Pet Grooming and Boarding Management System

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE

DEGREE OF **BACHELOR OF TECHNOLOGY**

IN **INFORMATION TECHNOLOGY**

OF THE ANNA UNIVERSITY

November 2024

PROJECT WORK

Submitted by,

SUPRIYA D V – 722820522051

Under the Guidance of,

Ms. Jayapratha. T, M.E.,

Assistant Professor/IT

BATCH

2023–2027

INFORMATION TECHNOLOGY

Sri Eshwar College of Engineering

(An Autonomous Institution– Affiliated to Anna University)

COIMBATORE – 641 202

Sri Eshwar College of Engineering

(An Autonomous Institution– Affiliated to Anna University)

COIMBATORE – 641 202

BONAFIDE CERTIFICATE

Certified that this Report titled **"Pet Grooming and Boarding Management System" System"** is the Bonafide work of **Supriya DV – 722820522051** who carried out the project work under my supervision.

SIGNATURE

Dr.P.John Augustine, M.E., Ph.D.
Professor & Head,
Dept. of Information Technology,
Sri Eshwar College of Engineering,
Coimbatore – 641 202.

SIGNATURE

Ms.T.Jayapratha M.E.,
Assistant Professor,
Dept. of Information Technology,
Sri Eshwar College of Engineering,
Coimbatore – 641 202

Submitted for the **Autonomous Semester End Mini-Project Viva-Voice** held on

……..

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

The **Pet Grooming and Boarding Management System** is a comprehensive console-based application designed to streamline the management of grooming and boarding services for pets. This system simplifies appointment scheduling, boarding reservations, and pet and owner data management, ensuring a seamless experience for administrators and clients.

Built on Java, the system leverages key programming principles such as Classes, Inheritance, Interfaces, and Collections to offer a modular and scalable solution. A MySQL database securely stores pet and owner details, along with booking records, ensuring reliable data management and accessibility. Collections like ArrayList are used for real-time tracking of appointments and bookings, while file handling enables persistent storage of transaction logs and histories.

Key features of the system include automated scheduling of grooming sessions, dynamic booking of boarding slots with real-time availability checks, and detailed record management of pets and their owners. It incorporates exception handling to ensure robust functionality, managing issues such as duplicate bookings or invalid data entries. The intuitive menu-driven interface provides administrators with options to book services, view existing appointments, and update customer details. Advanced functionalities such as multithreading enable concurrent booking transactions, enhancing the system's efficiency in high-demand scenarios. The system also supports flexible pricing and service management by maintaining a centralized table for grooming and boarding services with customizable rates and durations.

By integrating Java Database Connectivity (JDBC), the application ensures efficient and secure interaction with the backend database. This system is tailored to meet the needs of pet care centers, grooming salons, and boarding facilities, providing a reliable, user-friendly, and scalable solution for managing pet-related services with precision and efficiency.

.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVES

The **Pet Grooming and Boarding Management System** efficiently manages grooming appointments, boarding reservations, and pet-owner data. It provides a secure, scalable, and user-friendly platform for services like grooming and boarding while maintaining comprehensive records. Leveraging Java concepts such as Classes, Interfaces, Collections, and JDBC, the system ensures modular design, real-time data handling, and seamless database integration. Key features include automated appointment scheduling, dynamic boarding slot allocation, and fee calculations. Exception handling prevents issues like invalid data and duplicate appointments, while file handling ensures persistent record storage. Multithreading enables smooth concurrent operations. Adaptable to pet salons, veterinary clinics, and boarding kennels, the system enhances operational efficiency and customer satisfaction.

1.2 SCOPE OF THE PROJECT

The scope of the **Pet Grooming and Boarding Management System** includes developing a console-based application that allows users to schedule grooming appointments, book boarding slots, manage pet and owner details, and retrieve booking histories. The system will manage and store data related to pets, owners, and bookings in a relational database, ensuring accurate service availability and real-time updates. It will provide a user-friendly interface for booking services, viewing schedules, and managing cancellations, with a focus on data integrity, error handling, and efficient performance. The project will be designed to handle multiple concurrent user interactions for smooth operations.

CHAPTER 2

SYSTEM ANALYSIS AND SPECIFICATION

2.1 PROBLEM DESCRIPTION

The **Pet Grooming and Boarding Management System** tackles inefficiencies in scheduling grooming appointments, managing boarding slots, and maintaining pet-owner records. Manual processes often lead to errors like double bookings, misplaced records, and service delays. This project provides an automated solution to streamline operations, ensuring efficient appointment scheduling, boarding management, and secure data storage. With real-time data handling, JDBC integration, and exception management, the system delivers accurate, reliable, and error-free functionality, improving operational efficiency and customer satisfaction.

2.2 FUNCTIONAL REQUIREMENTS

SOFTWARE AND HARDWARE REQUIREMENTS

Software Requirements: Java 21, JDK, MySQL, MySQL Workbench, JDBC, Java Standard Library, MySQL JDBC Driver IDE (e.g., IntelliJ, Eclipse).

Hardware Requirements: A system with a minimum of 8GB RAM, 100GB storage, and a multi-core processor.

2.3 NON-FUNCTIONAL REQUIREMENTS

Performance: Handle multiple concurrent transactions such as train bookings and cancellations with minimal latency, ensuring fast and responsive operations using efficient database queries.

Scalability: Support the addition of new train routes, stations, and a growing number of passengers without a decrease in performance or user experience.

Reliability: Ensure high availability with proper backup mechanisms in place, enabling quick recovery from system failures.

Security: Implement role-based access control for administrators and passengers to ensure data privacy and secure management of bookings.

Usability: Provide a user-friendly console interface for passengers to book tickets, check availability, and cancel bookings, while also providing a simple interface for administrators to manage train schedules.

Compatibility: Ensure the system operates seamlessly on various operating systems, including Windows, macOS, and Linux.

Data Integrity: Ensure accurate and consistent records using database constraints and triggers.

Backup and Recovery: Implement regular backups of the database to ensure that critical data is not lost, with quick recovery options in place in case of a failure.

Auditability: Maintain logs of critical actions such as ticket bookings, cancellations, and payment transactions for traceability and accountability, ensuring that all operations are traceable in case of issues or disputes.

CHAPTER 3
SYSTEM DESIGN

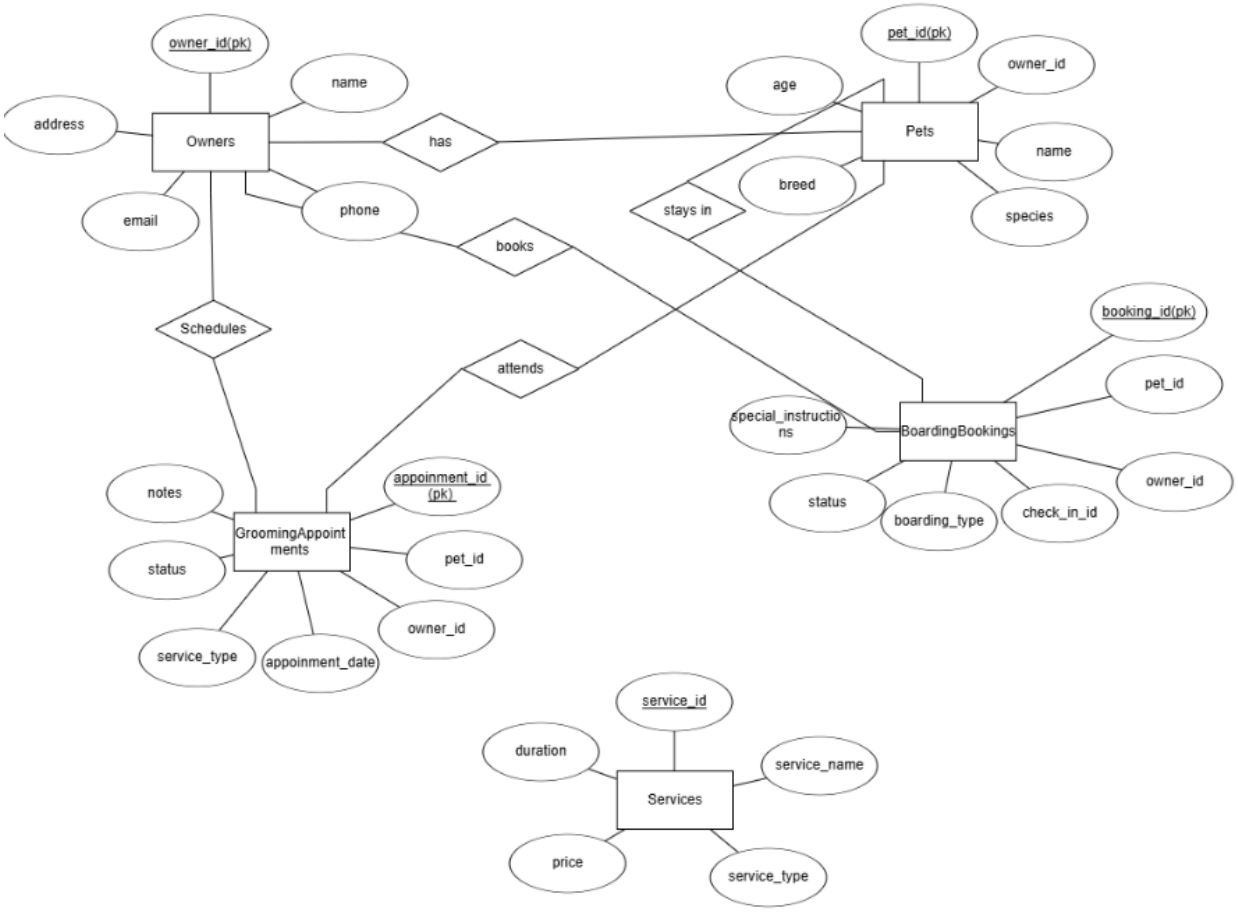

Figure 3.1 ER Diagram for Pet Grooming and Boarding Management System

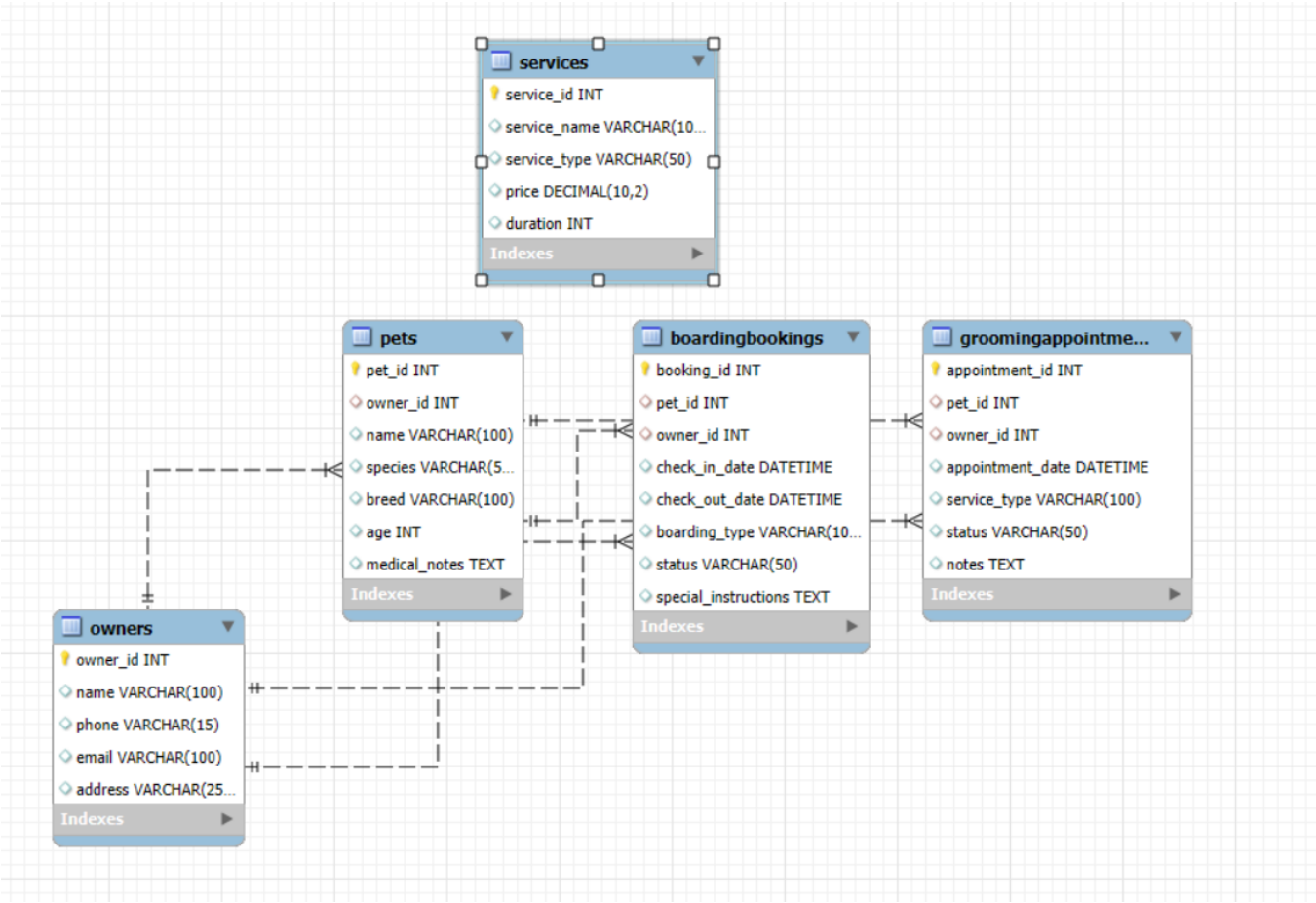

Figure 3.2 SCHEMA Diagram for Pet Grooming and Boarding Management System

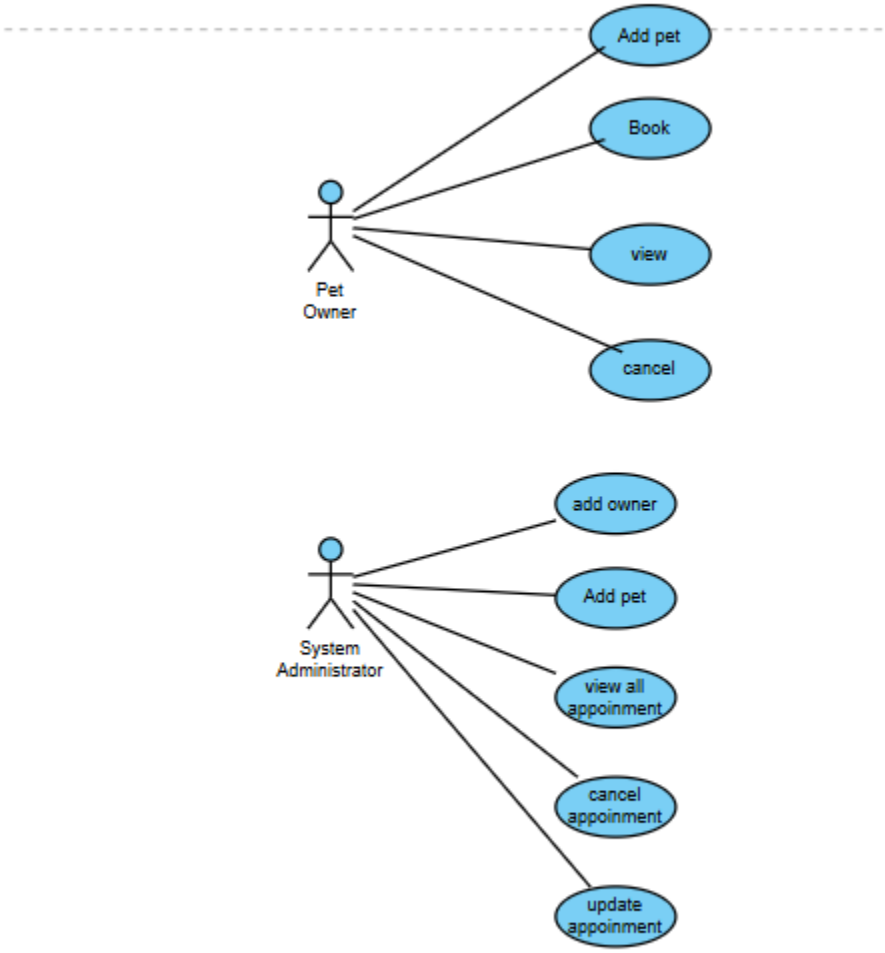

Figure 3.3 USE CASE Diagram for Pet Grooming and Boarding Management System

CHAPTER 4

PROPOSED SOLUTION

4.1 CORE FUNCTIONALITIES

1. **Grooming Appointment Scheduling**: Efficiently schedule grooming services based on pet type, availability, and service preferences.
2. **Boarding Slot Booking** : Manage boarding slots with real-time availability updates and support for different boarding types like daycare or overnight stays.
3. **Service Fee Calculation** : Automatically calculate fees for grooming and boarding services based on service type, duration, and special instructions.
4. **Pet and Owner Data Management**: Securely store and retrieve pet and owner information in a database, ensuring data integrity and validation.
5. **Real-Time Booking Tracking**: Maintain live updates for grooming appointments and boarding slots, preventing double bookings.
6. **Exception Handling:** Manage errors like invalid entries, duplicate bookings, or unavailable slots to ensure smooth operations.
7. **Multithreading:** Support concurrent bookings and cancellations to handle high-demand scenarios efficiently.
8. **File Handling**: Provide offline storage and backup for booking histories and pet-owner records for future reference.

4.2 USER INTERFACE

Provide an intuitive, console-based interface for administrators to manage booking operations efficiently.

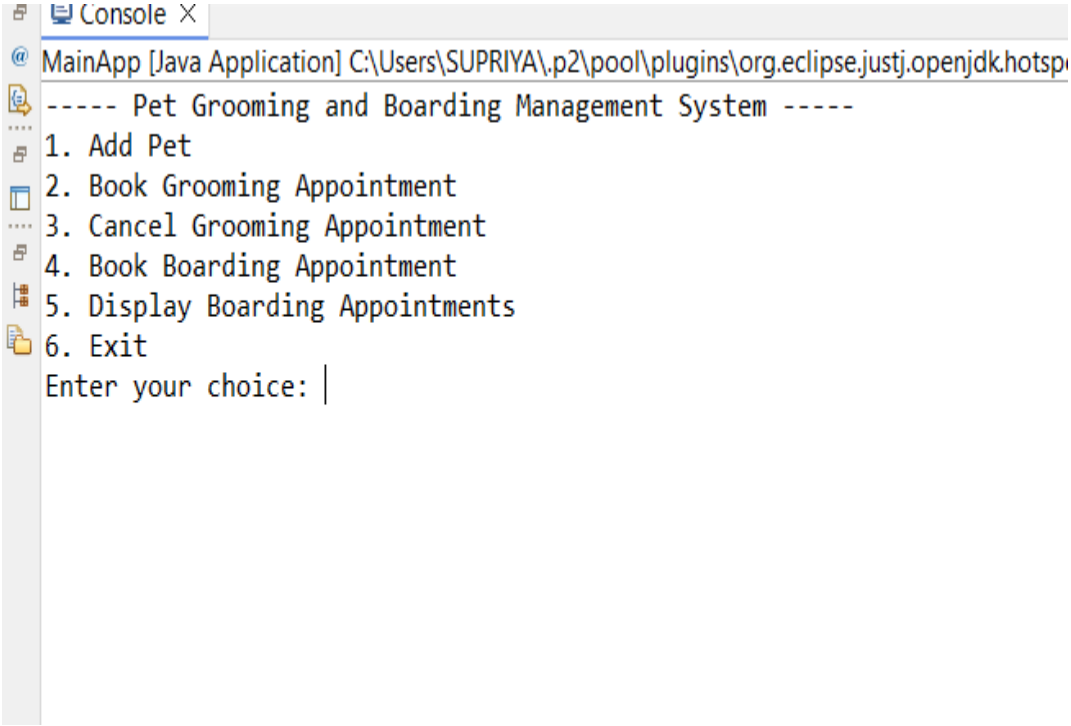

Figure 4.2 USER INTERFACE

CHAPTER 5

PROJECT DESCRIPTION

5.1 MODULE DESCRIPTION

1. Pet Management Module: Manages pet information, including adding, viewing, and updating pet details.
2. Owner Management Module: Handles pet owner registration, login, and profile management.
3. Grooming Module: Facilitates booking, canceling, and displaying grooming appointments.
4. Boarding Module: Manages booking and displaying boarding reservations, including special instructions.
5. Payment Module: Calculates service charges for grooming and boarding and manages payment confirmations securely.
6. Database Module: Handles data storage and retrieval operations for pets, owners, and bookings using MySQL.
7. File Handling Module: Provides backup of booking histories and pet-owner data for offline access.
8. Error Handling Module: Manages exceptions and displays user-friendly error messages.

5.2 JDBC CONNECTIVITY

```
package com.petgrooming;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {

   public static Connection getConnection() throws SQLException {
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         return
DriverManager.getConnection("jdbc:mysql://localhost:3306/petgroomingdb", "root",
"supriyakumar10");
      } catch (ClassNotFoundException | SQLException e) {
         throw new SQLException("Error connecting to the database: " + e.getMessage());
      }
   }
}
```

CHAPTER 6
IMPLEMENTATION

Sample Code:

```
package com.petgrooming;

import java.sql.*;
import java.util.Scanner;

public class MainApp {

   private static Connection getConnection() throws SQLException {
      // Replace with your MySQL credentials
      return DriverManager.getConnection("jdbc:mysql://localhost:3306/petgroomingdb",
"root", "supriyakumar10");
   }

   private static void addPet(Scanner scanner) {
      try (Connection conn = getConnection()) {
         // Get pet ID and owner ID at runtime
         System.out.print("Enter pet ID: ");
         int petId = scanner.nextInt();
         scanner.nextLine();  // Consume newline character

         System.out.print("Enter pet name: ");
         String petName = scanner.nextLine();
         System.out.print("Enter pet type (e.g., Dog, Cat): ");
         String petType = scanner.nextLine();

         // Check if the pet already exists by pet_id
         String checkPetQuery = "SELECT * FROM pets WHERE pet_id = ?";
         try (PreparedStatement checkStmt = conn.prepareStatement(checkPetQuery)) {
            checkStmt.setInt(1, petId);
            ResultSet rs = checkStmt.executeQuery();
```

```
            if (rs.next()) {
                System.out.println("Pet with ID " + petId + " already exists.");
                return;
            }
        }

        // Prompt for owner ID and ensure owner exists
        System.out.print("Enter owner ID: ");
        int ownerId = scanner.nextInt();
        scanner.nextLine();  // Consume newline character

        String checkOwnerQuery = "SELECT * FROM owners WHERE owner_id = ?";
        try (PreparedStatement checkStmt = conn.prepareStatement(checkOwnerQuery))
{
            checkStmt.setInt(1, ownerId);
            ResultSet rs = checkStmt.executeQuery();
            if (!rs.next()) {
                System.out.print("Owner with ID " + ownerId + " does not exist. Please
enter owner details.\n");
                System.out.print("Enter owner name: ");
                String ownerName = scanner.nextLine();
                System.out.print("Enter contact number: ");
                String contactNumber = scanner.nextLine();

                String insertOwnerQuery = "INSERT INTO owners (owner_id, name,
phone) VALUES (?, ?, ?)";
                try (PreparedStatement insertOwnerStmt =
conn.prepareStatement(insertOwnerQuery)) {
                    insertOwnerStmt.setInt(1, ownerId);
                    insertOwnerStmt.setString(2, ownerName);
                    insertOwnerStmt.setString(3, contactNumber);
                    insertOwnerStmt.executeUpdate();
                    System.out.println("Owner added successfully.");
                }
```

```
            }
        }

        // Insert the new pet
        String insertPetQuery = "INSERT INTO pets (pet_id, pet_name, pet_type,
owner_id) VALUES (?, ?, ?, ?)";
        try (PreparedStatement insertStmt = conn.prepareStatement(insertPetQuery)) {
            insertStmt.setInt(1, petId);
            insertStmt.setString(2, petName);
            insertStmt.setString(3, petType);
            insertStmt.setInt(4, ownerId);
            int rowsAffected = insertStmt.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Pet " + petName + " added successfully.");
            } else {
                System.out.println("Failed to add pet.");
            }
        }
    } catch (SQLException e) {
        System.out.println("Error: " + e.getMessage());
    }
}

private static void bookGroomingAppointment(Scanner scanner) {
    try (Connection conn = getConnection()) {
        // Get pet ID and owner ID at runtime
        System.out.print("Enter pet ID: ");
        int petId = scanner.nextInt();
        System.out.print("Enter owner ID: ");
        int ownerId = scanner.nextInt();
        scanner.nextLine();  // Consume newline character

        System.out.print("Enter grooming date (YYYY-MM-DD): ");
        String groomingDateStr = scanner.nextLine();
```

```
            Date groomingDate = Date.valueOf(groomingDateStr);

            System.out.print("Enter grooming type (e.g., Bath, Haircut): ");
            String groomingType = scanner.nextLine();

            // Insert grooming appointment into database
            String insertGroomingQuery = "INSERT INTO groomingappointments (pet_id,
owner_id, grooming_date, grooming_type) VALUES (?, ?, ?, ?)";
            try (PreparedStatement insertStmt =
conn.prepareStatement(insertGroomingQuery)) {
                insertStmt.setInt(1, petId);
                insertStmt.setInt(2, ownerId);
                insertStmt.setDate(3, groomingDate);
                insertStmt.setString(4, groomingType);
                int rowsAffected = insertStmt.executeUpdate();
                if (rowsAffected > 0) {
                    System.out.println("Grooming appointment booked successfully.");
                } else {
                    System.out.println("Failed to book grooming appointment.");
                }
            }
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void bookBoardingAppointment(Scanner scanner) {
        try (Connection conn = getConnection()) {
            // Get pet ID and owner ID at runtime
            System.out.print("Enter pet ID: ");
            int petId = scanner.nextInt();
            System.out.print("Enter owner ID: ");
            int ownerId = scanner.nextInt();
            scanner.nextLine();  // Consume newline character
```

```
        System.out.print("Enter check-in date (YYYY-MM-DD): ");
        String checkInDateStr = scanner.nextLine();
        Date checkInDate = Date.valueOf(checkInDateStr);

        System.out.print("Enter check-out date (YYYY-MM-DD): ");
        String checkOutDateStr = scanner.nextLine();
        Date checkOutDate = Date.valueOf(checkOutDateStr);

        System.out.print("Enter boarding type: ");
        String boardingType = scanner.nextLine();

        System.out.print("Enter special instructions: ");
        String specialInstructions = scanner.nextLine();

        // Insert boarding appointment into database
        String insertBoardingQuery = "INSERT INTO boardingbookings (pet_id,
owner_id, check_in_date, check_out_date, boarding_type, special_instructions)
VALUES (?, ?, ?, ?, ?, ?)";
        try (PreparedStatement insertStmt =
conn.prepareStatement(insertBoardingQuery)) {
            insertStmt.setInt(1, petId);
            insertStmt.setInt(2, ownerId);
            insertStmt.setDate(3, checkInDate);
            insertStmt.setDate(4, checkOutDate);
            insertStmt.setString(5, boardingType);
            insertStmt.setString(6, specialInstructions);
            int rowsAffected = insertStmt.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Boarding appointment booked successfully.");
            } else {
                System.out.println("Failed to book boarding appointment.");
            }
        }
```

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

    private static void displayBoardingAppointments() {
        try (Connection conn = getConnection()) {
            // Display all boarding appointments
            String query = "SELECT * FROM boardingbookings";
            try (Statement stmt = conn.createStatement()) {
                ResultSet rs = stmt.executeQuery(query);
                while (rs.next()) {
                    System.out.println("Pet ID: " + rs.getInt("pet_id"));
                    System.out.println("Owner ID: " + rs.getInt("owner_id"));
                    System.out.println("Check-in: " + rs.getDate("check_in_date"));
                    System.out.println("Check-out: " + rs.getDate("check_out_date"));
                    System.out.println("Boarding Type: " + rs.getString("boarding_type"));
                    System.out.println("Special Instructions: " +
rs.getString("special_instructions"));
                    System.out.println();
                }
            }
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("----- Pet Grooming and Boarding Management System -----
");
            System.out.println("1. Add Pet");
            System.out.println("2. Book Grooming Appointment");
```

```
        System.out.println("3. Cancel Grooming Appointment");
        System.out.println("4. Book Boarding Appointment");
        System.out.println("5. Display Boarding Appointments");
        System.out.println("6. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                addPet(scanner);
                break;
            case 2:
                bookGroomingAppointment(scanner);
                break;
            case 3:
                cancelGroomingAppointment(scanner);  // Call the cancel method here
                break;
            case 4:
                bookBoardingAppointment(scanner);
                break;
            case 5:
                displayBoardingAppointments();
                break;
            case 6:
                System.out.println("Exiting the system.");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice, please try again.");
        }
    }
}


private static void cancelGroomingAppointment(Scanner scanner) {
```

```
        try (Connection conn = getConnection()) {
            // Get grooming appointment ID to cancel
            System.out.print("Enter grooming appointment ID to cancel: ");
            int groomingId = scanner.nextInt();

            // Check if the grooming appointment exists
            String checkGroomingQuery = "SELECT * FROM groomingappointments
WHERE grooming_id = ?";
            try (PreparedStatement checkStmt =
conn.prepareStatement(checkGroomingQuery)) {
                checkStmt.setInt(1, groomingId);
                ResultSet rs = checkStmt.executeQuery();
                if (!rs.next()) {
                    System.out.println("Grooming appointment with ID " + groomingId + " does
not exist.");
                    return;
                }
            }

            // Delete the grooming appointment
            String deleteGroomingQuery = "DELETE FROM groomingappointments
WHERE grooming_id = ?";
            try (PreparedStatement deleteStmt =
conn.prepareStatement(deleteGroomingQuery)) {
                deleteStmt.setInt(1, groomingId);
                int rowsAffected = deleteStmt.executeUpdate();
                if (rowsAffected > 0) {
                    System.out.println("Grooming appointment with ID " + groomingId + " has
been cancelled successfully.");
                } else {
                    System.out.println("Failed to cancel grooming appointment.");
                }
            }
        } catch (SQLException e) {
```

```
            System.out.println("Error: " + e.getMessage());
        }
    }


}
```

SQL code

```
-- Create owners table
CREATE TABLE owners (
    owner_id INT PRIMARY KEY,
    owner_name VARCHAR(255),
    contact_number VARCHAR(20)
);

-- Create pets table
CREATE TABLE pets (
    pet_id INT PRIMARY KEY,
    pet_name VARCHAR(255),
    pet_type VARCHAR(50),
    owner_id INT,
    FOREIGN KEY (owner_id) REFERENCES owners(owner_id)
);

-- Create groomingappointments table
CREATE TABLE groomingappointments (
    appointment_id INT AUTO_INCREMENT PRIMARY KEY,
    pet_id INT,
    owner_id INT,
    service_type VARCHAR(255),
    status VARCHAR(255),
    notes TEXT,
    appointment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    grooming_date DATE,  -- Added column for grooming date
    grooming_type VARCHAR(255),  -- Added column for grooming type
    grooming_id INT UNIQUE,  -- Added grooming_id column with a unique
constraint
    FOREIGN KEY (pet_id) REFERENCES pets(pet_id),
    FOREIGN KEY (owner_id) REFERENCES owners(owner_id)
);
```

```
-- Create boardingbookings table
CREATE TABLE boardingbookings (
    booking_id INT AUTO_INCREMENT PRIMARY KEY,
    pet_id INT,
    owner_id INT,
    check_in_date DATE,
    check_out_date DATE,
    boarding_type VARCHAR(255),
    special_instructions TEXT,
    status VARCHAR(255) DEFAULT 'Pending',  -- Added status for the boarding
booking
    FOREIGN KEY (pet_id) REFERENCES pets(pet_id),
    FOREIGN KEY (owner_id) REFERENCES owners(owner_id)
);

-- Sample data for owners table
INSERT INTO owners (owner_id, owner_name, contact_number) VALUES
(1, 'John Doe', '9876543210'),
(2, 'Jane Smith', '9876549876'),
(3, 'Ana', '3456798367');

-- Sample data for pets table
INSERT INTO pets (pet_id, pet_name, pet_type, owner_id) VALUES
(10, 'Thara', 'Dog', 1),
(11, 'Lizzy', 'Cat', 2),
(12, 'Lucy', 'Dog', 3);

-- Sample data for groomingappointments table
INSERT INTO groomingappointments (pet_id, owner_id, service_type, status, notes,
grooming_date, grooming_type, grooming_id) VALUES
(10, 1, 'Full Grooming', 'Pending', 'Fast grooming', '2024-11-20', 'Full Grooming',
101),
(11, 2, 'Haircut', 'Completed', 'Needs trimming', '2024-11-19', 'Haircut', 102),
```

```
(12, 3, 'Bath', 'Pending', 'Quick bath', '2024-11-21', 'Bath', 103);

-- Sample data for boardingbookings table
INSERT INTO boardingbookings (pet_id, owner_id, check_in_date, check_out_date,
boarding_type, special_instructions) VALUES
(10, 1, '2024-11-22', '2024-11-25', 'Standard Boarding', 'Needs extra blankets'),
(11, 2, '2024-11-23', '2024-11-26', 'Luxury Boarding', 'Requires medication during
stay'),
(12, 3, '2024-11-24', '2024-11-27', 'Premium Boarding', 'Food preferences:
vegetarian');

-- Check the created tables and the data inserted
use PetGroomingDB;
SHOW TABLES;

-- Retrieve all groomingappointments
SELECT * FROM groomingappointments;

-- Retrieve all boardingbookings
SELECT * FROM boardingbookings;

-- Retrieve all pets and their owners
SELECT pets.pet_id, pets.pet_name, pets.pet_type, owners.owner_name
FROM pets
JOIN owners ON pets.owner_id = owners.owner_id;

-- Retrieve all boarding details with pet and owner information
SELECT boardingbookings.booking_id, boardingbookings.check_in_date,
boardingbookings.check_out_date, boardingbookings.boarding_type,
boardingbookings.special_instructions, pets.pet_name, owners.owner_name
FROM boardingbookings
JOIN pets ON boardingbookings.pet_id = pets.pet_id
JOIN owners ON boardingbookings.owner_id = owners.owner_id;
```

CHAPTER 7
RESULTS AND DISCUSSIONS

```
----- Pet Grooming and Boarding Management System -----
1. Add Pet
2. Book Grooming Appointment
3. Cancel Grooming Appointment
4. Book Boarding Appointment
5. Display Boarding Appointments
6. Exit
Enter your choice: 1
Enter pet ID: 341
Enter pet name: Jenny
Enter pet type (e.g., Dog, Cat): Dog
Enter owner ID: 143
Owner with ID 143 does not exist. Please enter owner details.
Enter owner name: Supriya
Enter contact number: 9908765467
Owner added successfully.
Pet Jenny added successfully.
----- Pet Grooming and Boarding Management System -----
1. Add Pet
2. Book Grooming Appointment
3. Cancel Grooming Appointment
4. Book Boarding Appointment
5. Display Boarding Appointments
6. Exit
Enter your choice: 2
Enter pet ID: 341
Enter owner ID: 143
Enter grooming date (YYYY-MM-DD): 2024-09-09
Enter grooming type (e.g., Bath, Haircut): Bath
Grooming appointment booked successfully.
```

```
----- Pet Grooming and Boarding Management System -----
1. Add Pet
2. Book Grooming Appointment
3. Cancel Grooming Appointment
4. Book Boarding Appointment
5. Display Boarding Appointments
6. Exit
Enter your choice: 5
Pet ID: 123
Owner ID: 321
Check-in: 2024-09-23
Check-out: 2023-09-25
Boarding Type: Full care
Special Instructions: nothing

Pet ID: 98
Owner ID: 89
Check-in: 2024-08-08
Check-out: 2024-08-09
Boarding Type: full care
Special Instructions: Feed him every 2hours

Pet ID: 988
Owner ID: 889
Check-in: 2024-09-09
Check-out: 2024-09-22
Boarding Type: Full care
Special Instructions: Nothing

----- Pet Grooming and Boarding Management System -----
1. Add Pet
2. Book Grooming Appointment
3. Cancel Grooming Appointment
4. Book Boarding Appointment
5. Display Boarding Appointments
6. Exit
Enter your choice: 6
Exiting the system.
```

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

The Pet Grooming and Boarding Management System offers an efficient and automated platform for managing grooming appointments, boarding reservations, and pet-owner data. By integrating a user-friendly interface with a robust backend database, the system addresses challenges such as manual scheduling conflicts, inaccurate data handling, and operational delays. Real-time slot allocation ensures smooth operations and optimal resource utilization, while secure data storage safeguards sensitive pet and owner details. Multithreading facilitates concurrent operations, ensuring minimal delays even during peak times. With its modular design and features like automated scheduling, payment management, and error handling, the system improves service efficiency and enhances customer satisfaction. Overall, the system streamlines pet grooming and boarding operations, enabling pet salons and boarding facilities to deliver reliable and professional services.

FUTURE ENHANCEMENTS

- **Mobile App Integration**: Develop a mobile application to enable pet owners to book grooming and boarding appointments, view their schedules, and manage payments conveniently via smartphones.
- **Advanced Notification System**: Implement automated notifications for appointment reminders, payment updates, and upcoming bookings via SMS or email.
- **Dynamic Slot Allocation**: Introduce a dynamic slot allocation system to optimize resource usage based on demand and pet preferences.
- **Payment Gateway Integration**: Add support for online payments using popular methods such as credit/debit cards and digital wallets, ensuring secure and hassle-free transactions.
- **Real-Time Availability Updates**: Provide live updates on grooming and boarding slot availability, ensuring accurate information and minimizing double bookings.

- **Enhanced Reporting and Analytics**: Incorporate advanced reporting features to track appointment trends, revenue, and service utilization, helping facility owners make data-driven decisions.
- **Owner Profile Management**: Introduce personalized user profiles for pet owners to save details about their pets, track service history, and receive tailored recommendations for grooming or boarding services.
- **Emergency Care Integration**: Include features to manage urgent pet care requests or emergency boarding requirements, ensuring comprehensive service delivery.

REFERENCE

- JDBC connectivity Javatpoint:

  Java Database Connectivity with MySQL - javatpoint

  Complex SQL Queries | Complex SQL Queries for Practice

- Java console based application Java Console Programming – JavaTPoint
  - Stack Overflow
  - Reddit - Learn Java