FRANKFURT UNIVERSITY OF APPLIED SCEINCE

JAVA PROJECT REPORT

PET SHOP

Module: Programming Exercises

1. Nguyễn Thanh Sơn – 10422072
2. Nguyễn Thanh Tú – 10422080
3. Võ Tấn Sang – 10422114

Lecturer: Ralf-Oliver Mevius

Frankfurt am Main, SS2025

Table of Contents

Contents

Abbreviation

DMS	Database Management System
OOP	Object-Oriented Programming
HTTP	Hypertext Transfer Protocol
UI	User Interface
DAO	Data Access Object
DBMS	Database Management System
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
JDBC	Java Database Connectivity
JSP	JavaServer Pages
MVC	Model-View-Controller
SQL	Structured Query Language
IDE	Integrated Development Environment
CRUD	Create, Read, Update, Delete
JSTL	JSP Standard Tag Library
API	Application Programming Interface

LIST OF FIGURE

LIST OF TABLE

1. INTRODUCTION

The Pet Shop Website is a web-based application designed to streamline the operations of a typical pet store. It focuses on digitalizing inventory, customer interactions, and transaction management. With increasing demand for convenience, automation, and efficient resource handling in the retail pet industry, this system offers a centralized solution for managing pets, orders, users, and shopping carts through a structured interface backed by a relational database.

This website will offer the following core business functions:

- Adding, updating, and deleting pet listings
- Managing user accounts and roles
- Viewing and processing customer orders
- Shopping cart and checkout functionality
- User login authentication and access control

The project adheres to the Model-View-Controller (MVC) architectural pattern to separate concerns across different layers: interface (controllers), logic (services), and persistence (DAO and model classes). Each layer is implemented using Java and integrated with a MySQL database, ensuring data persistence and consistency.

By implementing common practices such as DAO (Data Access Object) pattern, interface-based service abstraction, servlet controllers, and session-based authentication, this application aims to provide a robust and scalable backend foundation suitable for future enhancements and integrations.

2. TECHNOLOGIES USED

2.1 Programming languages

- Java

Java was used as the primary programming language for implementing the application's backend, including servlets, services, DAOs, and model entities. It provides object-oriented features, robust memory management, and compatibility with the Java Servlet API and JDBC.

- HTML /CSS / JavaScript

  These standard web technologies were utilized for building the frontend interface. HTML defines the structure of web pages, CSS is used for styling and layout (with Bootstrap support), and JavaScript supports user interactions such as form handling and dynamic updates.

2.2 Backend Technologies

- Java Servlets (Servlet API)

  The core logic of the application is handled by servlet classes that extend HttpServlet and respond to HTTP GET and POST requests. Controllers such as AddController, LoginController, and CheckoutController manage the workflow between user actions and the backend services.

- JSP (JavaServer Pages)

  JSP was used to generate dynamic HTML content. It allows embedding Java code directly into HTML pages for rendering pet listings, shopping carts, and order summaries. Data from controllers is passed to JSP files via request attributes.

- Maven

  Maven was used for project configuration, dependency management, and build automation. It simplifies compiling the project, managing JARs, and running the application in a local server environment.

2.3 Database Technologies

- MySQL

MySQL served as the relational database management system. It stores persistent data such as pet records, user accounts, and orders. The schema is normalized and includes foreign key relationships for data integrity.

- JDBC (Java Database Connectivity)
  JDBC was used in the DAO layer to connect Java code to the MySQL database. SQL queries for CRUD operations were written manually and executed using PreparedStatement and ResultSet objects.

2.4 Frontend Frameworks and Utilities

- Bootstrap
  Bootstrap was integrated into JSP files to create a responsive and mobile-friendly user interface. It provides pre-built components like navigation bars, tables, and forms that enhance visual design.
- Apache Commons Libraries
  Utility libraries were optionally used for tasks such as string handling or collection manipulation, although core logic primarily relied on standard Java libraries.

2.5 Development and Testing Tools

- IDE: Visual Studio Code / IntelliJ IDEA
  Java code was written and tested using modern IDEs like Visual Studio Code or IntelliJ, which provide syntax support, project navigation, and debugging tools.
- Apache Tomcat
  Tomcat was used as a local servlet container during testing. It supports servlet deployment and allows the application to run in a simulated server environment.
- Git & Github
  Git was used for version control and source code tracking. The repository was hosted on GitHub for collaborative development and change management.

3. REQUIREMENT ANALYSIS

3.1 Functional Requirements

The system must support the following core functionalities:

For admin user:

- Login with username and password
- Add new pets to the catalog (name, price, type, breed, etc.)
- Edit existing pet information
- Delete pet records
- View a full list of available pets
- View all placed orders and their details
- Update order status
- Logout securely

For Customers:

- Browse available pets (no login required)
- Filter pets by type, breed, or price
- Add pets to a shopping cart
- Modify or remove items from the cart
- Proceed to checkout
- View personal order history
- View order details
- Register a new user account
- Login and logout

3.2 Non-Functional Requirements

- **Usability**: The system must be easy to navigate for both admins and customers, with a clear and responsive user interface.
- **Performance**: The application should return query results quickly and respond to user actions with minimal delay.
- **Scalability**: The system should support future integration of new features such as payment processing or delivery tracking.
- **Security**: User login credentials and session information must be protected using proper authentication and authorization mechanisms.
- **Maintainability**: The codebase should follow layered and modular design principles to simplify debugging, updates, and extensions.
- **Portability**: The system should be deployable on any Java-compatible servlet container (e.g., Tomcat).

3.3 Use Case Diagram

The following description outlines the main interactions between the system and its two primary user types: **Customer** and **Admin**.

Customer Use Cases (See in figure 1)

- View Products
- Add to Cart / Remove from Cart
- View Cart
- Register
- Login / Logout
- Checkout (Place Order)
- View Order History

- View Order Details

Admin Use Cases (See in figure 2)

- Login / Logout
- Add Pet
- Edit Pet
- Delete Pet
- View All Orders
- Edit Order Status

These actions are mapped to specific servlet controllers in the system, such as AddController, CartController, CheckoutController, and OrderController.

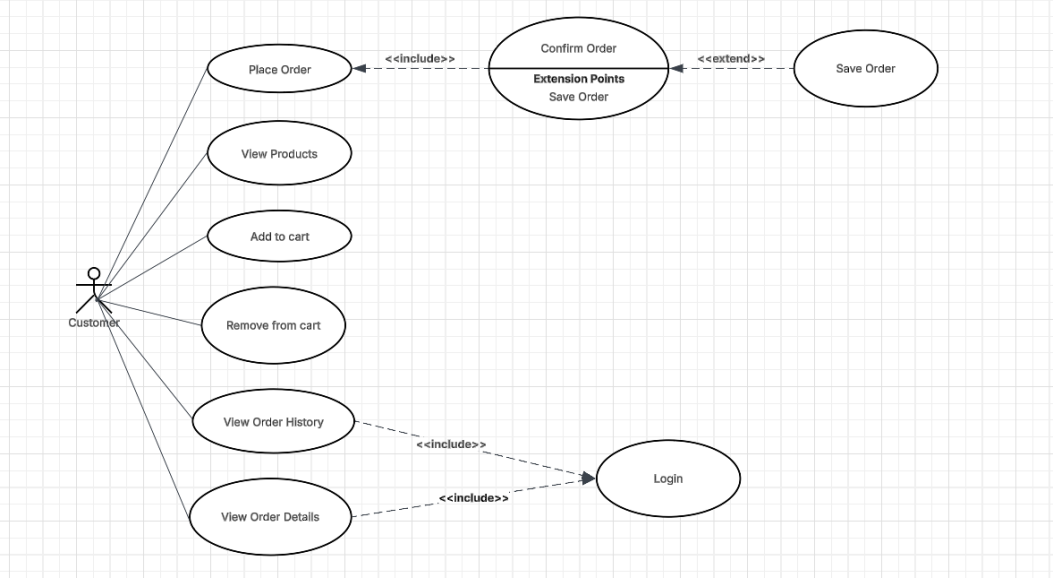


Figure 1. Use Case Diagram - Customer

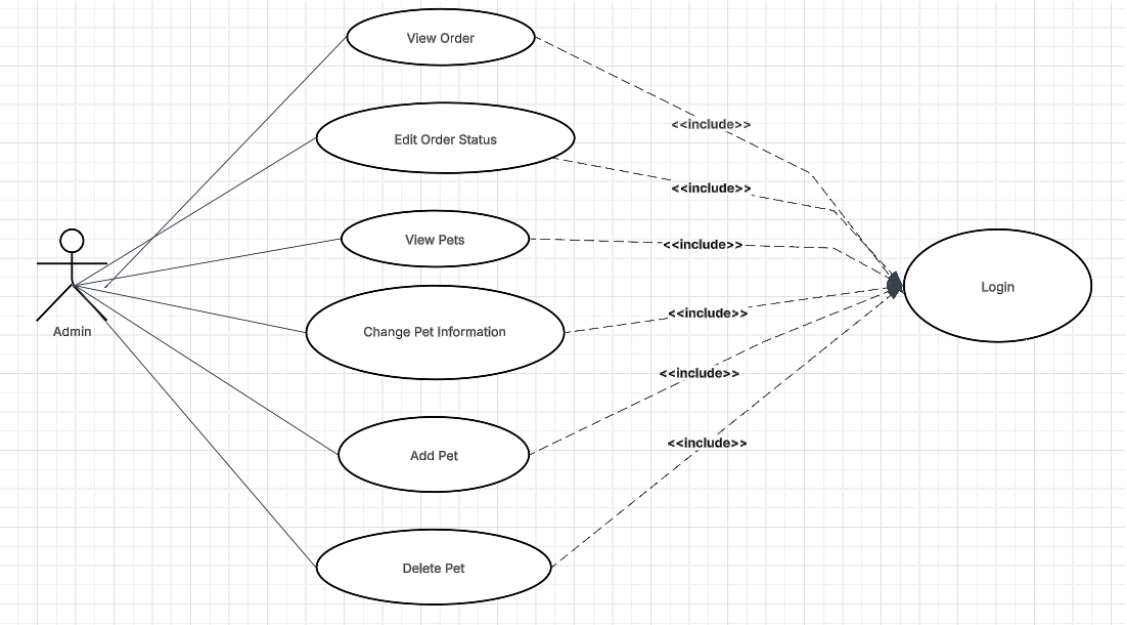


Figure 2. Use Case Diagram – Admin

4. SYSTEM ARCHITECTURE

4.1 MVC Design Pattern Overview

The MVC architecture separates the application into three core components:

- **Model**: Represents the data and business entities such as Pet, User, Order, and OrderDetail. These classes encapsulate fields and behaviors that correspond to database tables.
- **View**: Refers to the presentation layer built using JSP (JavaServer Pages). It renders the user interface (e.g., login page, pet listings, shopping cart) and displays data provided by the controller.
- **Controller**: Comprises servlet classes such as LoginController, AddController, CheckoutController, and others. These controllers receive user requests, process them

with the help of the service and DAO layers, and forward responses to the appropriate views.

4.2 Layered Architecture Breakdown

The internal structure of the backend is divided into multiple layers beyond MVC to improve reusability and clarity.

Controller Layer

- Handles HTTP requests and maps URLs to specific actions.
- Each controller is a servlet annotated with @WebServlet.
- Example: CheckoutController handles order placement and cart clearing logic.

Service Layer

- Contains business logic.
- Service interfaces such as iOrderService and iNewService define functionality contracts.
- Implementations like OrderService fetch and validate data before invoking DAO methods.

DAO Layer

- Manages database operations using SQL queries.
- DAO interfaces like iPetDAO and their implementations (PetDAO) abstract CRUD logic.
- Common behavior is centralized in AbstractDAO and GenericDAO.

Mapper Layer

- Converts rows from the database (ResultSet) into Java objects.
- Each mapper (e.g., PetMapper, UserMapper) implements the RowMapper<T> interface.

Model Layer

- Java classes like Pet, Users, and Order represent core domain objects.
- They include private fields, constructors, getters/setters, and extend a base class AbstractClass<T> for shared behavior.

Filter Layer

- Provides authentication control.
- The AuthFilter checks user session validity before allowing access to protected endpoints.

4.3 System Flow (Request Lifecycle)

The user initiates a request (e.g., browsing pets or placing an order).

The appropriate **Servlet Controller** intercepts the request.

The controller delegates logic to the **Service Layer**.

The service layer communicates with the **DAO Layer** for data retrieval or updates.

Mappers convert SQL results into Java model objects.

The controller sends the processed data to a **JSP View**, which renders the response in the browser.

4.4 System Architecture Diagram

This architecture ensures that each part of the Pet Shop Website is clearly defined, allowing for easier debugging, testing, and future enhancements. It also follows industry-standard web development patterns suitable for both academic and real-world deployment.

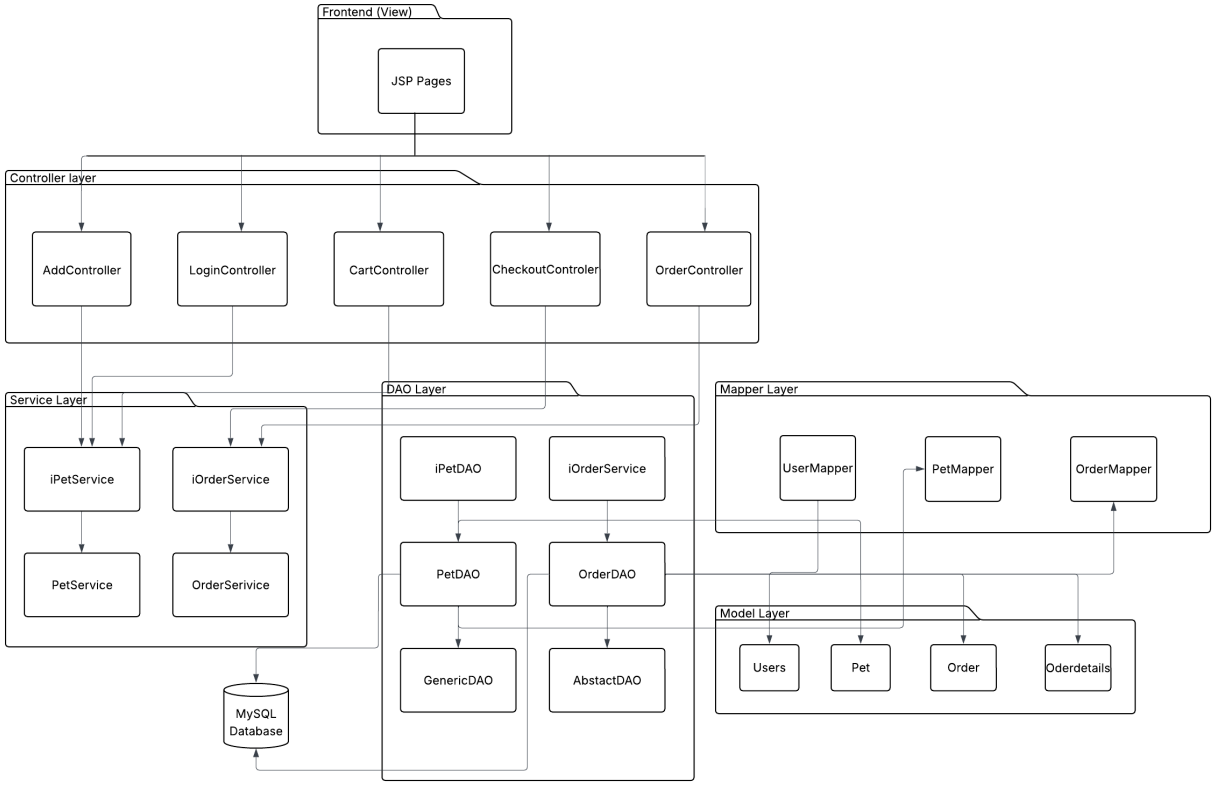


Figure 3. System Architecture Overview

5. MODEL LAYER

The Model Layer of the Pet Shop Website defines the core business entities used throughout the application. These entities represent real-world objects such as pets, users, orders, and order details. The model classes are implemented as JavaBeans (Plain Old Java Objects) and serve two main roles:

Representing data stored in the database.

Acting as data carriers between the DAO, service, and controller layers.

All model classes follow a consistent structure and are designed to be easily mapped from SQL query results using row mappers.

5.1. Entity Class Overview

The following classes make up the model layer:

5.1.1 Pet

Represents an animal listed on the website.

- **Fields**: id, name, price, type, breed, age, gender, description, addedBy
- **Purpose**: Used to display pet listings, filter pets by type or breed, and associate pets with orders or shopping carts.

5.1.2 Users

Represents a registered user of the website.

- **Fields**: id, username, email, password, role
- **Purpose**: Stores login credentials and identifies user roles (e.g., admin or staff). Session management relies on this class.

5.1.3 Order

Represents a customer purchase order.

- **Fields**: id, userId, totalPrice, createdDate
- **Purpose**: Links a user to the pets they purchased. Acts as the parent object in the order–order detail relationship.

5.1.4 OrderDetail

Represents an item in an order (a purchased pet).

- **Fields**: id, orderId, petId, quantity
- **Purpose**: Connects pets to orders and records individual quantities. Supports a normalized database structure.

5.1.4 CartItem

A temporary object used during the shopping process.

- **Fields**: petId, name, price, quantity, totalPrice

- **Purpose**: Stored in session memory to track customer selections before placing an order.

5.2 Use of a Generic Abstract Class

All model classes (except CartItem) extend a generic class called AbstractClass<T>, which provides shared attributes and methods:

```
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;
    }
}
```

This abstraction:

- Provides a consistent id property across all domain objects.
- Allows controller or service classes to reuse the listResult pattern for data packaging (especially in pagination or listings).
- Encourages DRY (Don't Repeat Yourself) principles in class design.

5.3 Mapping Between Models and Database Tables

Each model class corresponds directly to a table in the database:

Model Class	Corresponding Table
User	USERS
Pet	PETS
Order	ORDERS
OrderDetail	ORDER_DETAILS

Table 1. Mapping Between Model Classes and Database Tables

The fields in each class align with table columns, and the system uses row mappers (e.g., PetMapper, UserMapper) to convert between SQL result sets and Java objects.

5.4 Benefits of Model Layer Design

- Clear separation between data and logic.
- Easy integration with DAOs and services.
- Scalable — new fields or entities can be added with minimal changes to other layers.
- Ensures maintainable and testable code.

6. DATA ACCESS OBJECT LAYER

The Data Access Object (DAO) Layer in the Pet Shop Website is responsible for managing communication between the Java application and the MySQL database. This layer encapsulates all logic related to database operations such as retrieving, inserting, updating, and deleting data.

The DAO Layer follows a structured and reusable design pattern, where each business entity (such as Pet, Order, and Users) has its own DAO interface and implementation class. Additionally, shared logic is extracted into abstract base classes to reduce redundancy and improve maintainability.

6.1 DAO Interfaces and Implementations

For each entity, there is:

- A DAO interface (e.g., iPetDAO) that defines the operations.
- A corresponding implementation class (e.g., PetDAO) that executes SQL queries using JDBC.

Example:

- iPetDAO interface

```
public interface iPetDAO extends GenericDAO<Pet> {
    List<Pet> findAll();
    Pet findById(int id);
    void save(Pet pet);
    void update(Pet pet);
    void delete(int id);
    List<Pet> findByType(String type);
}
```

- PetDAO class

```
public class PetDAO extends AbstractDAO<Pet> implements iPetDAO {

    @Override
    public List<Pet> findAll() {
        String sql = "SELECT * FROM Pets ORDER BY created_at DESC";
        return query(sql, new PetMapper());
```

```
    }

    @Override
    public Pet findById(int id) {
        String sql = "SELECT * FROM Pets WHERE pet_id = ?";
        List<Pet> pets = query(sql, new PetMapper(), id);
        return pets.isEmpty() ? null : pets.get(0);
    }
```

6.2 Use of Generic and Abstract DAO

To avoid code duplication across different DAO classes, the system uses a base class hierarchy:

- **GenericDAO<T> -** a generic interface that defines common operations such as **query(), insert(), and update().**
- **AbstractDAO<T> -** An abstract class that implements common logic for executing SQL using JDBC. It provides a query() method that takes a SQL string and a RowMapper<T> to map the results.

Example: AbstractDAO core logic

```
@Override
    public <T> List<T> query(String sql, RowMapper<T> rowMapper, Object... parameters) {
        List<T> results = new ArrayList<>();
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        try {
            conn = connect();
            if (conn != null) {
                stmt = conn.prepareStatement(sql);
```

```
            // Set parameters
            if (parameters != null && parameters.length > 0) {
                for (int i = 0; i < parameters.length; i++) {
                    stmt.setObject(i + 1, parameters[i]);
                }
            }
            rs = stmt.executeQuery();
            while (rs.next()) {
                results.add(rowMapper.mapRow(rs));
            }
        }
    } catch (SQLException e) {
        System.out.println("Query error: " + e.getMessage());
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            System.out.println("Error closing resources: " + e.getMessage());
        }
    }
```

```
    return results;
}
```

This shared logic allows DAO implementations like PetDAO, UserDAO, and OrderDAO to focus purely on SQL queries and mapping logic, while avoiding duplication of low-level JDBC operations.

6.3 Integration with Mappers and Models

Every DAO method that returns data from the database relies on a corresponding mapper class to convert SQL ResultSet rows into Java model objects. These mappers implement the RowMapper<T> interface.

For example, the PetDAO class retrieves all pet records with the following line:

```
@Override
public List<Pet> findAll() {
    String sql = "SELECT * FROM Pets ORDER BY created_at DESC";
    return query(sql, new PetMapper());
}
```

Here:

- The SQL statement retrieves all pets from the database.
- PetMapper.mapRow() is invoked for each result row to create a Pet object.
- The final list of Pet objects is returned to the calling service or controller.

This design provides a clean separation between data access logic and object construction, and makes the system easier to debug and maintain.

6.4 DAO Classes in the Project

The project includes DAO interfaces and their corresponding implementations for each core entity:

DAO Interface	**Implementation**	Entity Managed

iPetDAO	PetDAO	Pet
iUserDAO	UserDAO	Users
iOderDAO	OrderDAO	Order and OrderDetail

Table 2. DAO Interfaces and Implementations

All DAO classes follow the same pattern and communicate directly with the database. This modular structure makes the application extensible, as new entities can easily be integrated by creating new DAO interfaces and implementations.

7. SERVICE LAYER

The Service Layer acts as the intermediary between the controller (presentation) layer and the DAO (data access) layer. It is responsible for handling the business logic of the application, validating inputs, orchestrating calls to multiple DAOs if necessary, and returning processed data to controllers.

By using interfaces and implementing classes, the Service Layer ensures loose coupling, testability, and the ability to easily swap or extend service logic without affecting other components.

7.1 Service Interface Design

Each entity has a corresponding service interface. These interfaces define the methods used by controllers and are implemented by service classes.

Example: iNewService.java

```
package com.laptrinhjavaweb.service;

import java.util.List;

import com.laptrinhjavaweb.model.Pet;
```

```
public interface iNewService {
    List<Pet> findAll();
```

Example: iOrderService.java

```
package com.laptrinhjavaweb.service;

import java.util.List;

import com.laptrinhjavaweb.model.Order;
import com.laptrinhjavaweb.model.OrderDetail;

public interface iOrderService {
    List<Order> getAllOrders();
}
```

These interfaces provide abstraction for operations such as retrieving all pets or orders, and saving an order.

7.2 Service Implementation

The service implementations contain actual logic and delegate database operations to DAO classes.

Example: NewService.java

```
package com.laptrinhjavaweb.service.impl;

import java.util.List;

import com.laptrinhjavaweb.dao.iPetDAO;
import com.laptrinhjavaweb.dao.impl.PetDAO;
import com.laptrinhjavaweb.model.Pet;
import com.laptrinhjavaweb.service.iNewService;
```

```
import jakarta.inject.Inject;

public class NewService implements iNewService {

    @Inject
    private iPetDAO petDAO;

    public NewService() {
        petDAO = new PetDAO();
    }

    @Override
    public List<Pet> findAll() {
        return petDAO.findAll();
    }
}
```

Example: OrderService.java

```
package com.laptrinhjavaweb.service.impl;

import java.util.List;

import com.laptrinhjavaweb.dao.iOrderDAO;
import com.laptrinhjavaweb.dao.impl.OrderDAO;
import com.laptrinhjavaweb.model.Order;
import com.laptrinhjavaweb.model.OrderDetail;
import com.laptrinhjavaweb.service.iOrderService;
```

```
import jakarta.inject.Inject;

public class OrderService implements iOrderService {

    @Inject
    private iOrderDAO orderDAO;

    public OrderService() {
        orderDAO = new OrderDAO();
    }

    @Override
    public List<Order> getAllOrders() {
        return orderDAO.getAllOrders();
    }
```

7.3 Usage in Controller Layer

Controllers use services to avoid direct dependency on DAOs. This improves maintainability and makes unit testing easier.

Example: HomeController.java

```
@Inject
    private iNewService petService;
@Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
// Get all pets for inventory
        List<Pet> pets = petService.findAll();
        request.setAttribute("pets", pets);
```

```
        RequestDispatcher rd = request.getRequestDispatcher("/views/admin/home.jsp");
    rd.forward(request, response);
}
```

7.4 Benefits of the Service Layer

- **Encapsulation of Business Logic**: Logic related to orders, pets, and users is centralized in services.
- **Testability**: Services can be mocked or tested independently of controllers and DAOs.
- **Loose Coupling**: Changing a DAO or adding new logic doesn't affect the controller structure.
- **Reusability**: Services can be used by multiple controllers or views without duplication.

8. AUTHENTICATION AND ACCESS CONTROL

The Pet Shop Website includes authentication and access control mechanisms to ensure secure interactions between users and the system. It implements session-based login, servlet filtering for protected endpoints, and role-based display control in JSP views. These mechanisms help restrict access to sensitive actions (such as modifying pet data or viewing orders) to authorized users only.

8.1 Login Authentication

Login functionality is implemented through the LoginController, which validates user credentials submitted from a login form. Upon successful authentication, the user object is stored in the HTTP session under the key "user". This session attribute is later checked across the system to determine access rights.

Code Snippet – LoginController.java

```
@Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // Retrieve username and password from form
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        try {
            // Validate user credentials
            Users user = userDAO.getUserByUsernameAndPassword(username, password);

            if (user != null) {
                // Credentials are valid: set user in session and redirect to dashboard
                HttpSession session = request.getSession();
                session.setAttribute("loggedUser", user);
                session.setAttribute("role", user.getRole()); // Ensure role is set for admin check
                session.setAttribute("username", user.getUsername()); // Ensure username is set for
header

                // Redirect based on user role
                if ("Admin".equalsIgnoreCase(user.getRole())) {
                    response.sendRedirect(request.getContextPath() + "/admin-home");
                } else {
                    response.sendRedirect(request.getContextPath() + "/shopping");
                }
            } else {
                // Invalid credentials: set error message and return to login page
                request.setAttribute("errorMessage", "Invalid username or password!");
                request.getRequestDispatcher("/views/web/login.jsp").forward(request, response);
```

```
        }
```

8.2 Access Filtering Using Servlet Filter

Protected endpoints such as /admin-home, /add, or /order-view are guarded using a servlet filter named AuthFilter. This filter intercepts HTTP requests and checks whether a valid user session exists. If not, the request is redirected to a login page.

Code Snippet – AuthFilter.java

```
public class AuthFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {}

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        HttpSession session = req.getSession(false);
        String uri = req.getRequestURI();

        // Check authentication
        if (session == null || session.getAttribute("username") == null) {
            res.sendRedirect(req.getContextPath() + "/login");
            return;
        }

        // Check authorization for admin paths
        if (isAdminPath && !"admin".equals(session.getAttribute("role"))) {
```

```
            res.sendRedirect(req.getContextPath() + "/shopping");
            return;
        }

        chain.doFilter(request, response);
    }
```

8.3 Role-Based Access in JSP Views

In addition to servlet-level access filtering, the system enforces **role-based display control** at the presentation layer. Admin-only interface elements—such as the button to add new pets—are shown conditionally based on the user's role.

The Pet Shop Website uses **JSTL** (<c:if>) to check user roles inside JSP pages. This ensures clean and readable view logic.

```
<c:if test="${user.role eq 'Admin'}">
    <a href="add" class="btn btn-primary btn-sm">Add New Pet</a>
</c:if>
```

Figure 4. Role-Based Display in home.jsp using JSTL

In this example:

- The user object is assumed to be stored in the session and exposed to the JSP.
- The "Add New Pet" button is only rendered if the logged-in user's role is "Admin".

This approach supports a consistent user experience while enforcing access control across both the backend and frontend layers.

8.4 Logout Mechanism

The logout functionality in the Pet Shop Website is implemented by invalidating the current HTTP session. This operation ensures that the user is securely logged out and cannot access protected pages without re-authentication.

Typically, this is triggered by a “Logout” button or link in the user interface.

Once the session is invalidated:

- The user attribute is removed.
- The AuthFilter will block further access to protected URLs.
- The user is redirected to the login page.

This mechanism ensures a clean and secure logout process.

8.5 Summary of Access Strategy

The Pet Shop Website combines servlet filters, session-based tracking, and conditional view rendering to protect sensitive parts of the application.

Component	Purpose
LoginController	Authenticates user credentials and starts session
AuthFilter	Protects restricted URLs by checking session
JSP <c:if> Conditions	Controls visibility of admin-only buttons in the UI
session.invalidate()	Clears session to log the user out

Table 3. Access Control Components

This layered access strategy ensures that unauthorized users are blocked from critical operations, and each role experiences a tailored user interface.

9. USER INTERFACE DESIGN

The Pet Shop Website provides a structured and user-friendly interface designed using **JSP (JavaServer Pages)** and styled with **Bootstrap**. The interface enables both customers and administrators to interact with the system's functionalities clearly and efficiently.

All views are rendered dynamically using data passed from servlet controllers via request attributes. The interface adapts based on login state and user role.

9.1 Technologies used for UI

- **JSP (JavaServer Pages)**: Used to generate HTML content dynamically from backend data.
- **HTML5 / CSS3**: Structure and styling of web pages.
- **Bootstrap**: Provides responsive layout and styled UI components.
- **JSTL (JSP Standard Tag Library)**: Used for logic control in JSP (e.g., <c:if>, <c:forEach>).
- **JavaScript (basic)**: Supports minor interactivity such as form submission or button actions.

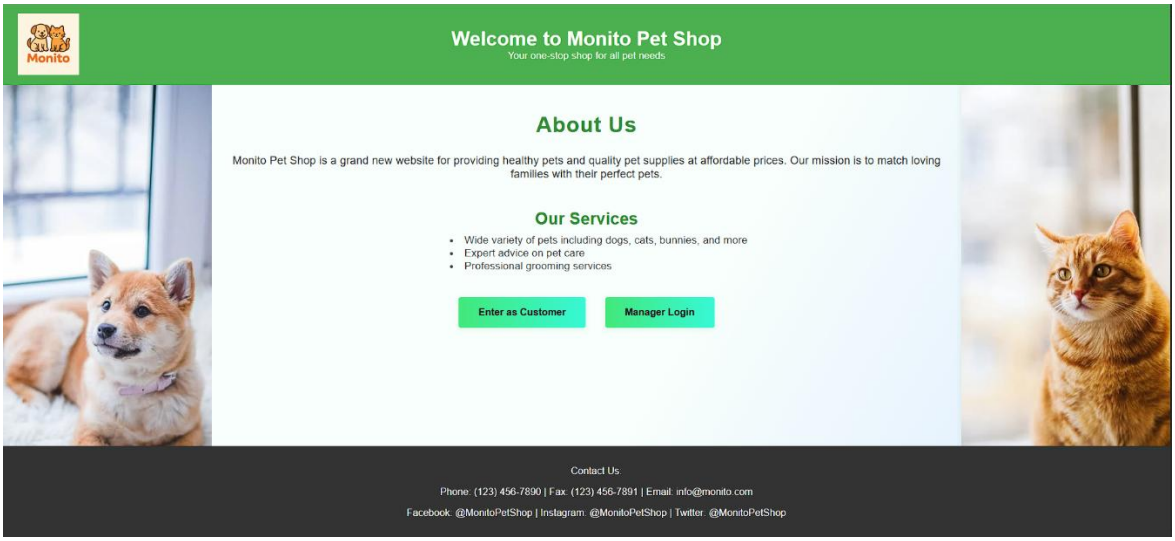


Figure 5. Welcome Page – Entry Point to Customer and Admin Interfaces

9.2 Key Interfaces and Screens

9.2.1 Admin Home

This page is shown to logged-in administrators and includes:

- A list of pets in inventory
- Buttons for editing or deleting pets

- A conditional button for adding new pets (admin-only)

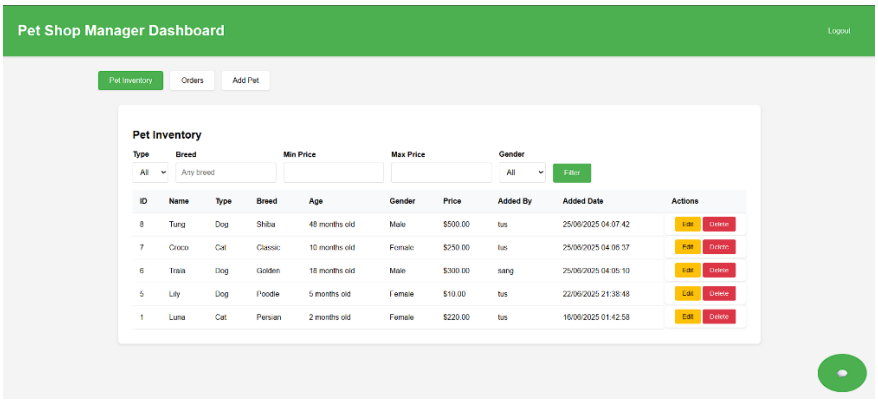


Figure 6. Admin Dashboard – Pet Inventory

9.2.2 Customer Shopping

This is the public-facing pet listing. It allows customers to:

- View available pets
- Add pets to cart
- Filter pets by type

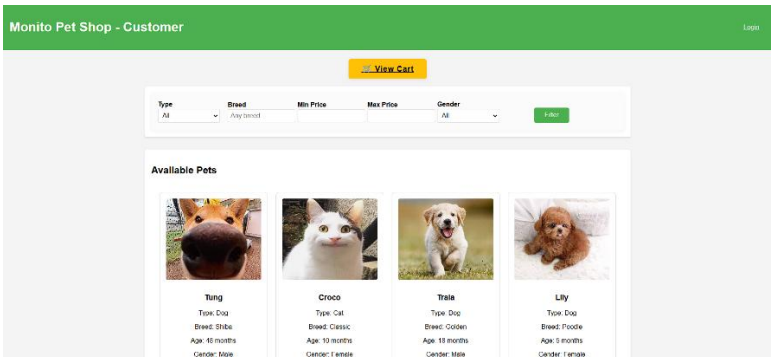


Figure 7. Customer Browsing View – Pet Listings and Filter Options

9.2.3 Cart Page

Displays the customer's current cart with options to:

- Update quantity

- Remove items
- Proceed to checkout

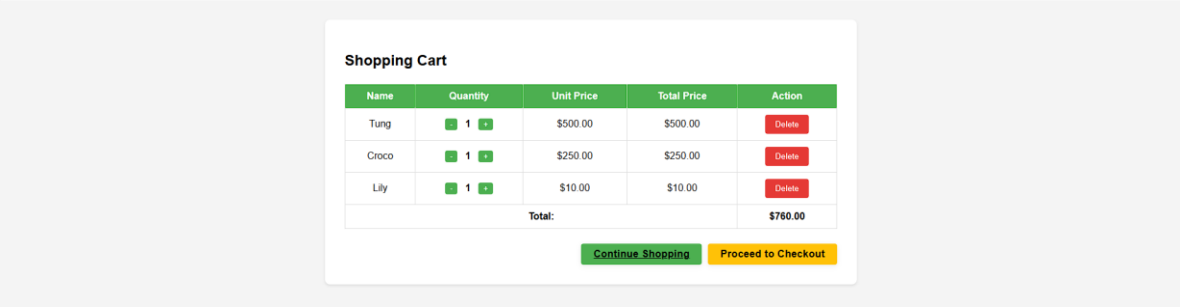


Figure 8. Shopping Cart – Selected Pets with Quantities and Price

9.2.4 Checkout Page

- checkout-auth.jsp redirects users to login if they try to check out while unauthenticated.
- checkout.jsp confirms the purchase and triggers order creation.

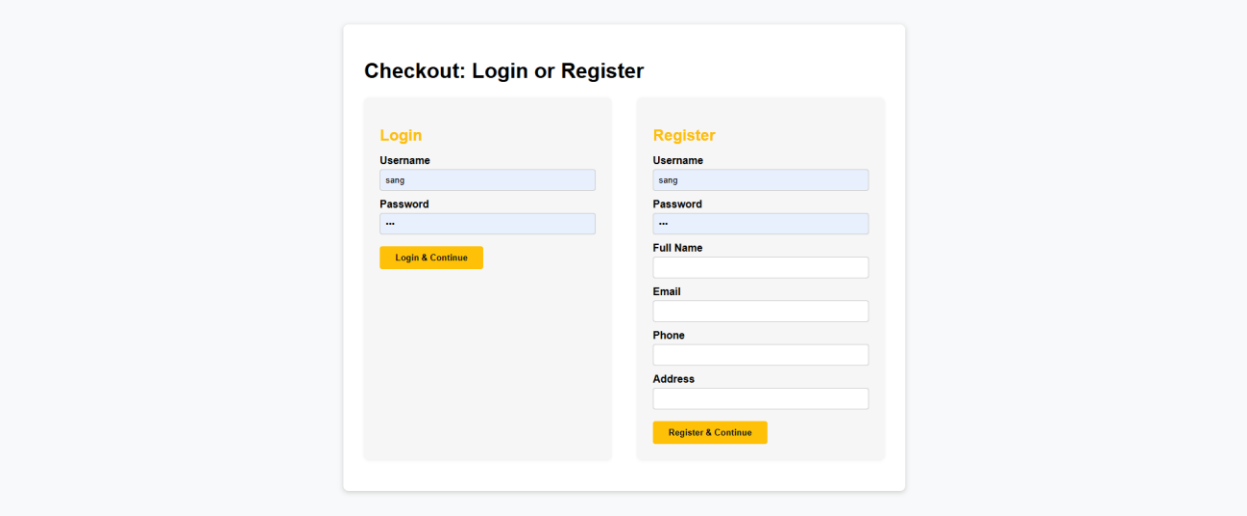


Figure 9. Checkout Login/Register Page

9.2.5 Order Confirmation

This view thanks the user for their purchase and confirms order submission.

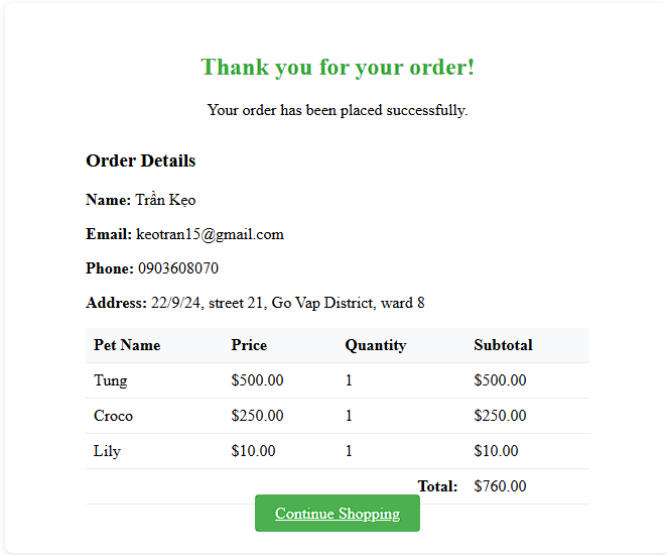


Figure 10. Order Confirmation

9.3 Reusable UI Components

The project includes a JSP fragment named taglib.jsp for reusable tag imports. It declares the JSTL core and formatting libraries:

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<%@ taglib prefix="fmt" uri="jakarta.tags.fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

This enables standardized display logic across pages without repeating boilerplate code.

9.4 UI Design Strengths

- **Consistency**: Layout and components are consistent across admin and customer views.
- **Responsiveness**: Bootstrap ensures the interface adjusts to various screen sizes.
- **Clarity**: Role-based display logic simplifies the interface by hiding irrelevant features.
- **Session Awareness**: The UI reacts to user state (logged in, logged out, admin role, etc.)

10. System Functionalities

The Pet Shop Website supports a wide range of functionalities for both administrators and customers. These functionalities are implemented using the MVC architecture, supported by servlet controllers, services, and DAO components. The interface dynamically responds to user role and session state to provide appropriate access.

This section outlines the major features of the system, grouped by user role.

10.1 Administrator Functionalities

Administrators are authenticated users with the "Admin" role. Upon login, they are redirected to the admin dashboard (home.jsp), where they can manage the pet inventory and view orders.

10.1.1 Login as Admin

- Access: LoginController.java
- Authenticates the admin via email and password
- Stores session attribute user
- Redirects to home.jsp

10.1.2 Add New Pet

- Access: AddController.java
- Admin fills a form with pet details (name, type, breed, price, etc.)
- Data is validated and saved via PetDAO
- Confirmation redirects back to the admin home

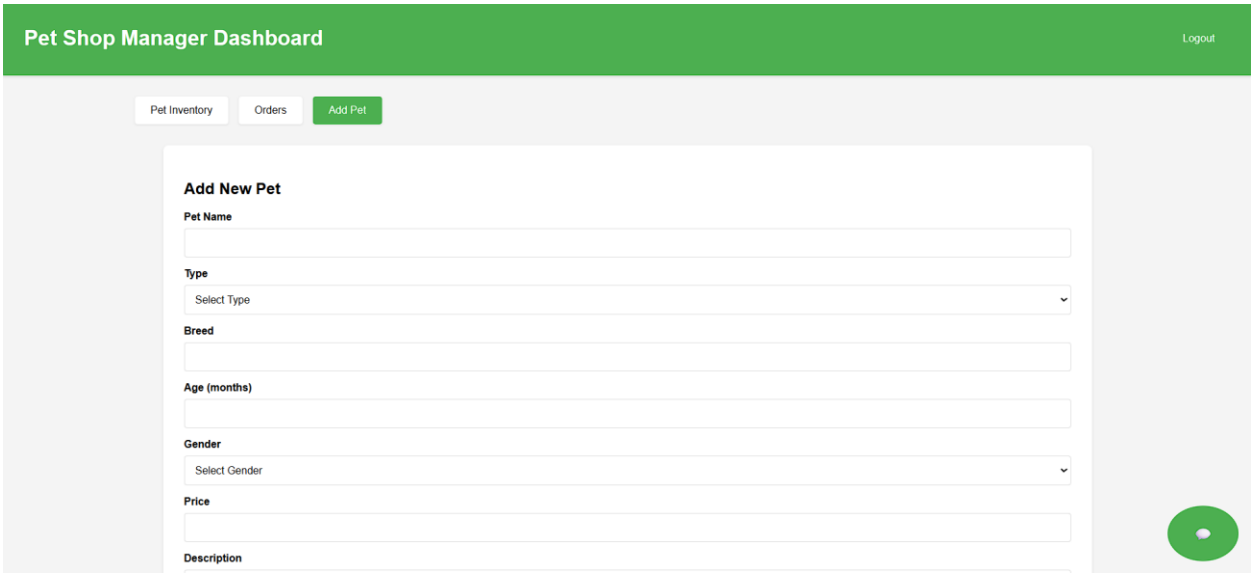


Figure 11. Add New Pet Form (1)

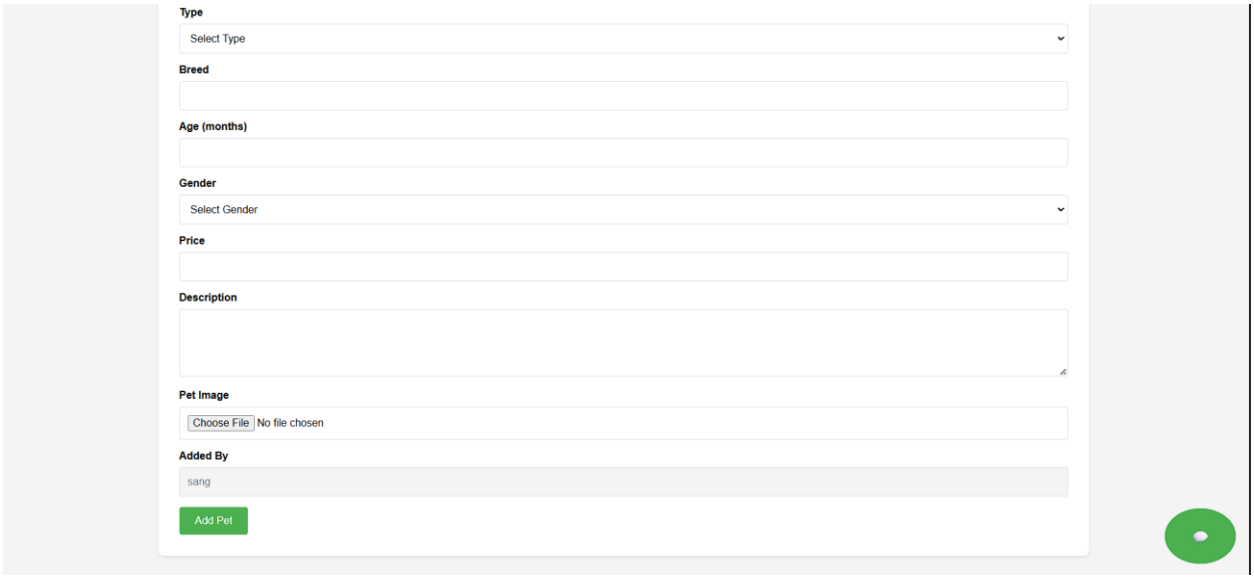


Figure 12. Add New Pet Form (2)

10.1.3 Edit Pet Information

- Access: EditController.java
- Admin clicks an “Edit” button beside a pet
- Pre-filled form loads pet data using findById()

- Submitting updates the record in the database

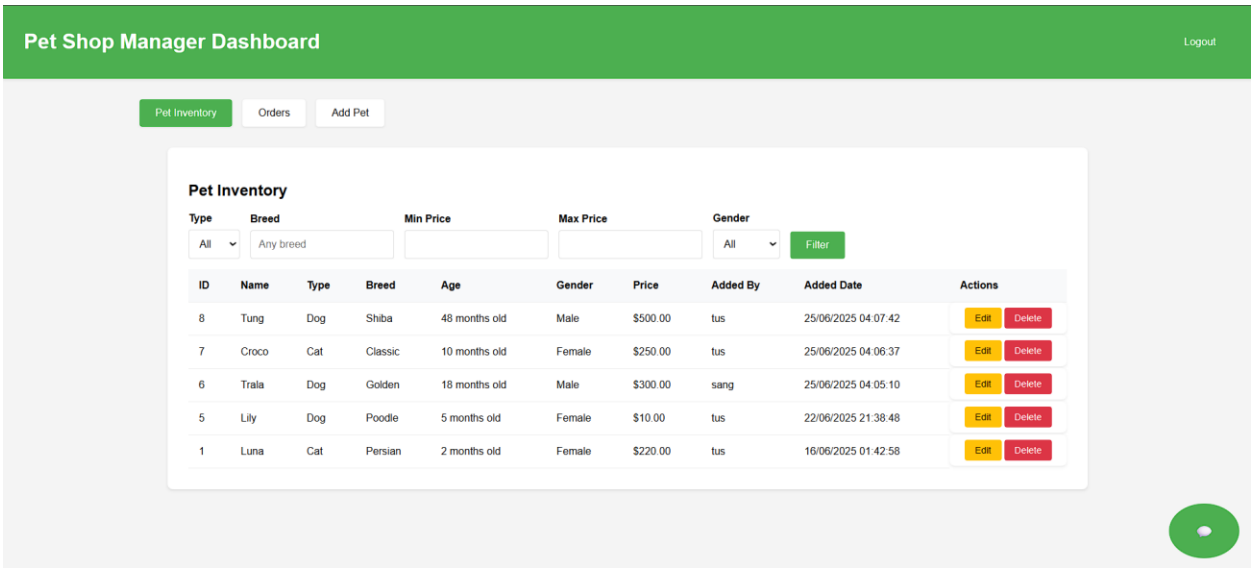


Figure 13. View and Manage Pet Information

10.1.4 Delete Pet

- Access: DeleteController.java
- Admin selects a pet to delete
- Deletes the record from the PETS table via DAO

10.1.5 View and Manage Orders

- Access: OrderController.java
- Displays all orders placed by users
- Admins can view order details and statuses

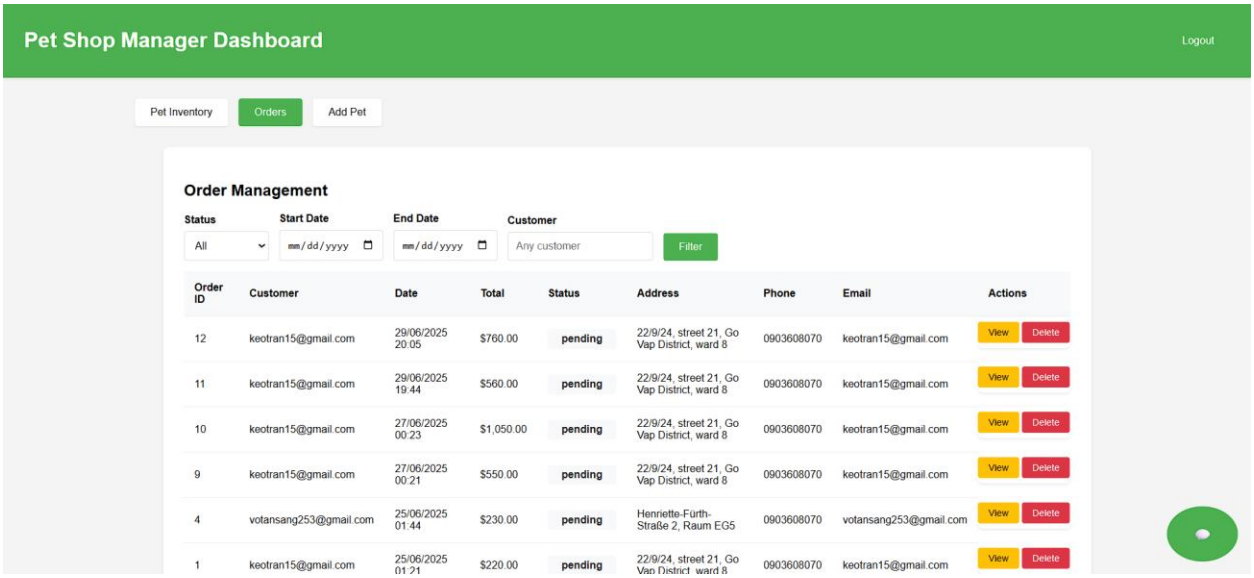


Figure 14. View and Manage Order

10.2 Customer Functionalities

Customers can browse pets without logging in but must authenticate to proceed with checkout and view order history.

10.2.1 Browse Pets

- Page: shopping.jsp
- Shows pet cards with image, name, type, and price
- “Add to Cart” form allows selection by guest or logged-in user

10.2.2 Add to Cart

- Access: CartController.java
- Adds selected pet to a session-based cart (List<CartItem>)
- User can update quantity or remove pets

10.2.3 View Cart

- Page: cart.jsp
- Lists current cart items

- Allows updating quantity or removing pets

10.2.4 Login / Register

- Handled via LoginController and (optionally) a registration controller
- Redirects back to the shopping/cart/checkout flow after login

10.2.5 Checkout

- Access: CheckoutController.java
- Converts CartItem objects into Order and OrderDetail
- Saves via OrderDAO and clears the cart

10.2.6 View Order History

- Access: OrderController.java (user-side)
- Authenticated users can view their previous orders and details

11. Conclusion

The Pet Shop Website successfully demonstrates the design and implementation of a full-stack, database-driven web application using Java technologies. Through the use of the Model-View-Controller (MVC) architecture, the project achieves a clean separation of concerns between the user interface, business logic, and data access layers.

The system provides key functionalities required in a retail pet store environment, including pet inventory management, user authentication, shopping cart processing, and order management. These features were implemented through a modular design involving servlet-based controllers, interface-driven services, JDBC DAOs, and a normalized MySQL database.

Security and access control were handled effectively using session-based authentication, servlet filters, and role-based rendering in the JSP interface. Administrators and regular users are provided with distinct functionalities and views, enhancing both usability and data protection.

The integration of JSP, JSTL, Bootstrap, and Java EE components resulted in a responsive and user-friendly interface, while backend components ensured data consistency, transaction safety, and future extensibility.

11.1 Achievements

- ✅ Built a functional Java web application with login, cart, and order management
- ✅ Used design patterns such as DAO, service abstraction, and MVC effectively
- ✅ Integrated a secure authentication mechanism with filtered access
- ✅ Applied relational database principles in table design and query logic
- ✅ Developed a responsive and role-aware user interface

11.2 Opportunities for Future Development

While the project fulfills its core objectives, there is potential for future improvements, such as:

- Adding support for image uploads and pet photos
- Integrating payment gateway APIs for real-time checkout
- Implementing email notifications for order confirmations
- Providing an admin dashboard with data analytics and charts
- Making the system fully mobile-responsive with adaptive layouts

11.3 Final Remarks

Overall, the Pet Shop Website offers a strong technical foundation for a real-world inventory and transaction management system. The project has not only met its functional and technical goals but has also provided valuable experience in applying Java web development, MVC architecture, and database integration in a cohesive software solution.

APPENDIX: DATABASE

A.1 Entity Relationship Diagram

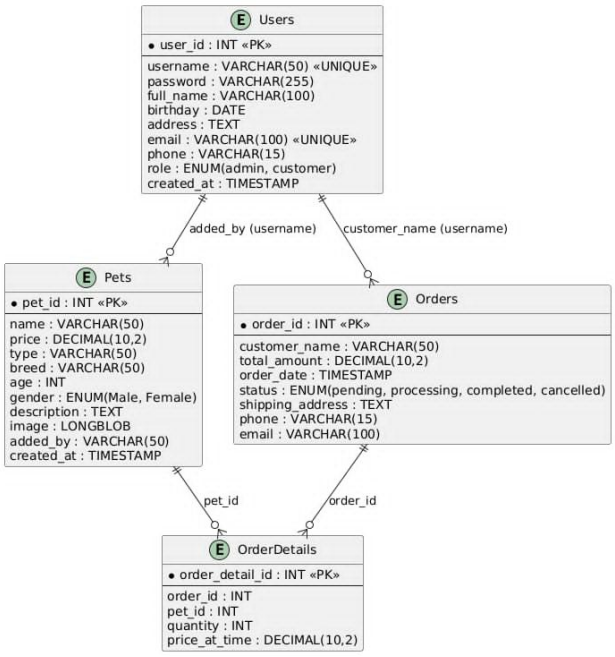


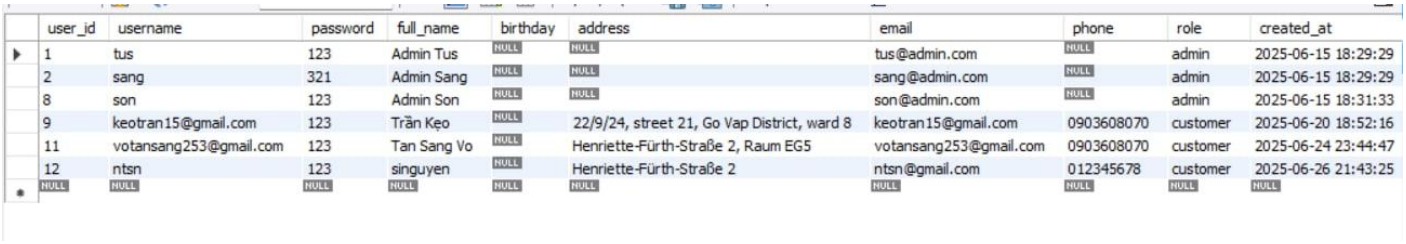
Figure 15. Entity Relationship Diagram of Pet Shop Database

A.2 Table Structures and Contents

a. USERS

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

Table 4. Table of User Database



user_id	username	password	full_name	birthday	address	email	phone	role	created_at
1	tus	123	Admin Tus	NULL	NULL	tus@admin.com	NULL	admin	2025-06-15 18:29:29
2	sang	321	Admin Sang	NULL	NULL	sang@admin.com	NULL	admin	2025-06-15 18:29:29
8	son	123	Admin Son	NULL	NULL	son@admin.com	NULL	admin	2025-06-15 18:31:33
9	keotran15@gmail.com	123	Trần Kẹo	NULL	22/9/24, street 21, Go Vap District, ward 8	keotran15@gmail.com	0903608070	customer	2025-06-20 18:52:16
11	votansang253@gmail.com	123	Tan Sang Vo	NULL	Henriette-Fürth-Straße 2, Raum EG5	votansang253@gmail.com	0903608070	customer	2025-06-24 23:44:47
12	ntsn	123	singuyen	NULL	Henriette-Fürth-Straße 2	ntsn@gmail.com	012345678	customer	2025-06-26 21:43:25
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 16. Users' Data

b. PETS

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 5. Table of Pet Information

pet_id	name	price	type	breed	age	gender	description	image	added_by	created_at
1	Luna	220.00	Cat	Persian	2	Female	Calm and fluffy.	BLOB	tus	2025-06-15 23:42:58
5	Lily	10.00	Dog	Poodle	5	Female	Curly hair with cute eyes	BLOB	tus	2025-06-22 19:38:48
6	Trala	300.00	Dog	Golden	18	Male	Friendly and Supportive	BLOB	sang	2025-06-25 02:05:10
7	Croco	250.00	Cat	Classic	10	Female	Sassy and talkative	BLOB	tus	2025-06-25 02:06:37
8	Tung	500.00	Dog	Shiba	48	Male	Silly and Freaky	BLOB	tus	2025-06-25 02:07:42
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 17. PETS Table Content

c. ORDERS

Column Name	Data Type	Description
id	INT (PK)	Order ID
userId	INT (FK)	Reference to USERS
totalPrice	DECIMAL(10,2)	Total order value
createdDate	DATETIME	Timestamp

Table 6. Table of Order Information

d. ORDER_DETAILS

Column Name	Data Type	Description
order_detail_id	INT (PK)	Order Detail ID
orderId	INT (FK)	Reference to ORDERS
petId	INT (FK)	Reference to PETS
quantity	INT	Number of pets ordered
price_at_time	DECIMAL(10, 2)	Price of pet

Table 7. Table of ORDER_DETAILS

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*. Springer.
5.