Courier Management System

Coding Task 10

Database Interaction

Connect your application to the SQL database for the Courier Management System

1. Write code to establish a connection to your SQL database.

Create a class DBConnection in a package connectionutil with a static variable connection of Type Connection and a static method getConnection() which returns connection. Connection properties supplied in the connection string should be read from a property file.

2. Create a Service class CourierServiceDb in dao with a static variable named connection of type Connection which can be assigned in the constructor by invoking the method in DBConnection Class.

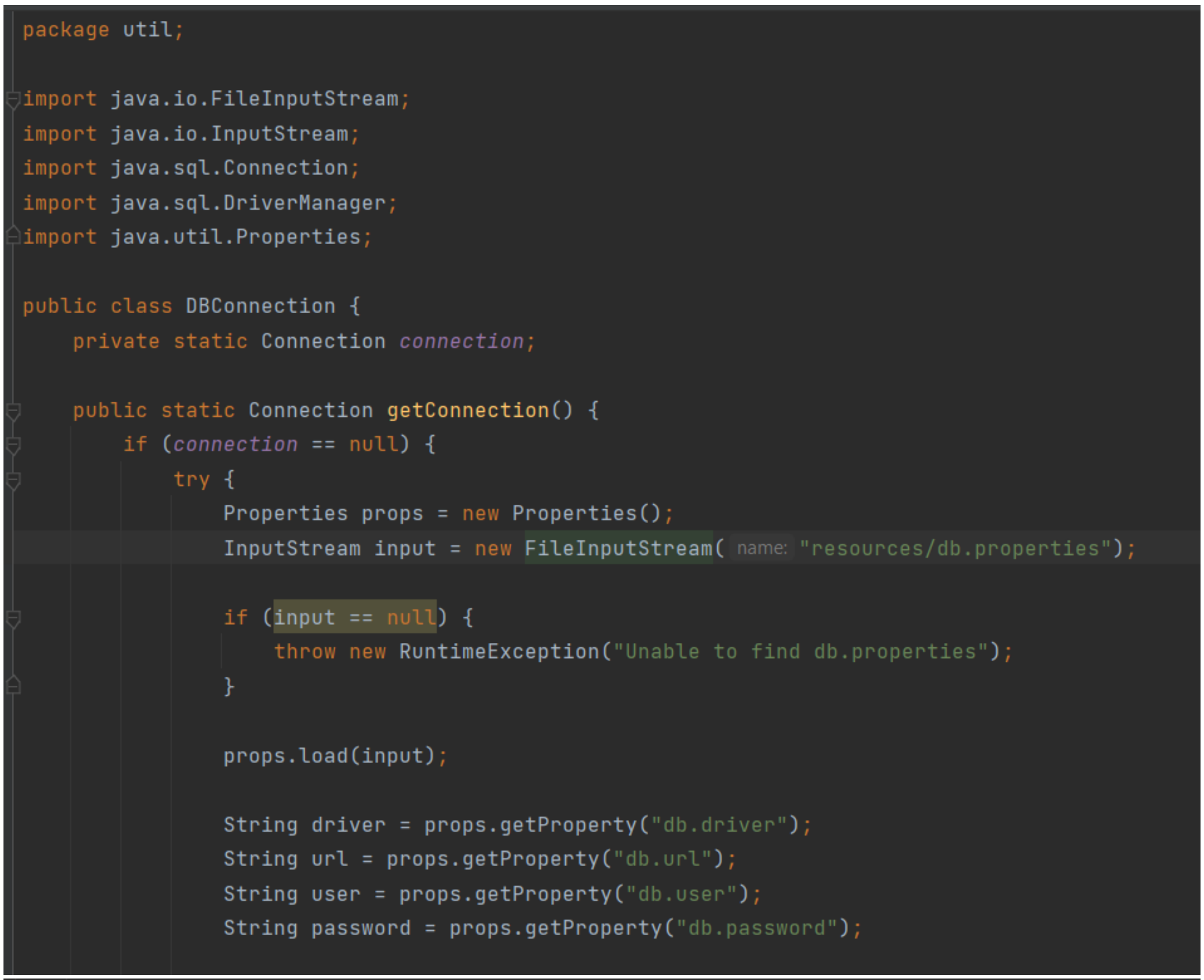
3. Include methods to insert, update, and retrieve data from the database (e.g., inserting a new order, updating courier status).

4. Implement a feature to retrieve and display the delivery history of a specific parcel by querying the database. 1. Generate and display reports using data retrieved from the database (e.g., shipment status report, revenue report).

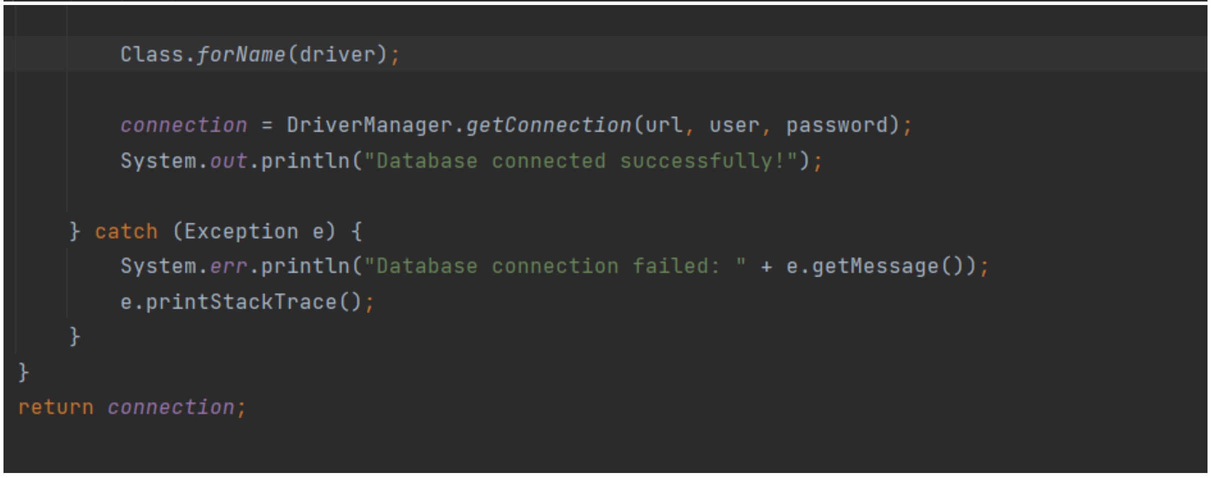
In this task, the primary aim is to establish a secure and reusable connection with the database to perform various operations essential for a Courier Management System. By using a ConnectionUtil class that reads credentials from a .properties file, the connection setup becomes modular and easily maintainable. This setup is utilized across different DAO classes to interact with the database for functionalities like placing courier orders, assigning delivery personnel, processing payments, tracking shipments, and generating service-based reports. This approach ensures consistent, efficient, and scalable interaction with the backend database throughout the application.

Task 10.1: Establishing SQL Database Connection via Utility Class

To ensure a clean and reusable database connection across the application, a dedicated utility class DBConnection was created within the connectionutil package. This class contains a static Connection variable and a getConnection() method that reads database credentials such as URL, username, password, and driver from an external db.properties file. The connection is established using JDBC, promoting modularity and easier configuration management. This abstraction allows changes to database details without modifying the core application logic.



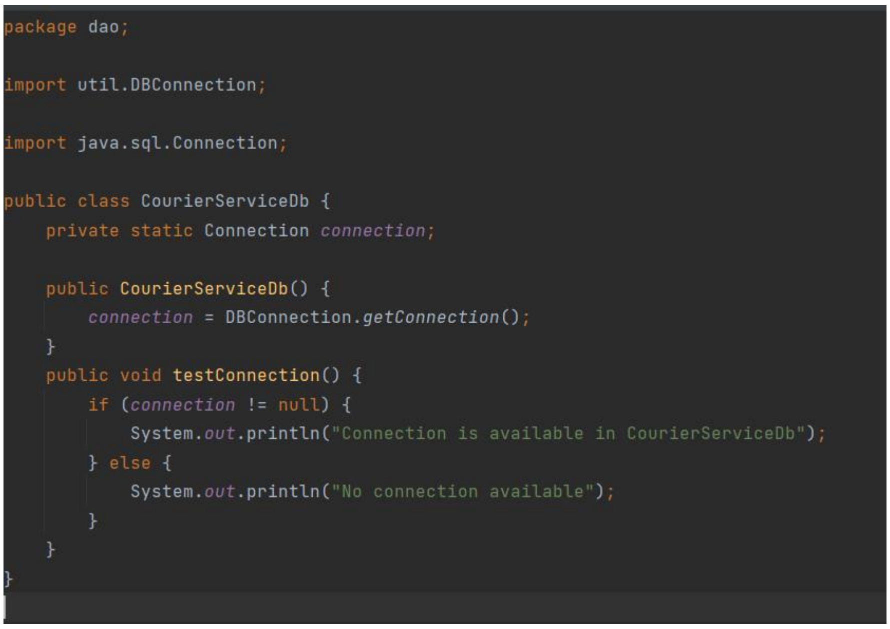
```
package util;

import java.io.FileInputStream;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;

public class DBConnection {
    private static Connection connection;

    public static Connection getConnection() {
        if (connection == null) {
            try {
                Properties props = new Properties();
                InputStream input = new FileInputStream( name: "resources/db.properties");

                if (input == null) {
                    throw new RuntimeException("Unable to find db.properties");
                }

                props.load(input);

                String driver = props.getProperty("db.driver");
                String url = props.getProperty("db.url");
                String user = props.getProperty("db.user");
                String password = props.getProperty("db.password");
```



```

        Class.forName(driver);

        connection = DriverManager.getConnection(url, user, password);
        System.out.println("Database connected successfully!");

    } catch (Exception e) {
        System.err.println("Database connection failed: " + e.getMessage());
        e.printStackTrace();
    }
}
return connection;
```

Task 10.2: Integrating Connection in DAO Layer

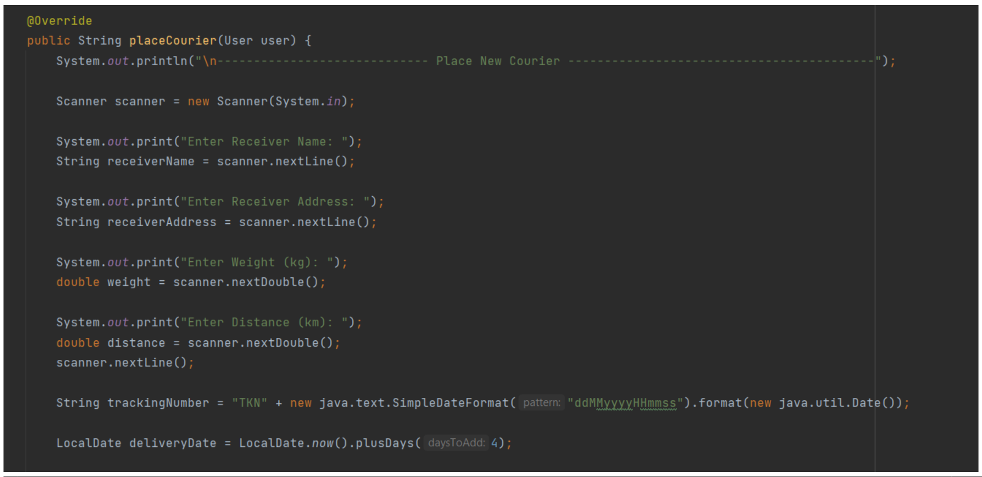
The CourierServiceDb class, located in the dao package, acts as the service layer for courier-related operations. It initializes a static Connection object using the getConnection() method from the DBConnection class, allowing the entire DAO layer to seamlessly communicate with the database. This setup provides a centralized approach for managing connections, ensuring consistency and reducing boilerplate code throughout the data access operations.



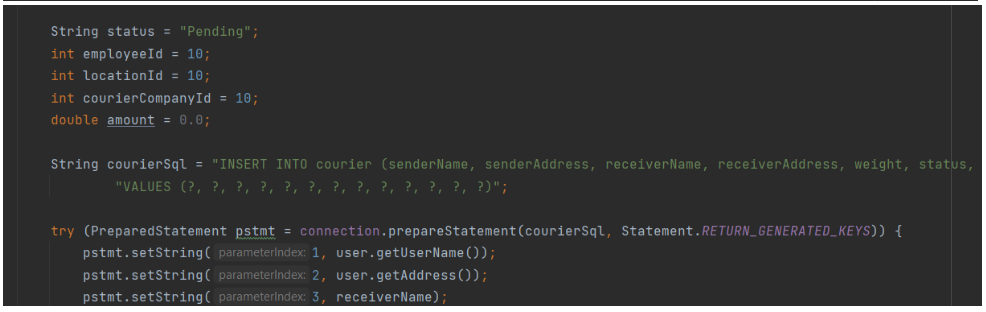
```java
package dao;

import util.DBConnection;

import java.sql.Connection;

public class CourierServiceDb {
    private static Connection connection;

    public CourierServiceDb() {
        connection = DBConnection.getConnection();
    }
    public void testConnection() {
        if (connection != null) {
            System.out.println("Connection is available in CourierServiceDb");
        } else {
            System.out.println("No connection available");
        }
    }
}
```

Task 10.3: CRUD Operations for Courier and User Management

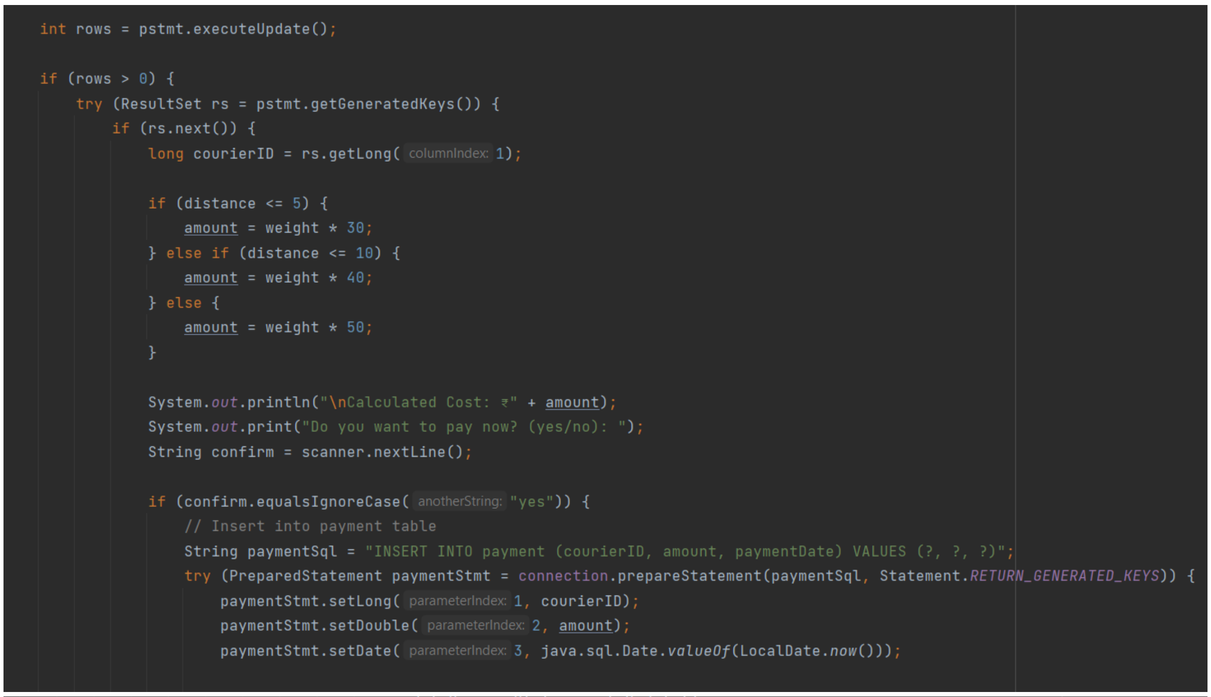
A variety of core functions have been implemented in the CourierUserServiceImpl class to handle operations such as placing a courier order, retrieving tracking status, canceling orders, assigning and listing couriers, processing user payments, and registering new users. SQL INSERT, SELECT, UPDATE, and DELETE statements are used through JDBC's PreparedStatement to interact with the relational database securely and efficiently. Error handling and validations such as employee existence checks and delivery status verifications are also included to ensure robust transactional control.



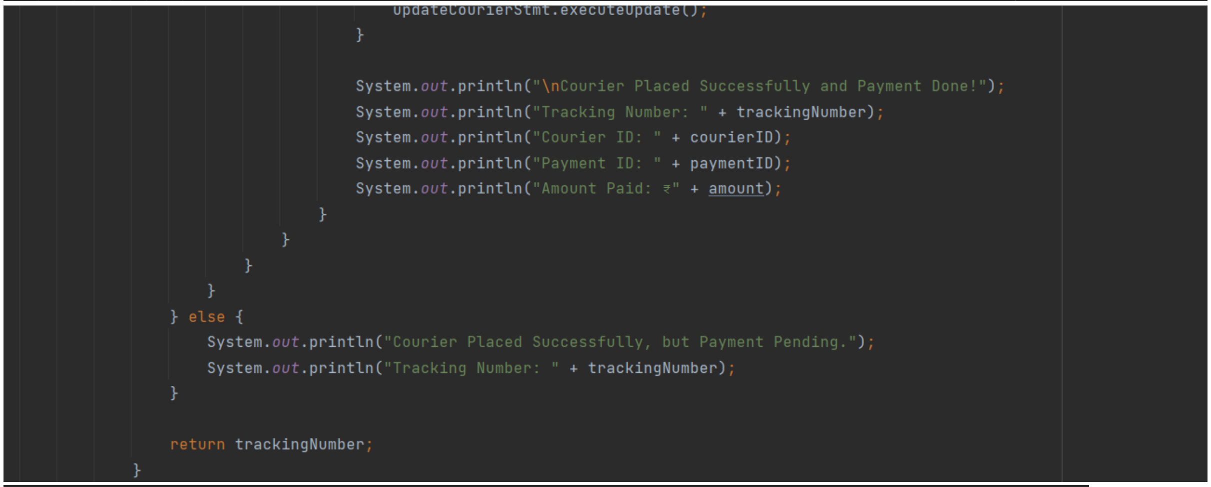
```java
@Override
public String placeCourier(User user) {
    System.out.println("\n--------------------------- Place New Courier ----------------------------------------");

    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter Receiver Name: ");
    String receiverName = scanner.nextLine();

    System.out.print("Enter Receiver Address: ");
    String receiverAddress = scanner.nextLine();

    System.out.print("Enter Weight (kg): ");
    double weight = scanner.nextDouble();

    System.out.print("Enter Distance (km): ");
    double distance = scanner.nextDouble();
    scanner.nextLine();

    String trackingNumber = "TKN" + new java.text.SimpleDateFormat( pattern: "ddMMyyyyHHmmss").format(new java.util.Date());

    LocalDate deliveryDate = LocalDate.now().plusDays( daysToAdd: 4);
```



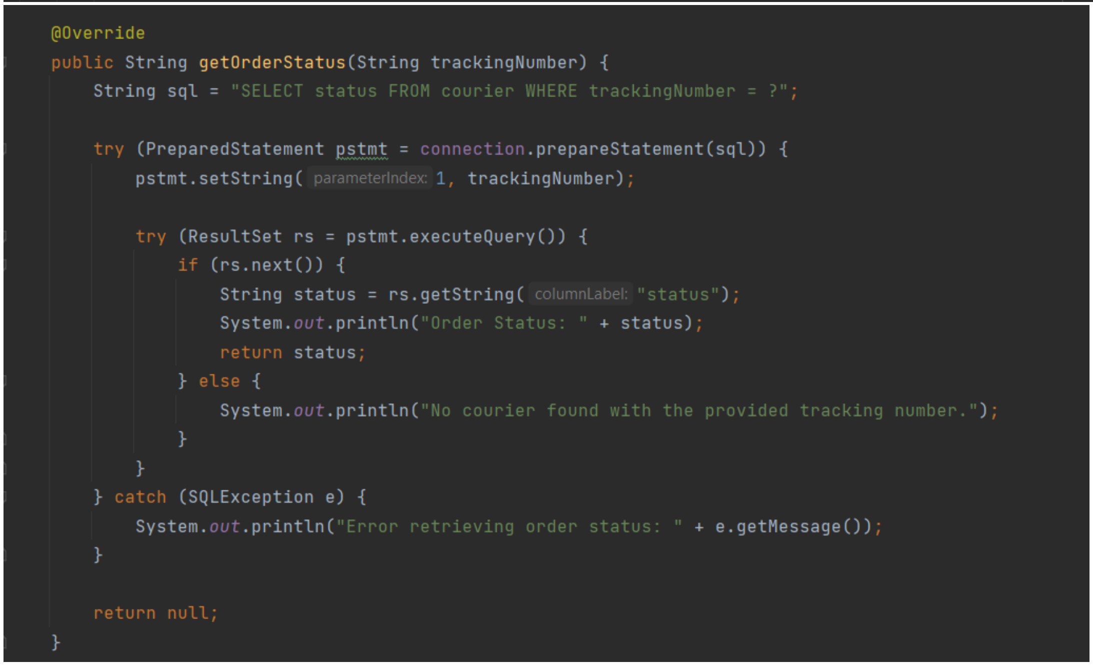
```java
    String status = "Pending";
    int employeeId = 10;
    int locationId = 10;
    int courierCompanyId = 10;
    double amount = 0.0;

    String courierSql = "INSERT INTO courier (senderName, senderAddress, receiverName, receiverAddress, weight, status,
            "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

    try (PreparedStatement pstmt = connection.prepareStatement(courierSql, Statement.RETURN_GENERATED_KEYS)) {
        pstmt.setString( parameterIndex: 1, user.getUserName());
        pstmt.setString( parameterIndex: 2, user.getAddress());
        pstmt.setString( parameterIndex: 3, receiverName);
```



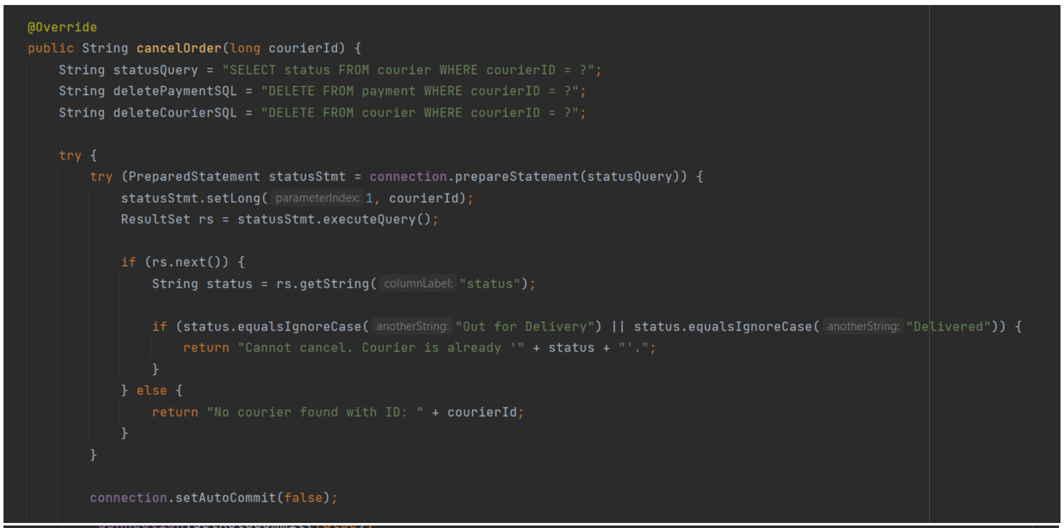
```java
int rows = pstmt.executeUpdate();

if (rows > 0) {
    try (ResultSet rs = pstmt.getGeneratedKeys()) {
        if (rs.next()) {
            long courierID = rs.getLong( columnIndex: 1);

            if (distance <= 5) {
                amount = weight * 30;
            } else if (distance <= 10) {
                amount = weight * 40;
            } else {
                amount = weight * 50;
            }

            System.out.println("\nCalculated Cost: ₹" + amount);
            System.out.print("Do you want to pay now? (yes/no): ");
            String confirm = scanner.nextLine();

            if (confirm.equalsIgnoreCase( anotherString: "yes")) {
                // Insert into payment table
                String paymentSql = "INSERT INTO payment (courierID, amount, paymentDate) VALUES (?, ?, ?)";
                try (PreparedStatement paymentStmt = connection.prepareStatement(paymentSql, Statement.RETURN_GENERATED_KEYS)) {
                    paymentStmt.setLong( parameterIndex: 1, courierID);
                    paymentStmt.setDouble( parameterIndex: 2, amount);
                    paymentStmt.setDate( parameterIndex: 3, java.sql.Date.valueOf(LocalDate.now()));
```



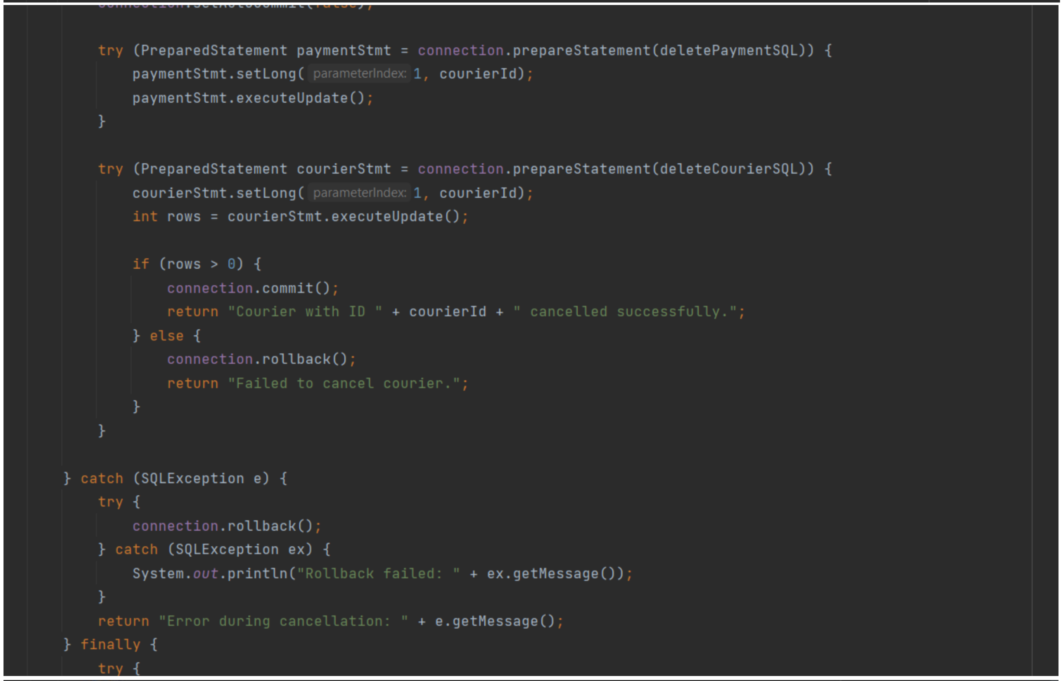
```java
                                updateCourierStmt.executeUpdate();
                            }

                            System.out.println("\nCourier Placed Successfully and Payment Done!");
                            System.out.println("Tracking Number: " + trackingNumber);
                            System.out.println("Courier ID: " + courierID);
                            System.out.println("Payment ID: " + paymentID);
                            System.out.println("Amount Paid: ₹" + amount);
                        }
                    }
                }
            }
        } else {
            System.out.println("Courier Placed Successfully, but Payment Pending.");
            System.out.println("Tracking Number: " + trackingNumber);
        }

        return trackingNumber;
    }
```



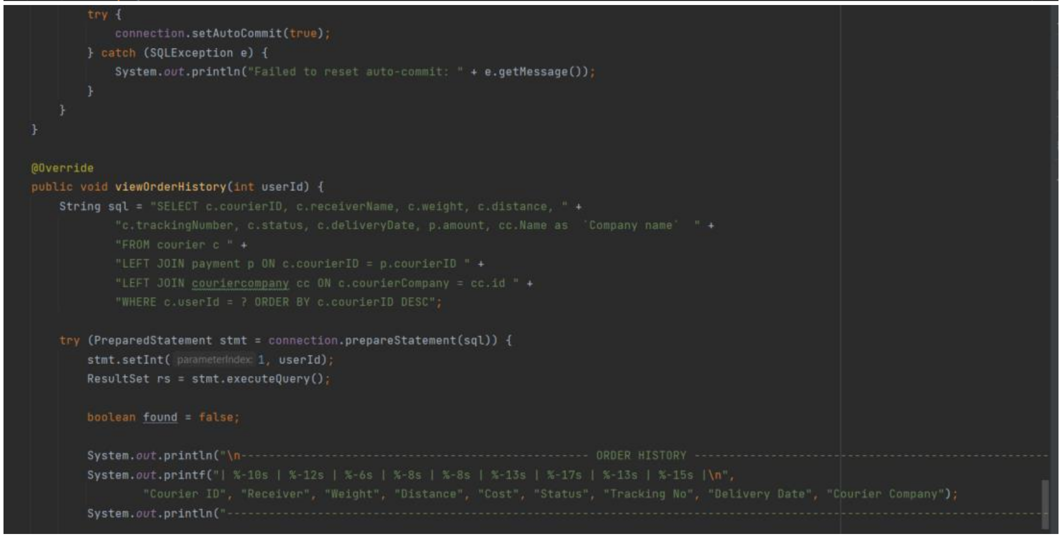
```java
@Override
public String getOrderStatus(String trackingNumber) {
    String sql = "SELECT status FROM courier WHERE trackingNumber = ?";

    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setString( parameterIndex: 1, trackingNumber);

        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                String status = rs.getString( columnLabel: "status");
                System.out.println("Order Status: " + status);
                return status;
            } else {
                System.out.println("No courier found with the provided tracking number.");
            }
        }
    } catch (SQLException e) {
        System.out.println("Error retrieving order status: " + e.getMessage());
    }

    return null;
}
```



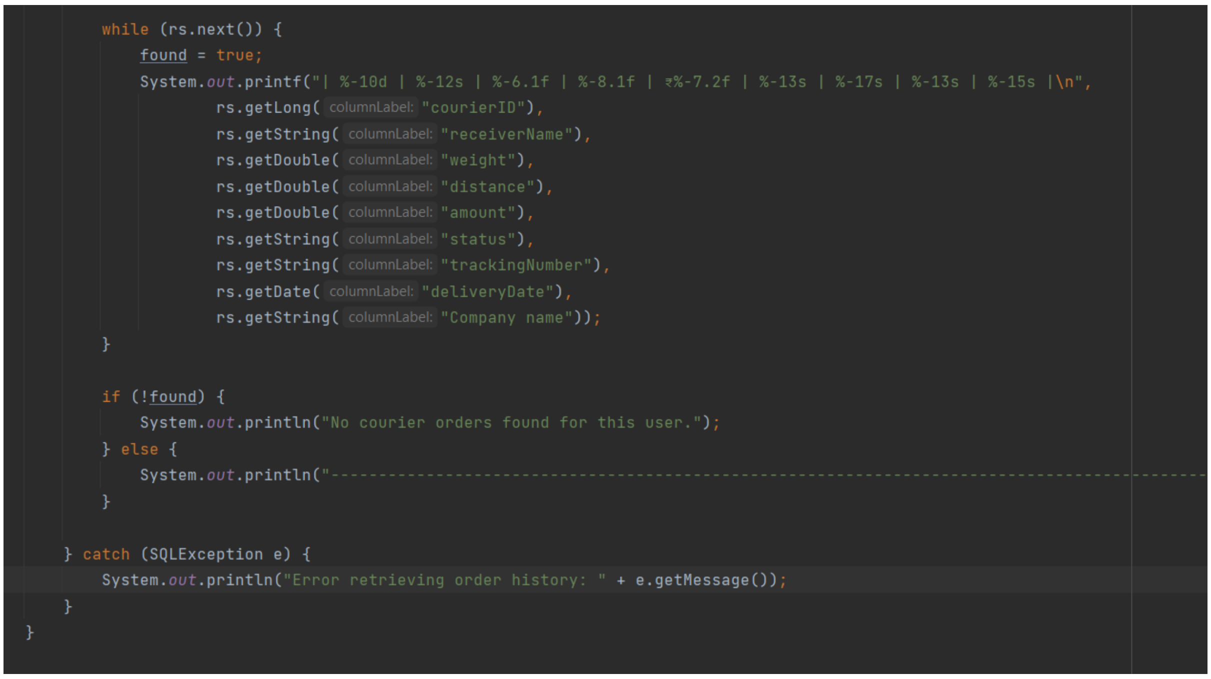
```java
@Override
public String cancelOrder(long courierId) {
    String statusQuery = "SELECT status FROM courier WHERE courierID = ?";
    String deletePaymentSQL = "DELETE FROM payment WHERE courierID = ?";
    String deleteCourierSQL = "DELETE FROM courier WHERE courierID = ?";

    try {
        try (PreparedStatement statusStmt = connection.prepareStatement(statusQuery)) {
            statusStmt.setLong( parameterIndex: 1, courierId);
            ResultSet rs = statusStmt.executeQuery();

            if (rs.next()) {
                String status = rs.getString( columnLabel: "status");

                if (status.equalsIgnoreCase( anotherString: "Out for Delivery") || status.equalsIgnoreCase( anotherString: "Delivered")) {
                    return "Cannot cancel. Courier is already '" + status + "'.";
                }
            } else {
                return "No courier found with ID: " + courierId;
            }
        }

        connection.setAutoCommit(false);
```



```java
        try (PreparedStatement paymentStmt = connection.prepareStatement(deletePaymentSQL)) {
            paymentStmt.setLong( parameterIndex: 1, courierId);
            paymentStmt.executeUpdate();
        }

        try (PreparedStatement courierStmt = connection.prepareStatement(deleteCourierSQL)) {
            courierStmt.setLong( parameterIndex: 1, courierId);
            int rows = courierStmt.executeUpdate();

            if (rows > 0) {
                connection.commit();
                return "Courier with ID " + courierId + " cancelled successfully.";
            } else {
                connection.rollback();
                return "Failed to cancel courier.";
            }
        }

    } catch (SQLException e) {
        try {
            connection.rollback();
        } catch (SQLException ex) {
            System.out.println("Rollback failed: " + ex.getMessage());
        }
        return "Error during cancellation: " + e.getMessage();
    } finally {
        try {
```

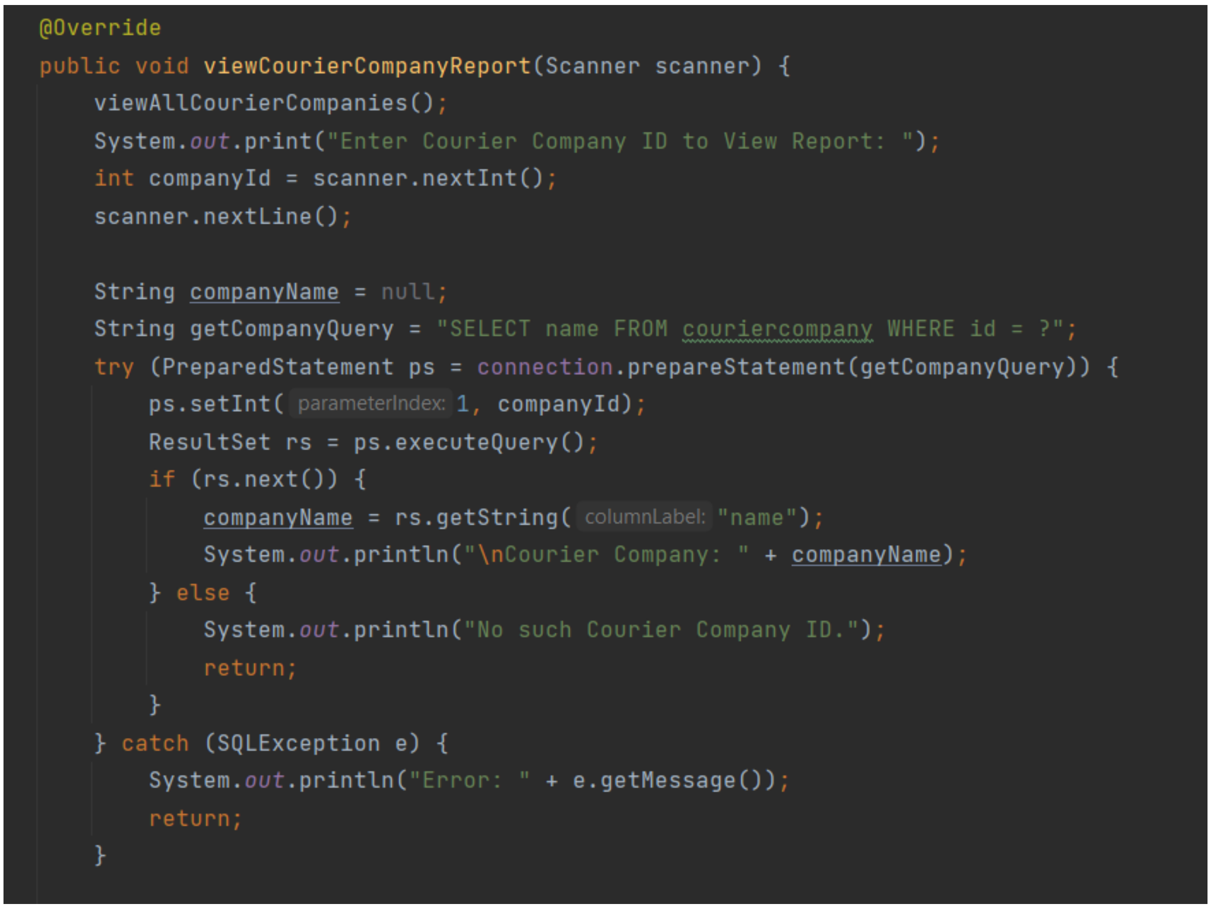


```java
        try {
            connection.setAutoCommit(true);
        } catch (SQLException e) {
            System.out.println("Failed to reset auto-commit: " + e.getMessage());
        }
    }
}

@Override
public void viewOrderHistory(int userId) {
    String sql = "SELECT c.courierID, c.receiverName, c.weight, c.distance, " +
            "c.trackingNumber, c.status, c.deliveryDate, p.amount, cc.Name as  `Company name`  " +
            "FROM courier c " +
            "LEFT JOIN payment p ON c.courierID = p.courierID " +
            "LEFT JOIN couriercompany cc ON c.courierCompany = cc.id " +
            "WHERE c.userId = ? ORDER BY c.courierID DESC";

    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setInt( parameterIndex: 1, userId);
        ResultSet rs = stmt.executeQuery();

        boolean found = false;

        System.out.println("\n---------------------------------------------------- ORDER HISTORY ----------------------------------------------------
        System.out.printf("| %-10s | %-12s | %-6s | %-8s | %-8s | %-13s | %-17s | %-13s | %-15s |\n",
                "Courier ID", "Receiver", "Weight", "Distance", "Cost", "Status", "Tracking No", "Delivery Date", "Courier Company");
        System.out.println("------------------------------------------------------------------------------------------------------------------------
```

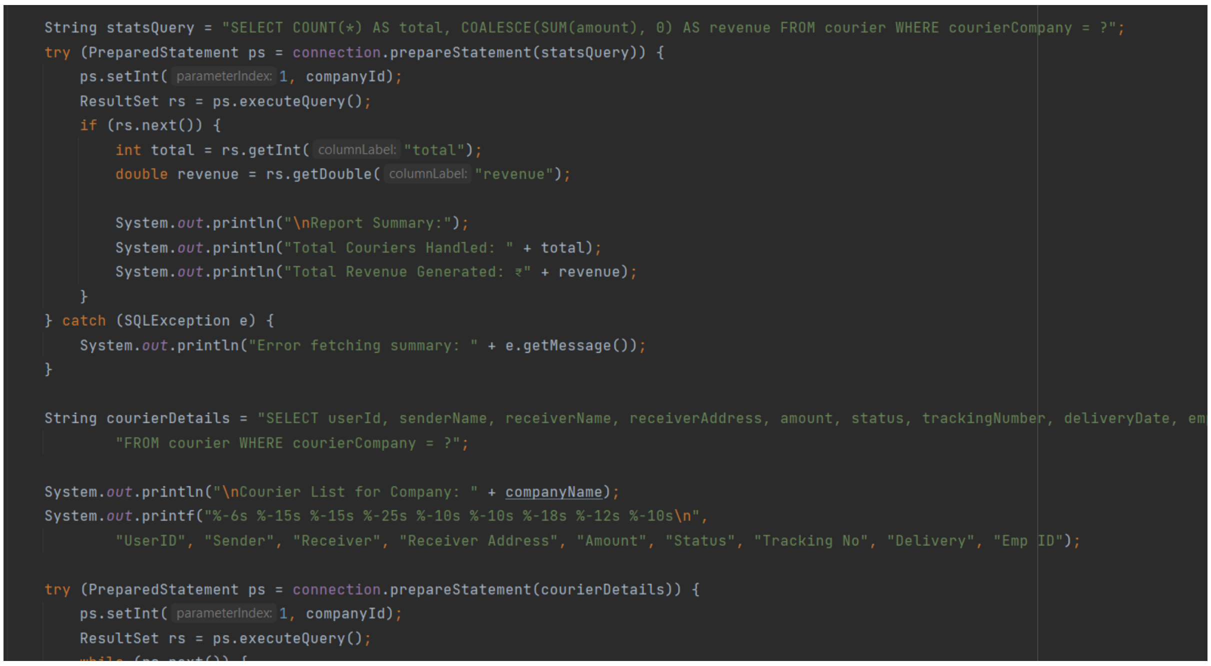


```
        while (rs.next()) {
            found = true;
            System.out.printf("| %-10d | %-12s | %-6.1f | %-8.1f | ₹%-7.2f | %-13s | %-17s | %-13s | %-15s |\n",
                    rs.getLong( columnLabel: "courierID"),
                    rs.getString( columnLabel: "receiverName"),
                    rs.getDouble( columnLabel: "weight"),
                    rs.getDouble( columnLabel: "distance"),
                    rs.getDouble( columnLabel: "amount"),
                    rs.getString( columnLabel: "status"),
                    rs.getString( columnLabel: "trackingNumber"),
                    rs.getDate( columnLabel: "deliveryDate"),
                    rs.getString( columnLabel: "Company name"));
        }

        if (!found) {
            System.out.println("No courier orders found for this user.");
        } else {
            System.out.println("---------------------------------------------------------------------------------------------------------
        }

    } catch (SQLException e) {
        System.out.println("Error retrieving order history: " + e.getMessage());
    }
}
```

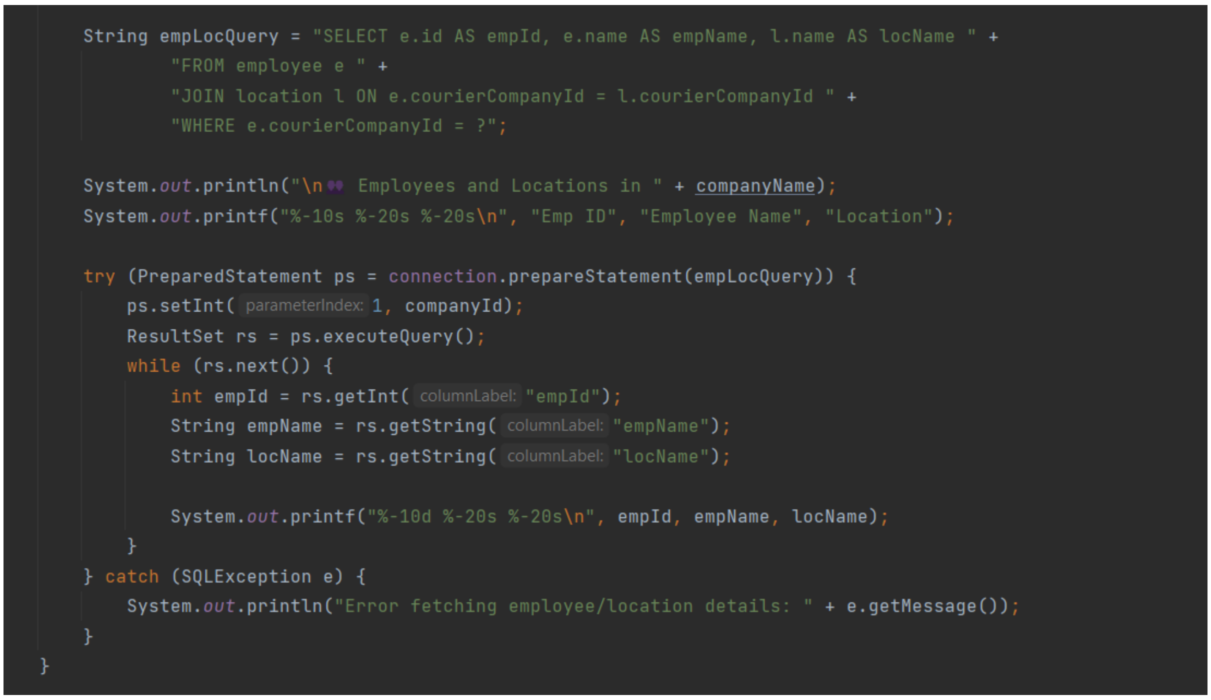
Task 10.4: Reporting and Delivery History Retrieval



```
@Override
public void viewCourierCompanyReport(Scanner scanner) {
    viewAllCourierCompanies();
    System.out.print("Enter Courier Company ID to View Report: ");
    int companyId = scanner.nextInt();
    scanner.nextLine();

    String companyName = null;
    String getCompanyQuery = "SELECT name FROM couriercompany WHERE id = ?";
    try (PreparedStatement ps = connection.prepareStatement(getCompanyQuery)) {
        ps.setInt( parameterIndex: 1, companyId);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            companyName = rs.getString( columnLabel: "name");
            System.out.println("\nCourier Company: " + companyName);
        } else {
            System.out.println("No such Courier Company ID.");
            return;
        }
    } catch (SQLException e) {
        System.out.println("Error: " + e.getMessage());
        return;
    }
```



```java
String statsQuery = "SELECT COUNT(*) AS total, COALESCE(SUM(amount), 0) AS revenue FROM courier WHERE courierCompany = ?";
try (PreparedStatement ps = connection.prepareStatement(statsQuery)) {
    ps.setInt( parameterIndex: 1, companyId);
    ResultSet rs = ps.executeQuery();
    if (rs.next()) {
        int total = rs.getInt( columnLabel: "total");
        double revenue = rs.getDouble( columnLabel: "revenue");

        System.out.println("\nReport Summary:");
        System.out.println("Total Couriers Handled: " + total);
        System.out.println("Total Revenue Generated: ₹" + revenue);
    }
} catch (SQLException e) {
    System.out.println("Error fetching summary: " + e.getMessage());
}

String courierDetails = "SELECT userId, senderName, receiverName, receiverAddress, amount, status, trackingNumber, deliveryDate, em
        "FROM courier WHERE courierCompany = ?";

System.out.println("\nCourier List for Company: " + companyName);
System.out.printf("%-6s %-15s %-15s %-25s %-10s %-10s %-18s %-12s %-10s\n",
        "UserID", "Sender", "Receiver", "Receiver Address", "Amount", "Status", "Tracking No", "Delivery", "Emp ID");

try (PreparedStatement ps = connection.prepareStatement(courierDetails)) {
    ps.setInt( parameterIndex: 1, companyId);
    ResultSet rs = ps.executeQuery();
```



```java
    String empLocQuery = "SELECT e.id AS empId, e.name AS empName, l.name AS locName " +
            "FROM employee e " +
            "JOIN location l ON e.courierCompanyId = l.courierCompanyId " +
            "WHERE e.courierCompanyId = ?";

    System.out.println("\n👥 Employees and Locations in " + companyName);
    System.out.printf("%-10s %-20s %-20s\n", "Emp ID", "Employee Name", "Location");

    try (PreparedStatement ps = connection.prepareStatement(empLocQuery)) {
        ps.setInt( parameterIndex: 1, companyId);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            int empId = rs.getInt( columnLabel: "empId");
            String empName = rs.getString( columnLabel: "empName");
            String locName = rs.getString( columnLabel: "locName");

            System.out.printf("%-10d %-20s %-20s\n", empId, empName, locName);
        }
    } catch (SQLException e) {
        System.out.println("Error fetching employee/location details: " + e.getMessage());
    }
}
```

Conclusion :

The Admin and User interfaces in this courier management system are thoughtfully designed to handle core operations efficiently. Admin methods provide comprehensive control over courier tracking, employee and company management, and system-wide reporting, ensuring smooth backend operations. Meanwhile, user-focused methods enable seamless courier booking, order tracking, and account handling, offering a user-friendly experience. Together, these interfaces form a cohesive and functional system that supports end-to-end courier service management.