**VIETNAMESE – GERMAN UNIVERSITY**

FACULTY OF ENGINEERING

COMPUTER SCIENCE DEPARTMENT

JAVA PROJECT REPORT

PET SHOP

***Module 61CSE215: Object Oriented Programming in JAVA***

1. Nguyễn Thanh Sơn – 10422072
2. Nguyễn Thanh Tú – 10422080
3. Trần Sĩ Nguyên – 10422057
4. Võ Tấn Sang – 10422114

Lecturer: Dr. Tran Hong Ngoc

Binh Duong, WS2024

Versions

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | In charge | Description |
| 19-October 24 | V1.0 | Vo Tan Sang, Tran Si Nguyen, Nguyen Thanh Tu, Nguyen Thanh Son | Update Section I, II and Appendix A |
| 04-November 24 | V2.0 | Vo Tan Sang, Tran Si Nguyen, Nguyen Thanh Tu, Nguyen Thanh Son | Update Section III and VI |
| 09-November 24 | V3.0 | Vo Tan Sang, Tran Si Nguyen, Nguyen Thanh Tu, Nguyen Thanh Son | Update Section III, IV, V, VIII |
| 16-November 24 | V3.5 | Vo Tan Sang, Tran Si Nguyen, Nguyen Thanh Tu, Nguyen Thanh Son | Update Section VII, Appendix B |

Abbreviation

DMS Database Management System

OOP Object-Oriented Programming

HTTP Hypertext Transfer Protocol

UI User Interface

List of Figures

[Figure 1 Usecase Diagram 7](#_Toc182917345)

[Figure 2. Diagram of Abstract Class Person 10](#_Toc182917346)

[Figure 3. Pet Information 25](#_Toc182917347)

[Figure 4. User 25](#_Toc182917348)

[Figure 5. Login Interface 25](#_Toc182917349)

[Figure 6. To-Do Tasks Interface 26](#_Toc182917350)

[Figure 7. Edit Task Interface 26](#_Toc182917351)

[Figure 8. Delete Task Interface 27](#_Toc182917352)

[Figure 9. Class Diagram 32](#_Toc182917353)

[Figure 10. UI Diagram 33](#_Toc182917354)

List of Tables

[Table 1. List of Objects (Product1, Employee1) 8](#_Toc182918224)

[Table 2. Details of Classes 11](#_Toc182918225)

[Table 3. Details of Abstract class 11](#_Toc182918226)

[Table 4. Data Accesss Control Table 17](#_Toc182918227)

[Table 5. Method Access Control Table 18](#_Toc182918228)

[Table 6. List of Primary Function 23](#_Toc182918229)

[Table 7. Table of User Database 30](#_Toc182918230)

[Table 8. Table of Pet Information 30](#_Toc182918231)

Table of Contents

Contents

[I. INTRODUCTION 7](#_Toc182919083)

[II. CLASS ANALYSIS 8](#_Toc182919084)

[1. Objects 8](#_Toc182919085)

[2. Classes 8](#_Toc182919086)

[III. CLASS DESIGN 10](#_Toc182919087)

[1. Classes 10](#_Toc182919088)

[2. OOP Techniques 11](#_Toc182919089)

[2.1 Overloading Methods 11](#_Toc182919090)

[2.2. Overriding Methods 12](#_Toc182919091)

[3. Inheritance 13](#_Toc182919092)

[IV. Package Design 15](#_Toc182919093)

[Package Descriptions 15](#_Toc182919094)

[com.petshop.controllers 15](#_Toc182919095)

[com.petshop.models 15](#_Toc182919096)

[com.petshop.services 15](#_Toc182919097)

[com.petshop.dao 15](#_Toc182919098)

[com.petshop.mapper 15](#_Toc182919099)

[V. Interface Design 16](#_Toc182919100)

[1.Login Interface 16](#_Toc182919101)

[2. Dashboard Interface 16](#_Toc182919102)

[3. Pet Management Interface 16](#_Toc182919103)

[VI. Access Control 17](#_Toc182919104)

[VII. Encapsulation vs Inheritance vs Polymorphism 18](#_Toc182919105)

[1. Encapsulation 18](#_Toc182919106)

[2. Inheritance 19](#_Toc182919107)

[3. Polymorphism 21](#_Toc182919108)

[VIII. Experiment 22](#_Toc182919109)

[1. Environment and Tools 22](#_Toc182919110)

[1.1. Environment**:** 22](#_Toc182919111)

[1.2. Tools: 22](#_Toc182919112)

[2. Project functions 23](#_Toc182919113)

[3. Database 23](#_Toc182919114)

[4. GUI 24](#_Toc182919115)

[IX. Conclusion 26](#_Toc182919116)

[1. Assessment 26](#_Toc182919117)

[2. Project Quality Score 27](#_Toc182919118)

[3. Future Improvements 27](#_Toc182919119)

[APPENDIX A: DATABASE 30](#_Toc182919120)

[1. Tables: 30](#_Toc182919121)

[2. System Functionalities 30](#_Toc182919122)

[3. Class Diagram 31](#_Toc182919123)

[APPENDIX B: GRAPHICAL USER INTERFACE (GUI) 32](#_Toc182919124)

# I. INTRODUCTION

In this project, we want to create a Pet Shop website with the goal of streamlining and simplifying pet store operations. The system will assist routine tasks like keeping track of pet inventory, and managing user information. The objective of this platform is to give the pet shop employees an easy-to-use interface so they can effectively manage their inventory, reservations, and client interactions.

This website will offer the following core business functions:

* Input and Update pet information
* User access control

In addition to the core functions, the Pet Shop website will also have some functions that allow the users to:

* Search for pets by ID.
* Generate To-do list

For this project, we use Java OOP for designing classes and applying the OOP such as abstraction, encapsulation, inheritance, and polymorphism.

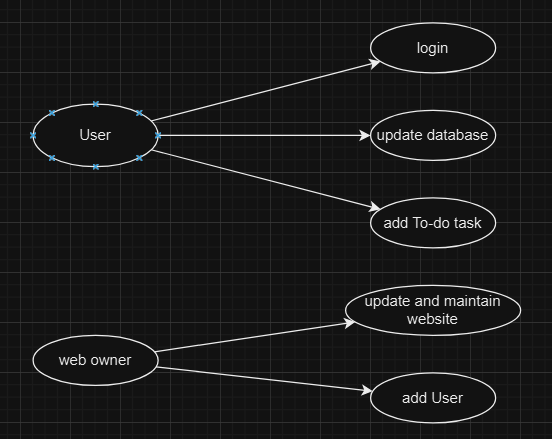


Figure 1 Usecase Diagram

# II. CLASS ANALYSIS

## 1. Objects

|  |  |  |  |
| --- | --- | --- | --- |
| No | Object Name | State | Behaviors |
| 1 | Product1 | Name, Price, Type, Breed, Age, Gender, Description, Added By | Available for purchase, Update Information |
| 2 | Employee1 | Name, Role, Email | Manage inventory, Process checkout |

Table 1. List of Objects (Product1, Employee1)

## 2. Classes

The following classes represent the different entities in the system. Each class has attributes and functions (to be defined during implementation).

* Product:
  + Attributes: Name, Price, Type, Breed, Age, Gender, Description, Added By
  + Methods: Update availability, Display info
* Employee:
  + Attributes: Name, Role, To do list, Email
  + Methods: Manage inventory, Process sales

**Inheritance Analysis:**

(Abstract Class):

* + Attributes:
    - ID: The ID of object.
    - List<objects>: List of object.
  + Methods: getter(), setter()

Staff (Inherits from (Abstract Class)):

* + Attributes:
    - name: The name of the employee
    - role: The role of the staff member in the shop (e.g., manager, sales assistant).
    - email: The contact email of the staff.
  + Methods: getter(), setter()

Product (Inherits from (Abstract Class)):

* + Attributes: Name, Price, Type, Breed, Age, Gender, Description, Added By.
  + Methods: getter(), setter()

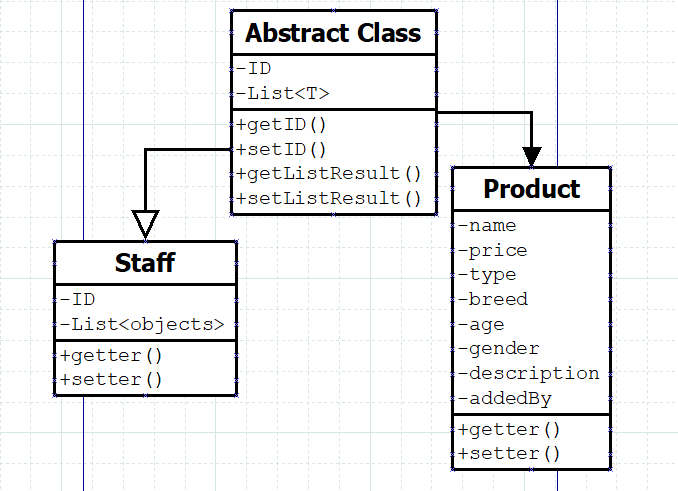


Figure 2. Diagram of Abstract Class Person

# III. CLASS DESIGN

## 1. Classes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Class | Instance Variable | *Methods* | *Description* |
| 1 | User (Employee) | - private String username  - private String email  - private String role | - public String getRole()  - public void setRole(String role)  - public String getUsername()  - public void setUsername(String username)  - public String getEmail()  - public void setEmail(String email) | *This class manages employee information and their role details.* |
| 2 | Product (Pet) | - private String name;  - private Double price;  - private String type;  - private String breed;  - private int age;  - private String gender;  - private String description;  - private String addedBy; | *- public String getName()*  *- public void setName*  *- public Double getPrice()*  *- public void setPrice(Double price)*  *- public String getType()*  *- public String getBreed()*  *- public int getAge()*  *- public void setAge(int age)*  *- public String getGender()*  *- public void setGender(String gender)*  *- public String getDescription()*  *- public void setDescription(String description)*  *- public String getAddedBy()*  *public void setAddedBy(String addedBy)* | *This class handles product information and inventory details.* |

Table 2. Details of Classes

*Abstract class*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Abstract class | - private int ‘id’; - private List<T> listResult | - getId()  - setId(int id)  - List<T> getListResult()  - void setListResult(List<T> listResult) | Abstract class to manage common user details and authentication. |

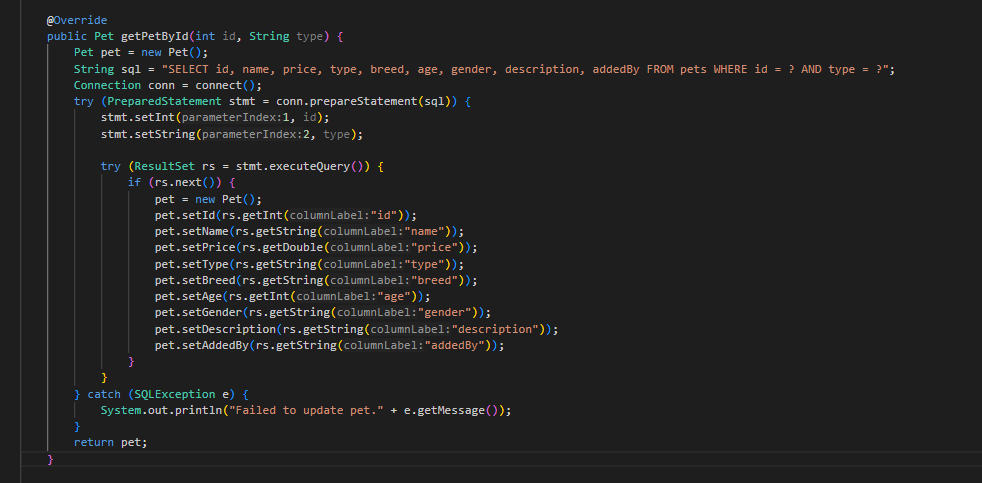
Table 3. Details of Abstract class

## 2. OOP Techniques

### 2.1 Overloading Methods

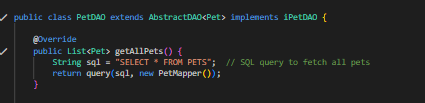
Method overloading allows multiple methods in the same class with the same name but different parameters.

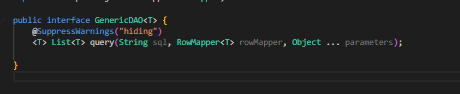




### 2.2. Overriding Methods

Method overriding allows a subclass to provide a specific implementation for a method defined in a parent class.









## 3. Inheritance

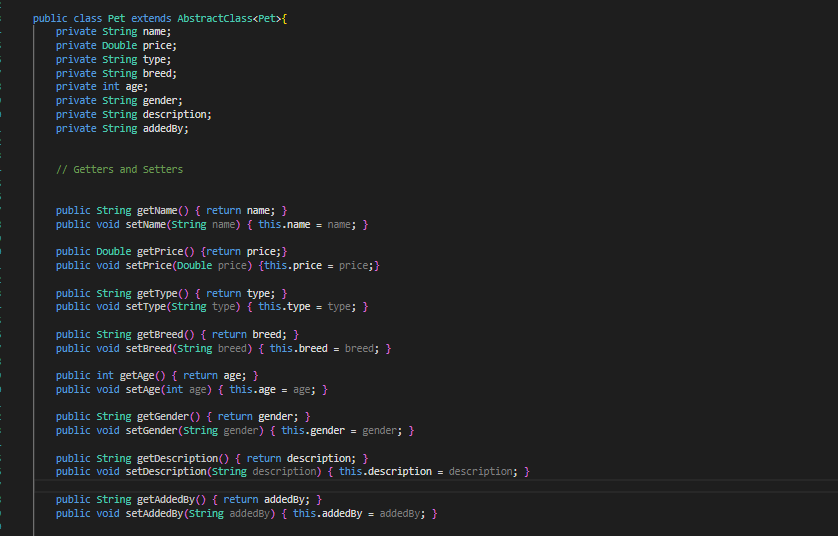
• Inheritance-Related Techniques:

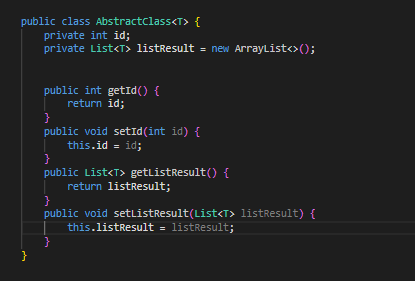
1. Single Inheritance: Each class inherits from the abstract class.

2. Method Overriding: Employee overrides displayInfo() and updateState() for Pet.

3. Constructor Chaining: Subclasses use constructors to initialize both inherited and new fields.

Representative Code: Inheritance





# IV. Package Design

## Package Descriptions

### com.petshop.controllers

This package contains the controller classes, which handle incoming HTTP requests, process the data, and return appropriate responses to the client. Each controller class corresponds to a specific entity in the system (e.g., PetController manages pet-related actions). Example: PetController contains endpoints to search, add, update, and delete pet records.

### com.petshop.models

This package holds the entity classes, representing the database tables and containing the core properties of each object in the pet shop (e.g., Pet, Customer). Each class maps directly to a database table and includes fields that define the structure of each record. Example: Pet has fields such as id, name, type, breed, and age.

### com.petshop.services

This package includes service classes that contain the business logic of the application. Services interact with repositories and perform operations before passing data to controllers. Each service corresponds to an entity and has methods to handle complex business logic and data processing. Example: PetService contains methods to handle pet searches, add new pets, and validate data before saving.

### com.petshop.dao

This package is responsible for handling database operations and interactions. It includes classes and methods that manage CRUD (Create, Read, Update, Delete) operations for entities such as pets, users, or tasks. This package serves as the bridge between the application and the database, ensuring data is correctly saved, retrieved, and updated.

### com.petshop.mapper

This package is responsible for mapping database records to Java objects and vice versa. It includes classes or interfaces that convert data from relational database formats (e.g., SQL result sets) into object-oriented models (e.g., Pet or User objects). This package is crucial for simplifying data handling and maintaining the integrity of application models.

# V. Interface Design

In this section, we outline the interface design used in the pet shop management project. The interfaces are designed to provide a smooth and intuitive user experience, allowing users to manage pets, customers, appointments, and sales efficiently. Below are the key interfaces and their functionalities.

1.Login Interface

The login screen is the first interface that users encounter. It requires users to enter their email and password to access the system.

Fields:

Email: Input field for user’s email.

Password: Input field for user’s password (masked for privacy).

Buttons:

Login: Authenticates user and navigates to the dashboard upon successful login.

Forgot Password: Allows users to reset their password if they’ve forgotten it.

2. Dashboard Interface

After logging in, users land on the dashboard. It provides an overview of key metrics and quick access to core functionalities.

Buttons:

Pet Summary: Displays total number of pets, available pets, and newly added pets.

Upcoming Appointments: Lists upcoming appointments for easy tracking.

Quick Links: Shortcuts to frequently used sections like adding a pet, viewing customer details, and scheduling appointments.

3. Pet Management Interface

This interface allows users to manage pet details, including adding, updating, or deleting pet records.

Fields:

Pet ID: Unique identifier for each pet (auto-generated).

Name: Input field for the pet's name.

Type: Dropdown menu for pet types (e.g., Dog, Cat, Bird).

Breed: Input field for pet’s breed.

Age: Input field for pet’s age.

Owner: Dropdown menu or autocomplete to select the pet’s owner.

Buttons:

Save: Saves new or updated pet information.

Delete: Deletes a pet’s record (with a confirmation prompt).

Search: Allows users to search for pets by name, type, or breed.

# VI. Access Control

In the Pet Shop Management System, access control is essential to safeguard data and ensure that only appropriate methods, classes, and interfaces have access to specific data. This section presents tables detailing the access control used for data and methods, highlighting the use of public, private, and protected access modifiers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Data | Class | Modifier | Description |
| 1 | username, password, role, to-do work | Staff | Private | Only accessible within Staff to maintain data security for staff information. |
| 2 | species, price, age, breed, gender, description | Pet | Public | Non-sensitive data; public for easy access by customers and staff. |

Table 4. Data Accesss Control Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Method | Class | Modifier | Description |
| 1 | searchPets() | Staff | Private | Accessible by customers for pet browsing and interactions. |
| 2 | manageInventory() | Staff | Private | Accessible within the package; ensures only authorized staff manage inventory. |
| 3 | updateAvailability() | Pet | Private | Enables updating pet availability; public for system interactions. |

Table 5. Method Access Control Table

# VII. Encapsulation vs Inheritance vs Polymorphism

## 1. Encapsulation

public interface iPetDAO extends GenericDAO<Pet> {

boolean addpet(Pet pet);

List<Pet> getAllPets();

}

**Encapsulation through Interfaces (`iPetDAO`):**  
- The `iPetDAO` interface defines a contract for any DAO class that handles `Pet` objects. The methods `addpet(Pet pet)` and `getAllPets()` expose only the necessary functionality that a client code might need. This ensures that the internal implementation in `PetDAO` remains hidden and can be modified without affecting external code.  
- This design achieves \*data hiding\* by keeping the database logic inside the `PetDAO` implementation.

import com.laptrinhjavaweb.dao.iPetDAO;

import com.laptrinhjavaweb.mapper.PetMapper;

import com.laptrinhjavaweb.model.Pet;

public class PetDAO extends AbstractDAO<Pet> implements iPetDAO {

    @Override

    public List<Pet> getAllPets() {

        String sql = "SELECT \* FROM PETS";  // SQL query to fetch all pets

        return query(sql, new PetMapper());

    }

    // Method to save a new pet to the database

    @Override

    public boolean addpet(Pet pet) {

        // String sql = "INSERT INTO PETS (?, ?, ?, ?, ?, ?, ?) VALUES" + "(?, ?, ?, ?, ?, ?, ?)";

        // Connection conn = connect();

        return true;

    }

}

**Encapsulation in the `PetDAO` Class:**  
- The `PetDAO` class implements the `iPetDAO` interface and provides concrete implementations for the methods. The database interaction, such as querying for pets or adding a pet, is encapsulated within these methods, ensuring that external classes do not directly interact with the database.  
- For example, the `getAllPets()` method executes an SQL query internally, but the caller does not need to know the SQL or database connection details. This simplifies the interface and promotes maintainability.

## 2. Inheritance

public class AbstractClass<T> {

private int id;

private List<T> listResult = new ArrayList<>();

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public List<T> getListResult() {

return listResult;

}

public void setListResult(List<T> listResult) {

this.listResult = listResult;

**}**

**}**

**Abstract Parent Class (AbstractClass):**

* The AbstractClass<T> serves as the base class for various entities in the system. It defines common attributes (e.g., id) and methods (e.g., save, delete) that all child classes must implement.
* This ensures that shared functionality is centralized, avoiding code duplication.

public class Pet extends AbstractClass<Pet>{

private String name;

private Double price;

private String type;

private String breed;

private int age;

private String gender;

private String description;

private String addedBy;

public String getName() { return name; }

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

public Double getPrice() {return price;}

public void setPrice(Double price) {this.price = price;}

public String getType() { return type; }

public void setType(String type) { this.type = type; }

public String getBreed() { return breed; }

public void setBreed(String breed) { this.breed = breed; }

public int getAge() { return age; }

public void setAge(int age) { this.age = age; }

public String getGender() { return gender; }

public void setGender(String gender) { this.gender = gender; }

public String getDescription() { return description; }

public void setDescription(String description) { this.description = description; }

public String getAddedBy() { return addedBy; }

public void setAddedBy(String addedBy) { this.addedBy = addedBy; }

}

**Child Class (Pet):**

* + The Pet class extends the AbstractClass<Pet> and inherits its attributes and methods. It also implements the abstract methods save and delete, providing specific behavior for saving and deleting pets in the system.
  + Additionally, the Pet class defines its unique attributes (e.g., name, price, breed) and their corresponding getter and setter methods.

## 3. Polymorphism

@WebServlet(urlPatterns = {"/admin-home"})

public class HomeController extends HttpServlet {

@Inject

private iNewService NewService;

private static final long serialVersionUID = 2686801510274002166L;

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

Pet petList = new Pet();

petList.setListResult(NewService.getAllPets());

request.setAttribute("petList",petList);

RequestDispatcher rd = request.getRequestDispatcher("/views/admin/home.jsp");

rd.forward(request, response);

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

}

}

**Method Overriding (Runtime Polymorphism):**

* The doGet and doPost methods in the HomeController class override the respective methods in the parent HttpServlet class.
* This demonstrates runtime polymorphism because the method behavior is determined at runtime based on the incoming HTTP request (e.g., GET or POST).

# VIII. Experiment

## 1. Environment and Tools

### 1.1. Environment**:**

* **Number of PCs**: 3 (used for testing and development)
* **CPU**: Intel Core i5-11400H (2 PCs) and Ryzen r7-5800H (1 PC)
* **RAM**: 16GB DDR4 on all PCs
* **Storage**: 512GB SSD on each PC
* **Operating System**: Windows 11 on two PCs and Windows 10 on one PC.

### 1.2. Tools:

* Java– for backend development
* JavaScript – for frontend development
* HTML **–** for frontend development
* CSS – for frontend development
* SQL – database for development

1. **Framework and Library**

* JSP – for frontend development and logical management
* Bootstrap – for styling the UI
* Spring boot – for logical management
* Maven – for organization

## 2. Project functions

In this section, we list and briefly describe the primary functions of the pet shop management system. The functions are designed to streamline the management of pets, customers, appointments, and sales transactions within a pet shop.

|  |  |
| --- | --- |
| Function | Description |
| Pet Search | Allows users to search for pets based on criteria like breed, type, age, and availability. |
| Add New Pet | Enables users to add new pet details, including name, breed, age, and owner information. |
| Update Pet Information | Allows users to update details of existing pets, such as changes in ownership or health status. |
| Delete Pet Record | Enables users to remove a pet's record from the system with a confirmation prompt. |

Table 6. List of Primary Function

## 3. Database

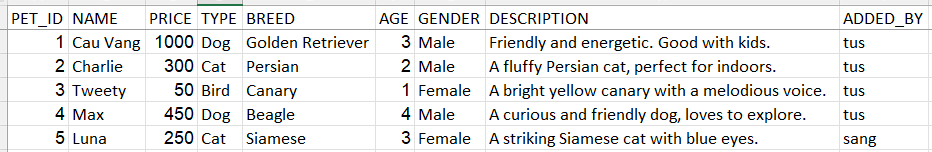


Figure 3. Pet Information

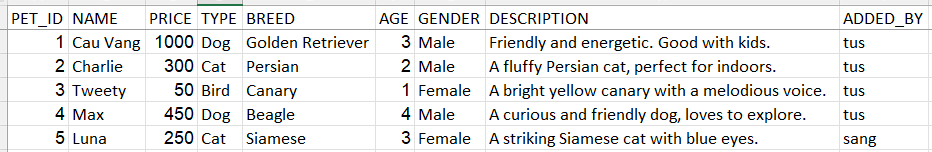


Figure 4. User

## 4. GUI

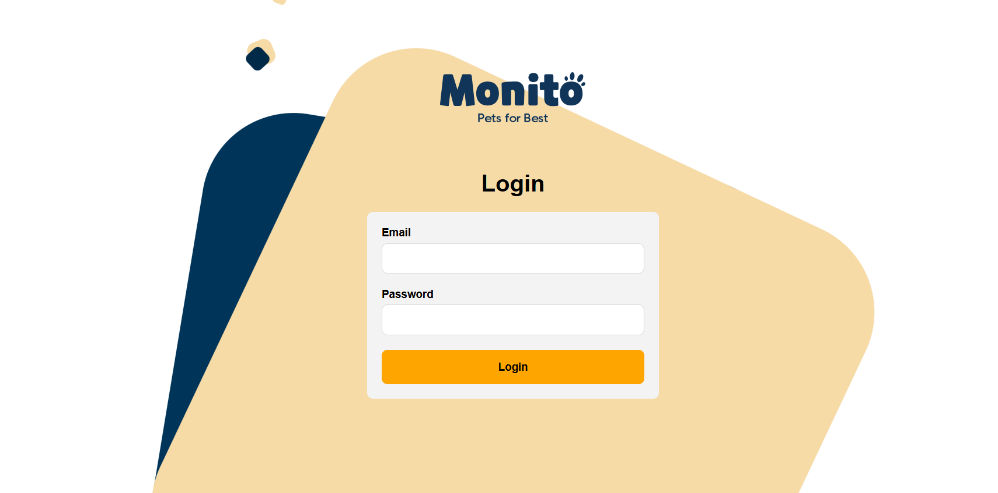


Figure 5. Login Interface

* This is a login interface for an application named "Monito". Email and Password Fields: These text input fields allow the user to enter their email address and password, respectively. When you fill out the email and password, the data will be stored into the database.
* Login Button: This button, in bright orange, triggers a login action. Once the email and password fields are filled out, pressing this button typically submits the credentials to the backend system for authentication.



Figure 6. To-Do Tasks Interface

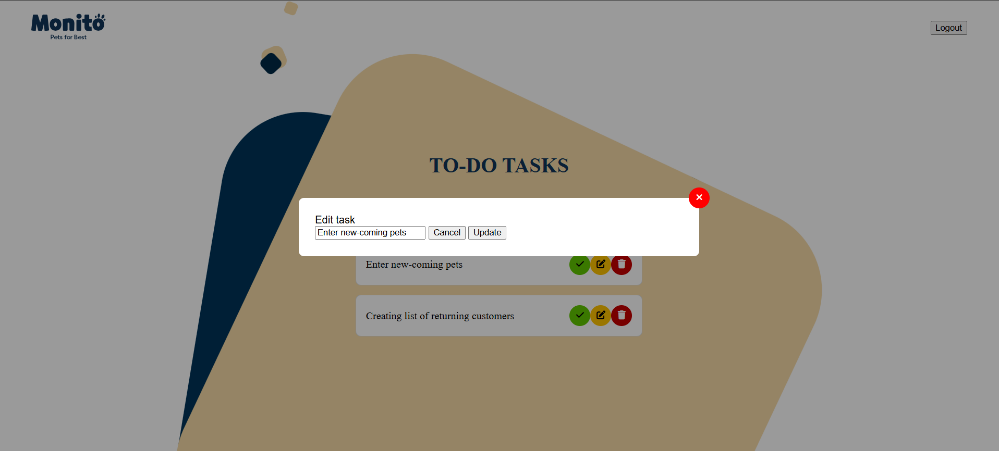
* To-do tasks interface shows the staffs the works that they have to finish within a day.

Figure 7. Edit Task Interface

* The yellow button is to change the task and update to the TO-DO TASKS.

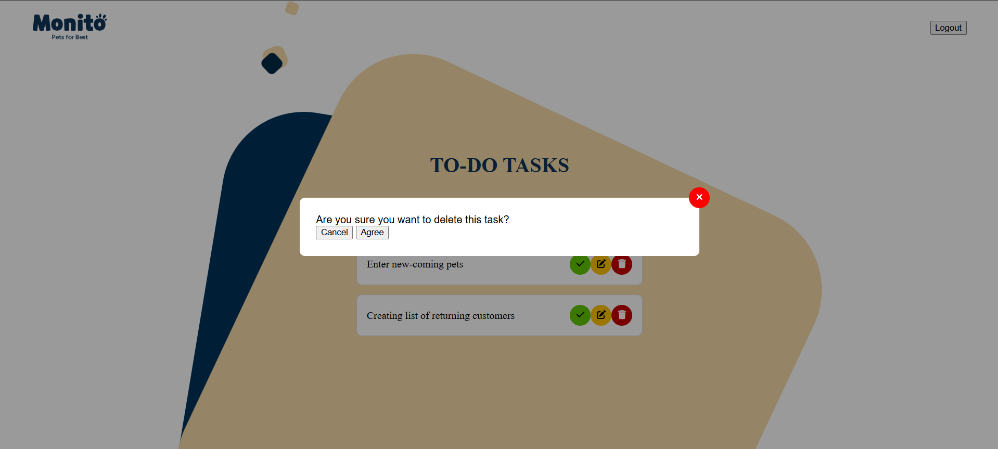


Figure 8. Delete Task Interface

* The function of the red button is to delete the task which is finished or unnecessary.

# IX. Conclusion

This project aimed to develop a Pet Shop Management System using Java OOP principles to streamline operations, enhance customer experience, and simplify backend processes. The system supports core functionalities like managing pets, customers, and bookings, integrating business logic, and presenting user-friendly interfaces.

## 1. Assessment

1. **Pros:**
   * The system successfully demonstrates core OOP principles such as encapsulation, inheritance, and polymorphism, ensuring a robust, modular, and reusable code structure.
   * It integrates a relational database with an efficient schema design, ensuring data consistency and ease of querying.
   * The user interface is intuitive, providing quick access to essential operations like pet management, customer details, and bookings.
   * Future scalability is ensured through the use of abstraction and a layered architecture.
2. **Cons:**
   * Some features, such as advanced reporting and real-time notifications, are currently absent and could enhance the system's utility.
   * The database design could include more detailed tracking of inventory and sales for improved analytics.
   * The interface design could be further optimized for mobile compatibility to extend its usability.

## 2. Project Quality Score

I would rate this project 7.5/10, based on its adherence to OOP principles, practical functionality, and future scalability. The score reflects the excellent foundational design while leaving room for improvement in extended features and interface polish.

## 3. Future Improvements

* **Additional Features:**
  + Add sales and revenue tracking functionalities to help the pet shop monitor performance.
  + Implement notifications for upcoming bookings and low inventory levels.
  + Develop a mobile-friendly interface for better accessibility.
* **Optimization:**
  + Integrate advanced search options using filters like price range, breed, and availability status.
  + Use cloud-based solutions to ensure better scalability and accessibility.
* **Technologies:**
  + Incorporate machine learning to suggest pet care tips or suitable products based on customer purchases.
  + Implement APIs for third-party integrations, such as payment gateways or delivery tracking.

In conclusion, the Pet Shop Management System achieves its core objectives, providing a solid framework for further enhancements. This project reflects an understanding of OOP principles and practical application in building scalable and maintainable software.

**REFERENCE**

1. CSS Tutorial, <https://www.w3schools.com/css/>
2. HTML Tutorial, <https://www.w3schools.com/html/>
3. MySQL Tutorial, <https://www.w3schools.com/mysql/default.asp>
4. Poo, D., Kiong, D., & Ashok, S. (2007). Object-Oriented programming and Java. In Springer eBooks.

# APPENDIX A: DATABASE

## 1. Tables:

* 1. Users Table

Stores information about the system users (e.g., admins, staff).

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| id | INT (PK) | Unique identifier for each user |
| name | VARCHAR(100) | Full name of the user |
| email | VARCHAR(100) | User email address |
| role | ENUM ('Admin', 'Staff') | Role of the user (Admin/Staff) |

Table 7. Table of User Database

* 1. Pets Table

Stores information about the pets available for sale or adoption.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| id | INT (PK) | Unique identifier for each pet |
| name | VARCHAR(100) | Name of the pet |
| price | DECIMAL(10,2) | Price of the pet |
| type | VARCHAR(50) | Type of pet (e.g., Dog, Cat) |
| breed | VARCHAR(100) | Breed of the pet |
| age | INT | Age of the pet in months |
| gender | ENUM ('Male', 'Female’) | Gender of the pet |
| description | VARCHAR(100) | Description or notes about the pet |
| addedBy | VARCHAR(100) | User who added the pet |

Table 8. Table of Pet Information

## 2. System Functionalities

* + 1. Admin Functionalities

1. Add, edit, or delete pet records in the database.
2. View a list of pets currently available in the system.
3. Manage user accounts, including adding or removing staff.
   * 1. User Functionalities
4. View available pets.
5. Update pet details, such as marking a pet as sold or updating prices.

## 3. Class Diagram

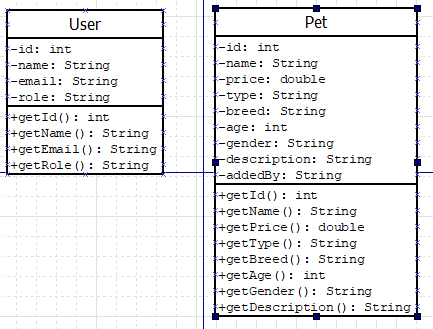


Figure 9. Class Diagram

# APPENDIX B: GRAPHICAL USER INTERFACE (GUI)

UI Diagram

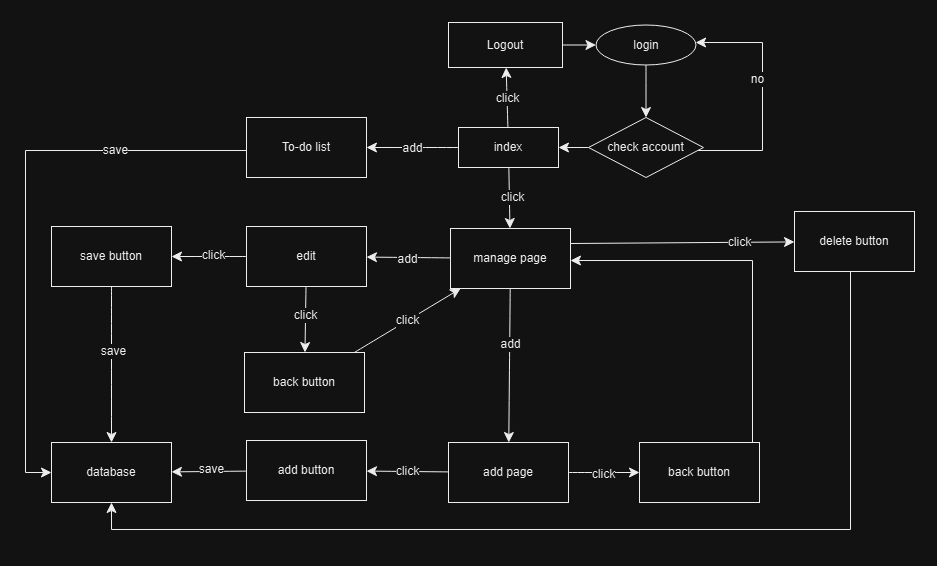


Figure 10. UI Diagram