideConnection();

        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
                ps = con.prepareStatement("select name from user where email=?");
                ps.setString(1, emailId);
                rs = ps.executeQuery();

                if (rs.next()) {
                        fname = rs.getString(1);

                        fname = fname.split(" ")[0];

                }

        } catch (SQLException e) {

                e.printStackTrace();
        }

        return fname;
}

@Override
public String getUserAddr(String userId) {
        String userAddr = "";

        Connection con = DBUtil.provideConnection();
        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
                ps = con.prepareStatement("select address from user where email=?");

                ps.setString(1, userId);

                rs = ps.executeQuery();
```

```
				if (rs.next())
					userAddr = rs.getString(1);

			} catch (SQLException e) {

				e.printStackTrace();
			}

			return userAddr;
		}

}
```

CART SERVICE

```
package com.shashi.service;

import java.util.List;

import com.shashi.beans.CartBean;

public interface CartService {

	public String addProductToCart(String userId, String prodId, int prodQty);

	public String updateProductToCart(String userId, String prodId, int prodQty);

	public List<CartBean> getAllCartItems(String userId);

	public int getCartCount(String userId);
	public int getCartItemCount(String userId, String itemId);

	public String removeProductFromCart(String userId, String prodId);

	public boolean removeAProduct(String userId, String prodId);

}
```

DEMAND SERVICE

```
package com.shashi.service;

import java.util.List;

import com.shashi.beans.DemandBean;

public interface DemandService {

	public boolean addProduct(String userId, String prodId, int demandQty);

	public boolean addProduct(DemandBean userDemandBean);

	public boolean removeProduct(String userId, String prodId);

	public List<DemandBean> haveDemanded(String prodId);

}
```

ORDER SERVICE

```
package com.shashi.service;

import java.util.List;

import com.shashi.beans.OrderBean;
import com.shashi.beans.OrderDetails;
import com.shashi.beans.TransactionBean;

public interface OrderService {

	public String paymentSuccess(String userName, double paidAmount);

	public boolean addOrder(OrderBean order);

	public boolean addTransaction(TransactionBean transaction);

	public int countSoldItem(String prodId);

	public List<OrderBean> getAllOrders();

	public List<OrderBean> getOrdersByUserId(String emailId);

	public List<OrderDetails> getAllOrderDetails(String userEmailId);

	public String shipNow(String orderId, String prodId);
}
```

PRODUCT SERVICE

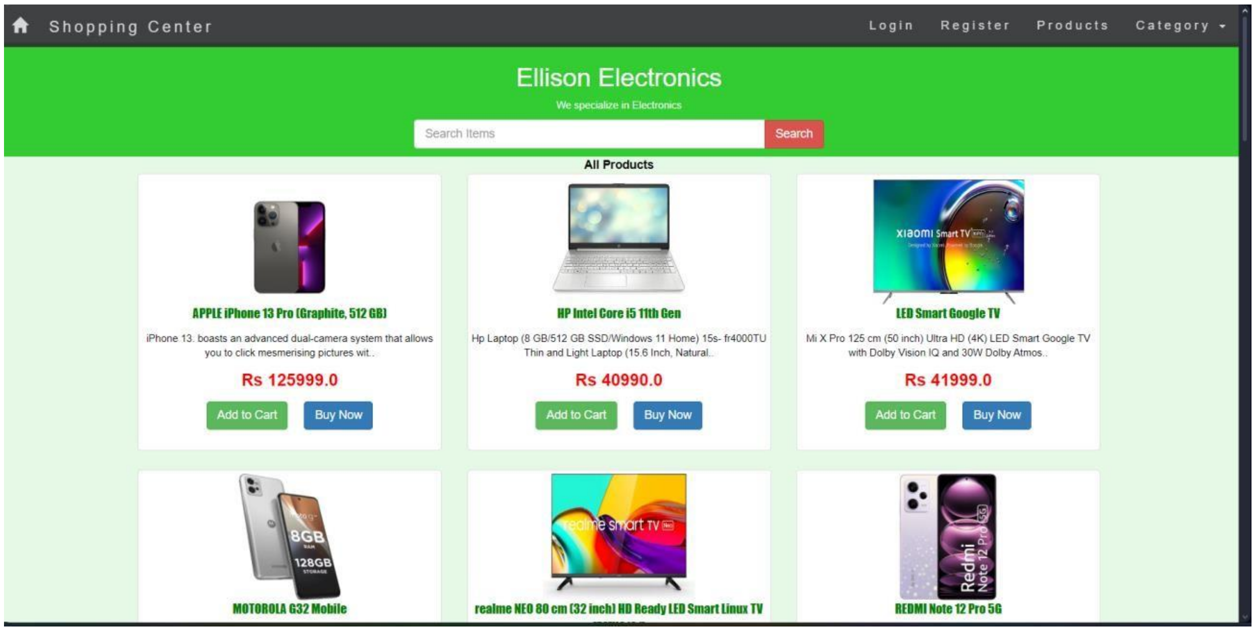
```
package com.shashi.service;

import java.io.InputStream;
import java.util.List;

import com.shashi.beans.ProductBean;

public interface ProductService {

	public String addProduct(String prodName, String prodType, String prodInfo, double
prodPrice, int prodQuantity,
			InputStream prodImage);

	public String addProduct(ProductBean product);

	public String removeProduct(String prodId);

	public String updateProduct(ProductBean prevProduct, ProductBean updatedProduct);

	public String updateProductPrice(String prodId, double updatedPrice);

	public List<ProductBean> getAllProducts();

	public List<ProductBean> getAllProductsByType(String type);

	public List<ProductBean> searchAllProducts(String search);

	public byte[] getImage(String prodId);

	public ProductBean getProductDetails(String prodId);

	public String updateProductWithoutImage(String prevProductId, ProductBean
updatedProduct);

	public double getProductPrice(String prodId);

	public boolean sellNProduct(String prodId, int n);

	public int getProductQuantity(String prodId);
}
```

USER SERVICE

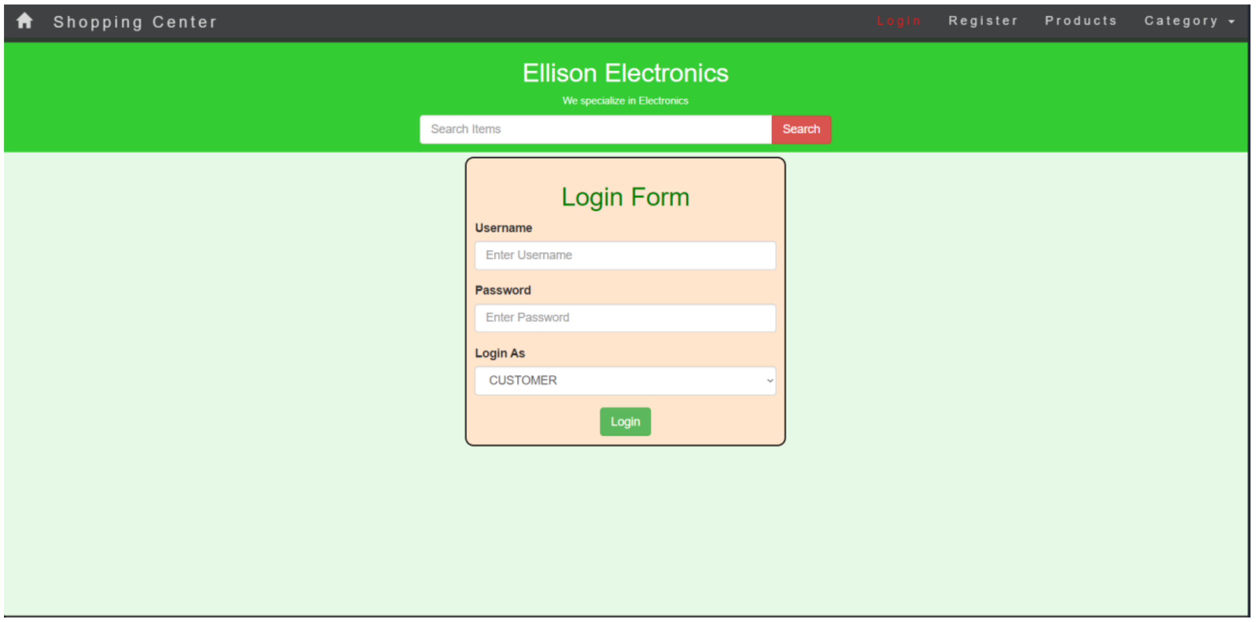
```
package com.shashi.service;

import com.shashi.beans.UserBean;

public interface UserService {

	/*
	 * private String userName; private Long mobileNo; private String emailId;
	 * private String address; private int pinCode; private String password;
	 */

	public String registerUser(String userName, Long mobileNo, String emailId, String address,
int pinCode,
			String password);

	public String registerUser(UserBean user);

	public boolean isRegistered(String emailId);

	public String isValidCredential(String emailId, String password);

	public UserBean getUserDetails(String emailId, String password);

	public String getFName(String emailId);

	public String getUserAddr(String userId);

}
```

V. RESULT AND DISCUSSION

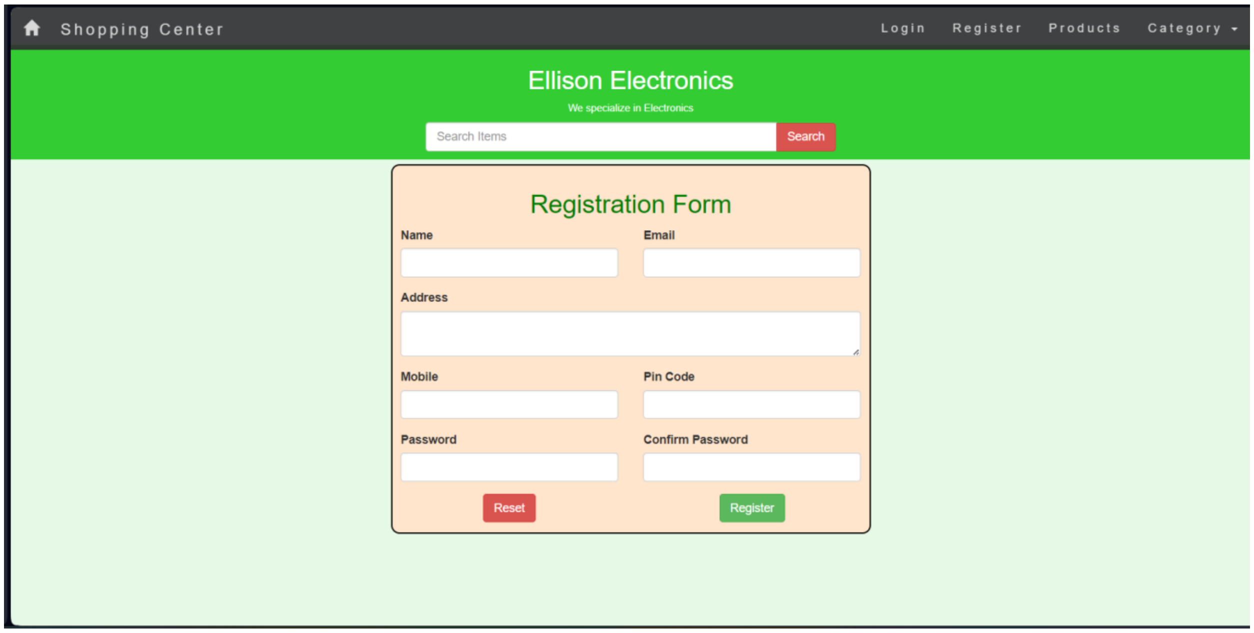
HOME PAGE


Shopping Center
Login
Register
Products
Category
Ellison Electronics
We specialize in Electronics
Search Items
Search
All Products
APPLE iPhone 13 Pro (Graphite, 512 GB)
iPhone 13: boasts an advanced dual-camera system that allows you to click mesmerising pictures wit..
Rs 125999.0
Add to Cart
Buy Now
HP Intel Core i5 11th Gen
Hp Laptop (8 GB/512 GB SSD/Windows 11 Home) 15s- fr4000TU Thin and Light Laptop (15.6 Inch, Natural..
Rs 40990.0
Add to Cart
Buy Now
LED Smart Google TV
Mi X Pro 125 cm (50 inch) Ultra HD (4K) LED Smart Google TV with Dolby Vision IQ and 30W Dolby Atmos..
Rs 41999.0
Add to Cart
Buy Now
MOTOROLA G32 Mobile
realme NEO 80 cm (32 inch) HD Ready LED Smart Linux TV
REDMI Note 12 Pro 5G

LOGIN PAGE



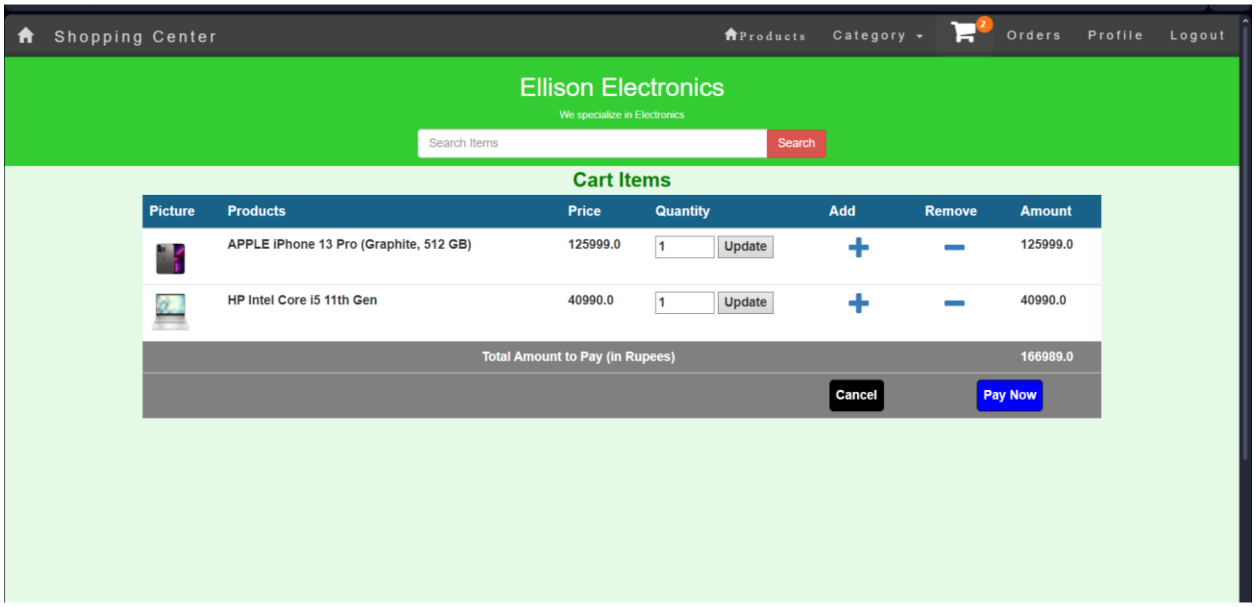
REGISTRATION PAGE



Registration Form

Name	Email
Address	
Mobile	Pin Code
Password	Confirm Password

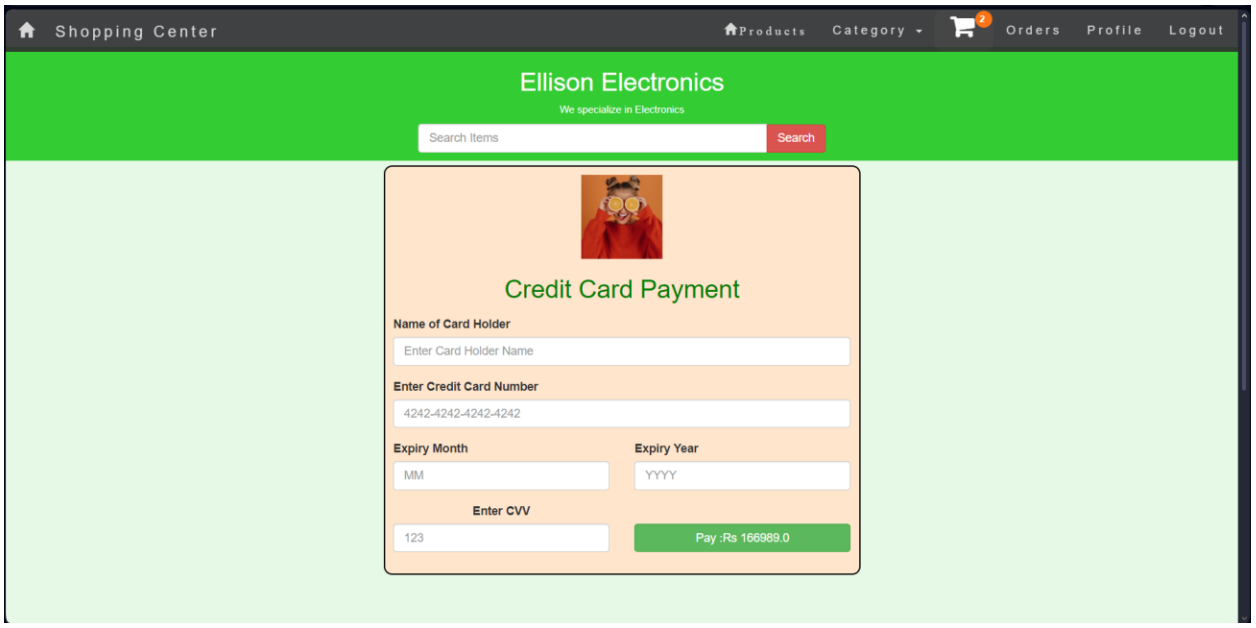
CART PAGE



Cart Items

Picture	Products	Price	Quantity	Add	Remove	Amount
	APPLE iPhone 13 Pro (Graphite, 512 GB)	125999.0	1 Update	+	−	125999.0
	HP Intel Core i5 11th Gen	40990.0	1 Update	+	−	40990.0
	Total Amount to Pay (in Rupees)					166989.0

PAYMENT PAGE



Credit Card Payment

Name of Card Holder: Enter Card Holder Name

Enter Credit Card Number: 4242-4242-4242-4242

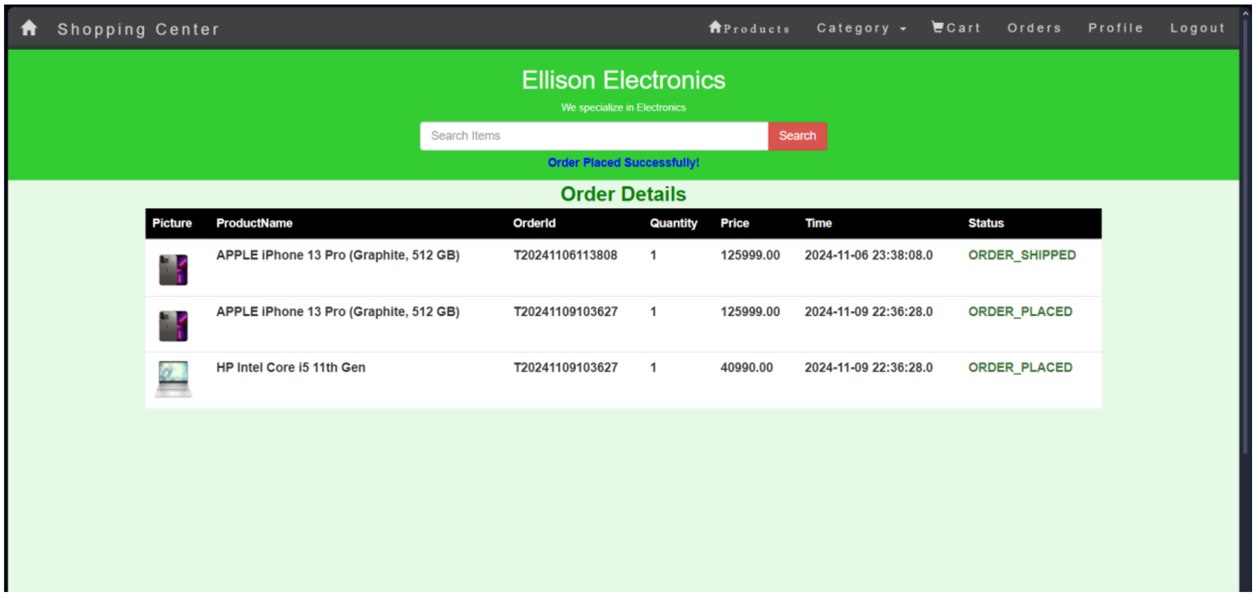
Expiry Month: MM

Expiry Year: YYYY

Enter CVV: 123

Pay :Rs 166989.0

ORDER PAGE AFTER PAYMENT

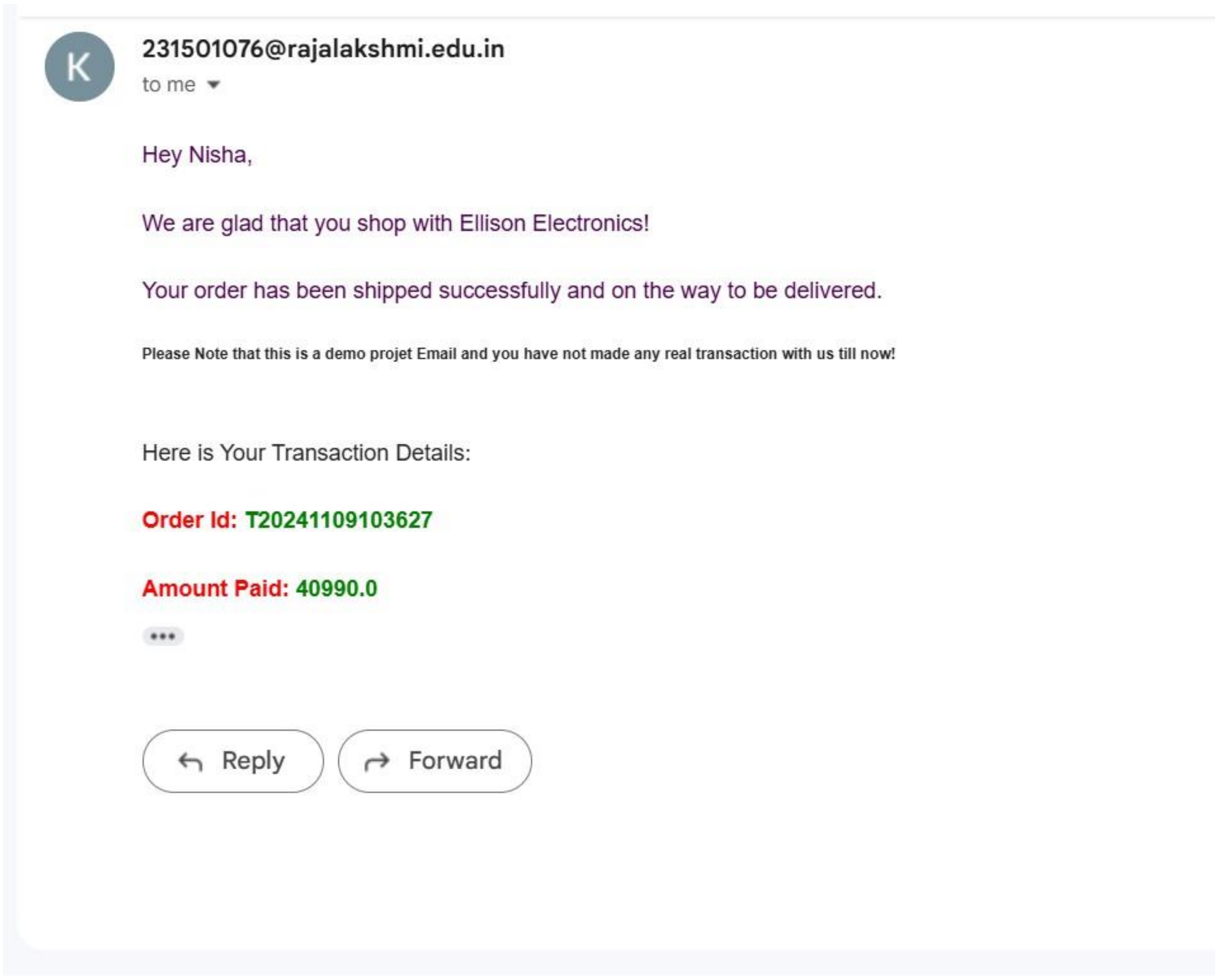


Order Placed Successfully!

Order Details

Picture	ProductName	OrderId	Quantity	Price	Time	Status
	APPLE iPhone 13 Pro (Graphite, 512 GB)	T20241106113808	1	125999.00	2024-11-06 23:38:08.0	ORDER_SHIPPED
	APPLE iPhone 13 Pro (Graphite, 512 GB)	T20241109103627	1	125999.00	2024-11-09 22:36:28.0	ORDER_PLACED
	HP Intel Core i5 11th Gen	T20241109103627	1	40990.00	2024-11-09 22:36:28.0	ORDER_PLACED

ORDER CONFIRMATION PAGE IN EMAIL



231501076@rajalakshmi.edu.in

to me

Hey Nisha,

We are glad that you shop with Ellison Electronics!

Your order has been shipped successfully and on the way to be delivered.

Please Note that this is a demo projet Email and you have not made any real transaction with us till now!

Here is Your Transaction Details:

Order Id: T20241109103627

Amount Paid: 40990.0

RESULTS

1. **User Features:**
    - ○ Registration & Login: Successfully implemented with email confirmation upon registration.
    - ○ Product Browsing: Users could easily view and filter products, enhancing navigation.
    - ○ Shopping Cart: Worked smoothly, with real-time updates when items were added or quantities adjusted.
    - ○ Checkout & Payment: Simulated credit card payments functioned correctly for demo purposes.
    - ○ Order Management: Users could track orders and receive status updates via email.
2. **Admin Functionality:**
    - ○ Product Management: Admins could effectively add, update, or remove products and manage inventory.
    - ○ Order Management: Admins could update the status of orders, marking them as shipped or delivered.
3. **Email Notifications:**
    - ○ Successfully sent automated emails for registration, order placement, restocking updates, and shipping confirmations.
4. **Performance & Security:**
    - ○ The website was responsive and performed well under standard testing, though it requires real payment gateway integration and enhanced security features for a live environment.

Overall, the project achieved its objectives, delivering a functional and user-friendly e-commerce experience with efficient admin tools and robust email communication.

DISCUSSION

1. **User Experience:**
   - **Strengths:** The user interface was intuitive and easy to navigate. Features like product filtering and a responsive shopping cart made the shopping process seamless.
   - **Areas for Improvement:** The checkout process, though functional for demonstration, could be improved with real payment gateway integration for production use. Additionally, incorporating user reviews or ratings would enhance the platform's interactivity.
2. **Email Integration:**
   - **Strengths:** The email notification system was robust and timely, enhancing communication between the store and the customers.
   - **Areas for Improvement:** Using a more secure and efficient email service could be explored in the future, especially to handle larger volumes of notifications.
3. **Admin Efficiency:**
   - **Strengths:** The admin panel allowed for effective management of products and orders, streamlining the e-commerce operations.
   - **Areas for Improvement:** Future versions could include features like sales analytics and automated inventory management alerts.
4. **Security Concerns:**
   - **Discussion:** Since the payment page is currently a demo, adding secure, real-world payment processing is critical. Also, implementing more security measures like data encryption and two-factor authentication (2FA) for user accounts would be beneficial.
5. **Performance:**
   - **Observation:** The website’s performance was satisfactory in handling multiple users and maintaining a consistent experience. However, stress testing could be conducted to ensure the platform’s scalability under high traffic.

VI. CONCLUSION

The development of this E-commerce website for selling electronic products online has successfully demonstrated the essential functionalities needed for a modern and efficient online shopping experience. The project provides a robust foundation by implementing features that cater to both user and administrative needs, ensuring seamless operation and management.

From a **user perspective**, the website offers a smooth and engaging experience. The **registration and login system** works flawlessly, making it easy for new users to join the platform while confirming their membership through automated email notifications.

The **order management system** is another significant accomplishment, offering users transparency and control over their purchases. Users can view detailed order summaries and track the shipping status, which is critical for building customer trust.

From an **administrative standpoint**, the platform provides an intuitive interface for managing products and orders. The **admin dashboard** simplifies tasks like adding new products, updating inventory, and removing items

The **email integration** feature significantly enhances the overall user experience. By automating communication, the system reduces the need for manual updates and ensures that users are kept informed at every stage of their interaction with the platform. This feature also plays a key role in improving customer engagement and retention.

Overall, the project has successfully met its objectives, delivering a functional and well-structured e-commerce platform. It effectively balances user needs with administrative controls, providing a positive and efficient shopping experience. The foundation built by this project is strong and scalable, making it well-suited for future development and real-world implementation. By addressing the highlighted areas for improvement, this e-commerce website has the potential to evolve into a fully operational and secure online marketplace, catering to a wide range of consumer and business requirements

VII REFERENCES

Web Development Resources:

W3Schools: For learning and implementing HTML, CSS, and JavaScript fundamentals used in the front-end development. Available at: **https://www.w3schools.com**

Java and Backend Development:

Oracle Java Documentation: For understanding Java programming and the Java Standard Edition (JDK 8+). Available at: **https://docs.oracle.com/javase**

Database Management:

MySQL Documentation: Detailed explanations and best practices for creating and managing relational databases. Available at: https://dev.mysql.com/doc

Project Management and Development Tools:

GitHub: For version control and project collaboration. Documentation available at: https://docs.github.com

RESEARCH PAPER:

Database Management Systems for E-Commerce Platforms: A Survey.

A Survey on E-commerce Systems: Design, Issues, and Challenges.

https://scholar.google.com

GITHUB LINK:

https://github.com/Kavyashri86/online-electronics-shopping-.git

https://github.com/Nisha820/online-electronics-shopping-.git