Faculty of Engineering
Frankfurt University of Applied Sciences

Programming Exercise

PETSHOP

Business Management System

Lecturer	Mevius Ralf-Oliver
Contributing Students:	Student Numbers:
Tran Ngoc Vu	1615687
Nguyen Thao Vy	1616134
Le Thai Ba Quan	1618086
Nguyen Gia Thong	1618732

Date: June, 2025

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Idea of the PE's project

The Pet Shop Management System is a comprehensive **desktop application** designed to streamline and modernize the operations of pet retail businesses. This robust `Java-based` solution provides pet shop owners with **an integrated platform** to efficiently manage their entire **business workflow**, from inventory control to customer relationships and financial reporting.

Comprehensive Business Management:

The system offers a complete suite of **management tools** including **product inventory management** (`FOOD`, `MEDICINE`, `TOY`), **live pet sales tracking**, **customer database** with loyalty point systems, and **automated billing** with `PDF invoice generation`. Pet shop owners can seamlessly handle all aspects of their business operations through an intuitive **graphical interface**.

Role-Based Access Control:

The platform implements a secure authentication system with distinct user roles for `Managers` and `Staff` members. This ensures appropriate **access levels** and maintains **operational security** while allowing efficient delegation of tasks across different levels of responsibility within the organization.

Advanced Inventory and Sales Features:

The system provides **real-time stock monitoring** with **automatic status updates**, **out-of-stock notifications** for restocking purposes and comprehensive sales tracking. The billing module includes `shopping cart` functionality with **stock validation**, ensuring accurate transactions and preventing over-selling scenarios.

Technology and Architecture:

Built using `Java Swing` for the **user interface** and `MySQL` for robust data management, the application follows ***Model-View-Controller (MVC)*** architecture principles. This design approach ensures **maintenance, scalability, and separation** of concerns, while technologies such as `iText PDF` integration enable professional invoice generation and `JFreeChart` supports advanced reporting capabilities.

Business Intelligence and Reporting:

The integrated reporting dashboard provides **real-time insights** into **total revenue, sales volume, and transaction history**, enabling data-driven decision-making for business growth and operational optimization.

1.2 Scope

The scope of this project encompasses **the design** and **the implementation** of an integrated **pet shop management system** that serves pet shop owners, staff members and supports customer management operations. The system provides functions such as:

- Comprehensive pet inventory management
- Product catalog maintenance
- Customer relationship tracking
- Sales transaction processing
- Billing operations

Users can manage different types of pets (`DOG`, `CAT)`, various product categories (`FOOD`, `MEDICINE`, `TOY`), and generate detailed **business reports**.
The system includes **advanced statistical analysis** and **data observation capabilities**, allowing users to:

- Monitor sales trends
- Track inventory turnover rates
- Analyze customer purchasing patterns
- Evaluate product performance metrics

The reporting module provides **comprehensive information** on business operations through **graphical representations, summary statistics, and customizable data filters** for informed decision making. The system includes role-based access control for different types of user (`MANAGER, STAFF`), secure **authentication mechanisms**, and intuitive user interfaces for **efficient daily operations**. The project is designed to be **scalable, maintainable, and secure** to protect critical business information.
The current scope does **not include** online customer booking systems or e-commerce capabilities but provides a foundation for **future expansion** into digital customer services.

1.3 Features

User Authentication and Role-Based Access Control:

The system provides **secure login and sign-up** functionality for different user roles including `MANAGER` and `STAFF`. Role-based access control ensures that users can only access features appropriate to their authorization level, maintaining **operational security** and **proper task delegation** within the pet shop organization.

Comprehensive Product Management:

Shop administrators can efficiently manage their entire product inventory including `TOY, FOOD, MEDICINE` for pets. The system provides full ***CRUD (Create, Read, Update, Delete)*** operations with **dynamic stock control** and **automatic status handling**.

Live Pet Sales Management:

The platform enables management of `PET` available for sale, including `CAT` and `DOG` with detailed breed information. `PETs` are treated as **one-time purchasable items**.

Customer Relationship Management with Loyalty System:

Owners can maintain a detailed customer database with a built-in **loyalty point system**. Customers automatically earn **1 point per $10 spent**.

Advanced Billing and Invoice System:

Includes **shopping cart functionality** with real-time stock validation and automatic `PDF invoice generation` using `iText`.

Inventory Control and Stock Management:

Supports **real-time inventory updates**, **out-of-stock alerts**, and **stock validation** during checkout to prevent overselling.

Business Intelligence and Reporting Dashboard:

Managers have access to real-time `revenue insights, transaction tracking`, and advanced `sales analytics`.

Database Integration and Data Persistence:

Built with `MySQL` **database integration**, secure data persistence using `JDBC`, and role-based access to stored data.

Tech Stack:

- **Front-end:** `Java Swing GUI Framework, Custom UI Components`
- **Back-end:** `Java 22, Maven, MVC Architecture`
- **Database:** `MySQL, JDBC`
- **PDF Generation:** `iText PDF Library (5.5.13.3)`

- **Data Visualization:** `JFreeChart (1.5.5)`
- **Security:** `BCrypt Password Hashing`
- **Design Patterns:** `Factory Pattern, DAO Pattern, Service Layer`
- **Dev Tools:** `Maven, Git, JUnit 4.13.2`

1.4 User Stories

1.4.1 Sarah's Story (Pet Shop Manager)

Sarah, the manager of "Happy Paws Pet Shop", logs in and uses the dashboard to check stock levels. She updates dog food stock, adds cat toys, and reviews weekly revenue ($2,847 from 23 transactions). She creates a staff account for Emma with appropriate permissions.

1.4.2 David's Story (Pet Manager)

David manages live pets. When a family visits to adopt a cat, he shows three available cats with price and health info. They adopt **Milo** ($150 + $45 care package). **David** updates **Milo**'s status as **"sold"**, and the system updates records.

1.5 Brainstorm Idea

System Components

- `User Stories`
- `ERD (Entity Relationship Diagram)`
- `Use-case Diagram`
- `Sequence Diagram`
- `UML Class Diagram`

Story Overview

- Pet shop owners need a centralized system to manage operations.
- Managers monitor staff and business performance through reporting.
- Staff handle sales, inventory, and adoptions.
- Loyalty programs help maintain customer engagement.

Use Cases

- Manager **logs in** to access dashboard
- **Search/filter** products by stock and category
- Staff **processes** product or pet transactions
- **View** customer profiles and loyalty points
- **Add/update** products and stock
- **Generate** invoices and reports

Functional Requirements

- The system should **display** current inventory status of all products and pets
- The system should have **filter** functionality (product types, stock status, price range)
- The system should **enable** transaction processing with shopping cart and checkout
- The system should **maintain** customer databases with loyalty tracking
- The system should **generate** `PDF invoices` automatically
- `STAFF` should be able to **update** inventory levels and product information
- `MANAGER` should be able to **access** comprehensive reports on sales and revenue
- The system should **handle** both product sales and pet adoptions seamlessly
- `STAFF` can **process** returns and exchanges when customers are unsatisfied

- The system should **provide** real-time stock alerts for inventory management

Minimum Viable Product (MVP)

- User authentication
- Product and customer management
- Shopping cart and checkout
- `PDF invoice` generation
- Inventory and pet management
- Basic reporting

Description for the main features

- Display current inventory status with real-time stock levels
- Display customer information including loyalty points and purchase history
- Process transactions efficiently with automatic stock validation and total calculation
- Generate professional PDF invoices for each sale with detailed itemization
- Manage multiple product categories (`TOY, FOOD, MEDICINE`) with specific attributes
- Handle live pet sales with detailed pet profiles and adoption tracking

Key Business Processes

```
Dashboard → Inventory overview → Staff selects section (Products, Pets, Customers)
→ Manages records, checks stock, performs transactions
```

Vocabulary

- **Users:** Pet shop **manager, staff, administrators**
- **Platform:** Centralized system for business operations
- **Goals:** Efficiency, automation, customer retention

Chapter 2

System Architecture

The **Pet Shop Management System** follows a layered ***Model-View-Controller (MVC)*** architecture pattern that ensures separation of concerns, maintainability, and scalability. The system is divided into **six main layers** as illustrated below.

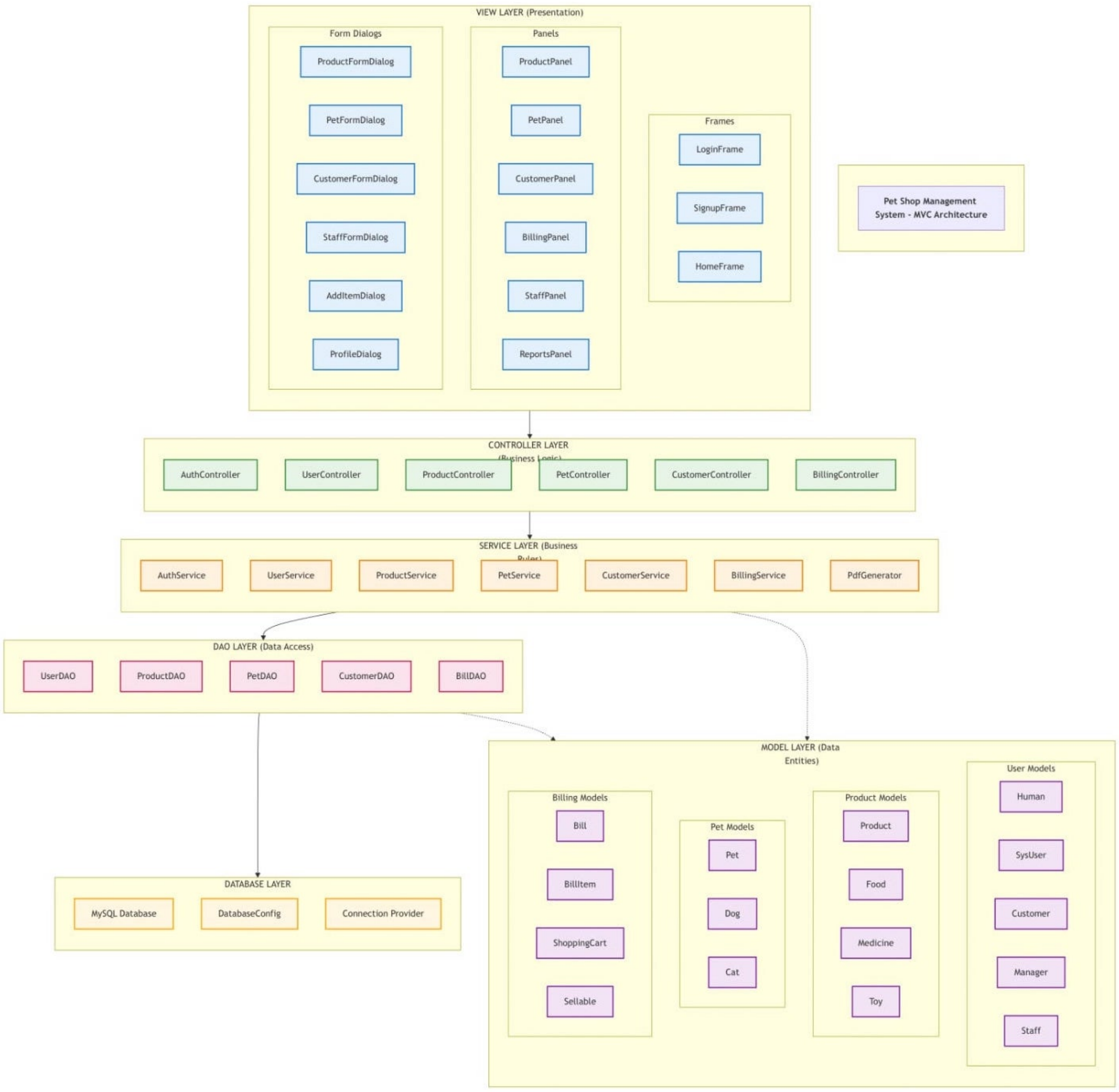


Figure 2.1: The MVC Architecture

2.1 View Layer (Presentation)

This layer handles all **user interface components** and **user interactions**, organized into three main categories:

2.1.1 Frames

- `LoginFrame`: User authentication interface for system access
- `SignupFrame`: User registration interface for new accounts
- `HomeFrame`: Main application window containing navigation and panels

2.1.2 Panels

- `ProductPanel`: Interface for product management (`TOY, FOOD, MEDICINE`)
- `PetPanel`: Interface for pet inventory management
- `CustomerPanel`: Interface for customer profile management
- `BillingPanel`: Interface for transaction processing and billing
- `StaffPanel`: Interface for staff account management
- `ReportsPanel`: Interface for business analytics and reporting

2.1.3 Form Dialogs

- `ProductFormDialog`: Add/Edit product information
- `PetFormDialog`: Add/Edit pet information
- `CustomerFormDialog`: Add/Edit customer profiles
- `StaffFormDialog`: Add/Edit staff accounts
- `AddItemDialog`: Add items to shopping cart
- `ProfileDialog`: User profile management

2.2 Controller Layer (Business Logic)

This layer handles **user requests** and **coordinates** between the View and Service layers:

- `AuthController`: Manages authentication and authorization operations

- `UserController`: Handles user profile and account management
- `ProductController`: Coordinates product-related operations
- `PetController`: Manages pet inventory operations
- `CustomerController`: Handles customer relationship management
- `BillingController`: Processes billing and transaction operations

2.3 Service Layer (Business Rules)

This layer contains core **business logic** and **validation rules**:

- `AuthService`: Authentication logic, password hashing, session management
- `UserService`: User account validation and profile management
- `ProductService`: Inventory management, stock validation, product categorization
- `PetService`: Pet health validation, breed management, adoption processes
- `CustomerService`: Loyalty point calculations, customer relationship management
- `BillingService`: Transaction calculations, discount application, financial processing
- `PdfGenerator`: Invoice generation and document management

2.4 DAO Layer (Data Access)

This layer handles all **database operations** and **data persistence**:

- `UserDAO`: User account data access operations
- `ProductDAO`: Product inventory data operations
- `PetDAO`: Pet information data management
- `CustomerDAO`: Customer profile data operations
- `BillDAO`: Transaction and billing data management

Transaction Management:

- **Connection Pooling**: Efficient database connection management
- **Transaction Control**: `Commit/Rollback` operations for data integrity
- **Query Optimization**: Optimized `SQL queries` for performance

2.5 Model Layer (Data Entities)

This layer defines the **data structure** and **business entities**:

2.5.1 User Models

- `Human`: `Abstract base class` for all human entities
- `SysUser`: System user with authentication credentials
- `Customer`: Customer with loyalty points and purchase history
- `Manager`: System administrator with full access rights
- `Staff`: Employee with limited access permissions

2.5.2 Product Models

- `Product`: `Abstract base class` for all sellable products
- `FOOD`: Pet food with expiration dates and nutritional information
- `MEDICINE`: Pet medicine with dosage and health specifications
- `TOY`: Pet toys with material and safety information

2.5.3 Pet Models

- `Pet`: `Abstract base class` for all pets
- `DOG`: Dog-specific information and characteristics
- `CAT`: Cat-specific information and characteristics

2.5.4 Billing Models

- `Bill`: Transaction record with customer and payment information
- `BillItem`: Individual items within each transaction
- `ShoppingCart`: Temporary cart for managing purchases
- `Sellable`: Interface for all sellable items (products and pets)

2.6 Database Layer

This layer represents **the physical data storage**:

- `MySQL Database`: Main database containing all business data
- `DatabaseConfig`: Database configuration and connection settings
- `Connection Provider`: Database connection management and pooling

Database Tables

- `Users`: Manager and staff account information
- `Products`: Product inventory with stock levels and specifications
- `Pets`: Live pet inventory with health and breed information
- `Customers`: Customer profiles with loyalty points
- `Bills`: Transaction records with payment details
- `Bill_Items`: Itemized transaction details

2.7 Data Flow Architecture

The system follows a strict ***MVC data flow*** pattern:

`View` → `Controller` → **Service** → `DAO` → `Database`

Chapter 3

Feature Analysis

3.1 Overview

The system supports secure login functionality for two user roles: **Staff** and **Manager**. Users must authenticate using their email and password to access the system. Upon successful authentication, the system applies authorization logic to control access based on user roles.

- **Staff users can:**
  - View and interact with major data tables such as `Customer`, `Product`, `Pet`, and `Bill`.
  - Cannot view or modify other staff members' information — only their own profile.
- **Managers can:**
  - Access the entire system without restrictions.
  - View and manage all entities including `Staff`, `Customer`, `Product`, `Pet`, `Bills`, and generate reports.
  - Manage user roles and permissions for Staff.

3.2 Use Case Diagram: Staff and Manager in Pet Shop Management System

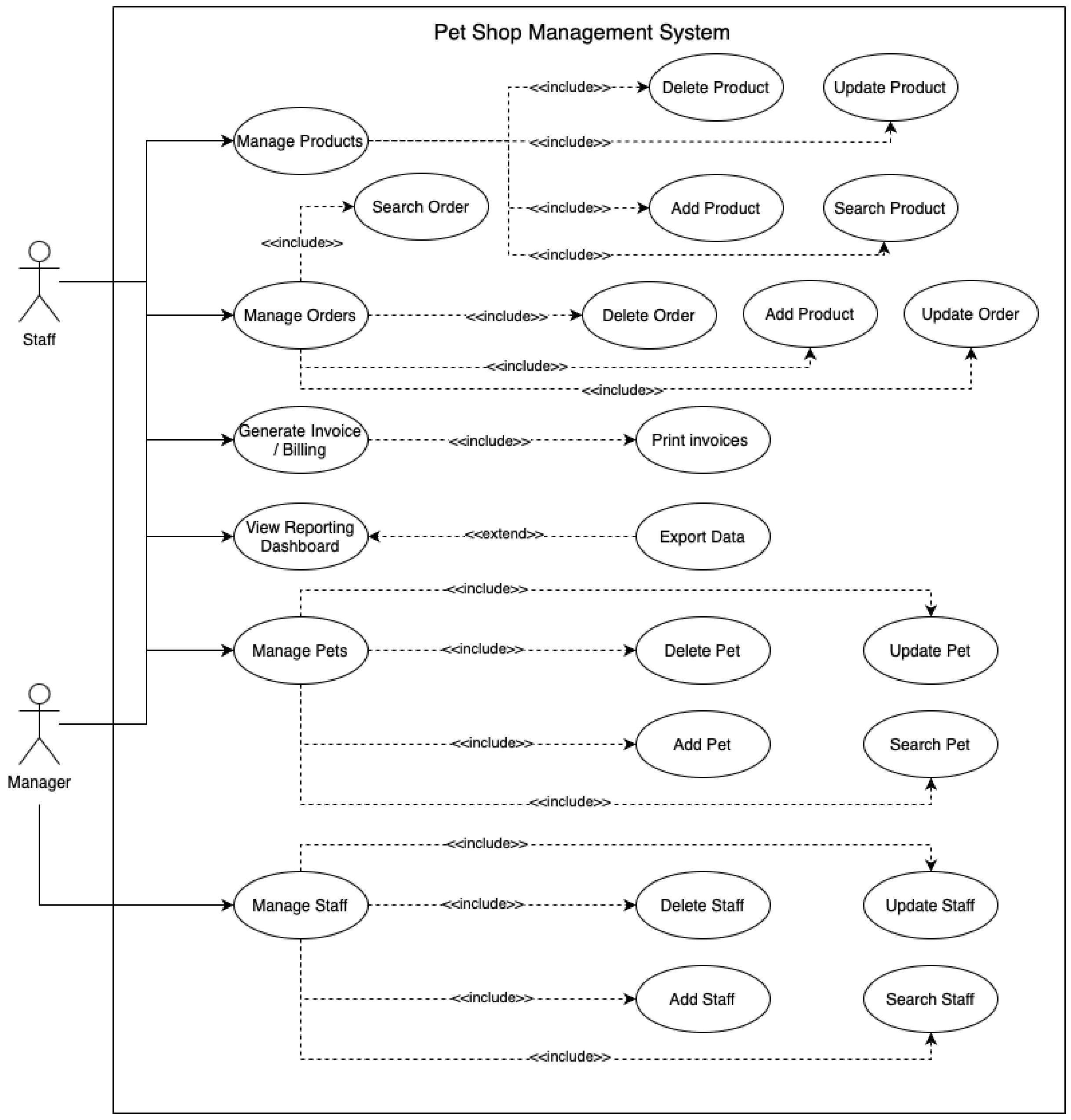


Figure 3.1: Use case diagram of Pet Shop Management System (PSMS)

3.3 Authentication & Authorization

3.3.1 Authentication – User Login

The system uses the `LoginFrame` class to display a modern and user-friendly log-in interface. Users (only Staff and Manager) enter their `Email` and `Password` to authenticate their account.

- Email and password fields are retrieved from `JTextField` and `JPasswordField`.

```
String email = emailField.getText().trim();

String password =
new String(passwordField.getPassword()).trim();
```

- Input validation is performed to check for empty fields.

```
if (email.equals("Email address") || email.isEmpty() ||
   password.isEmpty()
{
showErrorMessage("Please fill in all fields");

return;
}
```

- The controller is then called to perform authentication.

```
boolean success = authController.login (email, password);

if (success)
// Add a subtle success animation

showSuccessMessage("Login successful!");

dispose();
}
```

3.3.2 Authorization – Role-Based Access Control

After successful authentication, the system calls role-checking methods.

```
new HomeFrame(AuthController.isManager());
```

The method `AuthController.isManager()` determines the access level of the user:

- **Manager:** Full access to the system.
- **Staff:** Restricted access, cannot view/manage other staff information.

Feature / Data	Staff	Manager
Login	✓	✓
View Customer table	✓	✓
View Product table	✓	✓
View Pet table	✓	✓
View Bill table	✓	✓
View Staff table	✗ (self only)	✓
Edit other staff info	✗	✓
Assign roles and permissions	✗	✓
View reports (dashboard)	✓ (limited)	✓ (full)

Table 3.1: User Permissions Table

3.3.3 Sequence Diagram of Login Authentication

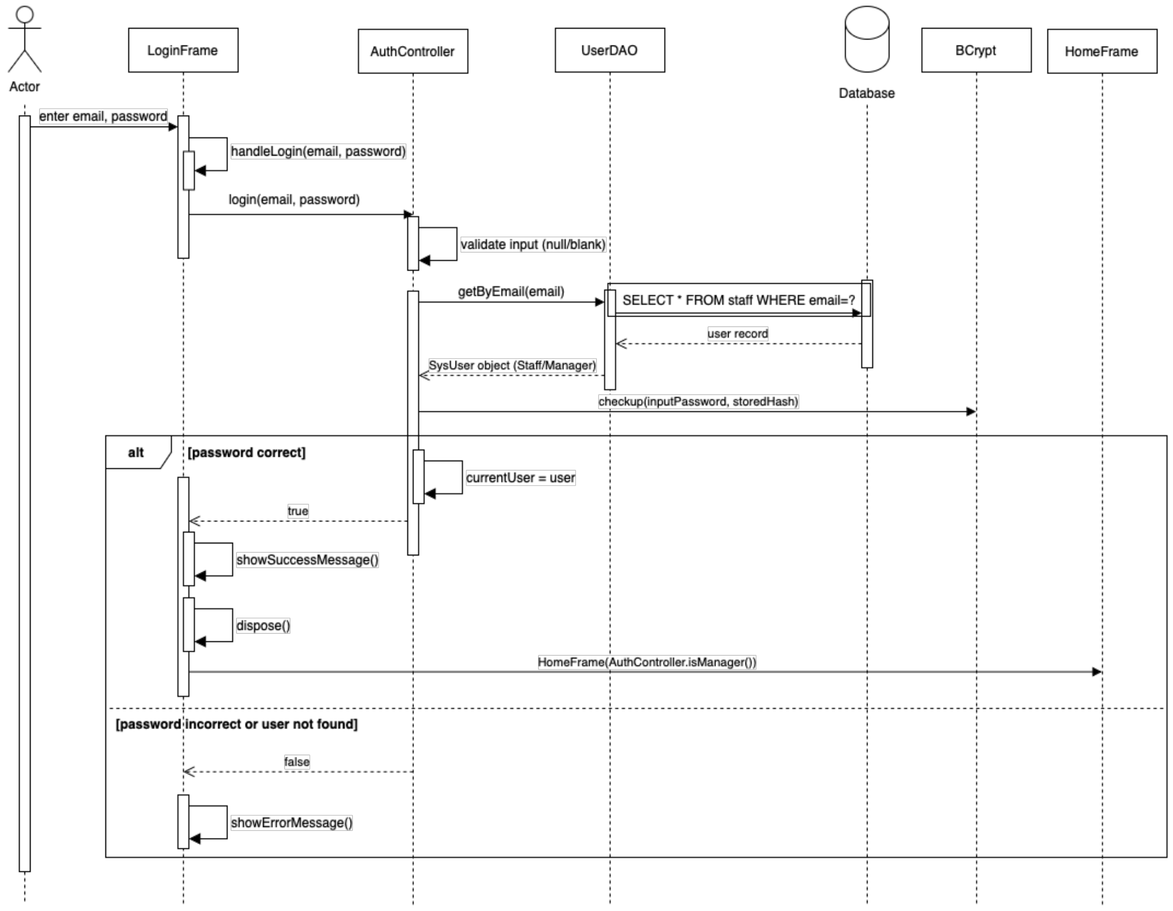

Actor
LoginFrame
AuthController
UserDAO
Database
BCrypt
HomeFrame
enter email, password
handleLogin(email, password)
login(email, password)
validate input (null/blank)
getByEmail(email)
SELECT * FROM staff WHERE email=?
user record
SysUser object (Staff/Manager)
checkup(inputPassword, storedHash)
alt
[password correct]
currentUser = user
true
showSuccessMessage()
dispose()
HomeFrame(AuthController.isManager())
[password incorrect or user not found]
false
showErrorMessage()

Figure 3.2: Sequence diagram of Login Authentication

3.4 User Management

3.4.1 Overview

The User Management module handles administration of all user types within the system, including internal users (Staff) and external users (Customers). It provides full CRUD capabilities, role-based access control, and ensures data integrity and security.

3.4.2 Staff Management

Access Control:

Only users with the `Manager` role can access this section.

Key Features:

- View Staff Table
- Add New Staff (passwords hashed with BCrypt)
- Edit Staff Information
- Delete Staff with confirmation
- Search Staff by ID

Functional Description:

- **Viewing Staff:** Data fetched via `UserController` and displayed in a table.
- **Adding Staff:** Trigger modal form, validate inputs, hash password, store via `UserDAO`.
- **Editing Staff:** Editable pre-filled form for modification.
- **Deleting Staff:** Requires confirmation, then removes staff and updates table.
- **Searching:** Filters table by ID; shows message if not found.

3.4.3 Customer Management

Access Control:

Both `Staff` and `Managers` have access.

Key Features:

- View Customer Table
- Add New Customer (no credentials required)
- Edit Customer Information
- Delete Customer with confirmation
- Filter/Search by loyalty levels or ID

Functional Description:

- **Viewing:** Use `CustomerController` to fetch all data.
- **Adding:** Validated input with real-time checks; duplicate email/phone prevented.
- **Editing:** Pre-filled editable form; validated and saved.
- **Deleting:** Prompts confirmation, deletes from DB and updates UI.
- **Searching:** Filters by ID with feedback if not found.

3.5 Product Management

3.5.1 Product Types (Toys, Food, Medicine)

Access Control:

Both `Staff` and `Manager` roles have access to the Product Management module.

Key Features:

- **View Product List:** Displays all products with attributes such as name, category (toy, food, medicine), price, description, and stock quantity.
- **Add New Product:** Opens a data-entry form to add new products.
- **Edit Product Information:** Enables updating fields like name, category, price, description, and stock.
- **Delete Product:** Removes a product after confirmation.
- **Filter/Sort Products:** Enables filtering by category or sorting by price or name.

Functional Description:

- Upon entering the product section, the system retrieves product data via `ProductController` and displays it in a table.
- The "Add Product" button opens a validated input form for product creation.

- “Edit” actions allow users to modify pre-filled product fields and update the database.
- “Delete” operations require confirmation before removing the record and refreshing the table.
- Filtering and sorting tools enhance product browsing and search efficiency.

3.5.2 Inventory Control

Access Control:

Accessible by both `Staff` and `Manager` roles.

Key Features:

- **View Inventory Status:** Real-time stock levels for all products.
- **Update Stock Levels:** Reflects changes due to purchases or sales.
- **Low Stock Alerts:** Notifies users when stock falls below set thresholds.
- **Inventory Reports:** Periodically generates tables or charts for trend analysis.

Functional Description:

- Displays product list with corresponding stock levels.
- Users select products to update stock levels.
- The system compares stock to predefined limits and triggers alerts.
- Reports support restocking and purchasing decisions through historical data.

3.6 Pet Management

3.6.1 Pet Registration

- Accessible by both `Staff` and `Manager`.
- Register pets with information such as type, breed, age, gender, and characteristics.

- Pets are linked to corresponding customer profiles.
- Information can be updated or edited as needed.

3.6.2 Pet Sales

- Transactions handled by `Staff` or `Managers`.
- Captures transaction data: pet info, customer info, sale price.
- Updates pet inventory after successful sale.
- Maintains sales history for analysis and reporting.
- Integrated with payment system for streamlined processing.

3.7 Billing System

3.7.1 Shopping Cart

Access Control:

Accessible by both `Staff` and `Manager` roles.

Key Features:

- **Add Products:** Add selected products with specific quantities.
- **Update Quantities:** Adjust product quantities before checkout.
- **Remove Products:** Remove items from the cart.
- **Calculate Totals:** Real-time subtotal, taxes, discounts, and total.
- **Save Cart:** Option to save for later or modify before payment.

Functional Description:

- Products added from the product list update the cart view with details.
- Quantities and totals adjust dynamically upon changes.

- Remove buttons allow for item deletion.
- Final cart can be submitted for invoice generation.

3.7.2 Invoice Generation

Access Control:

Performed by authorized `Staff` or `Managers`.

Key Features:

- **Automated Invoice Creation:** Details all cart items, prices, discounts, taxes, and final amount.
- **Unique Invoice Number:** Each invoice receives a unique, sequential number.
- **Persistent Storage:** Saved to database for auditing and reporting.
- **Invoice Display and Printing:** Viewable on screen and can be printed or saved as PDF.

Functional Description:

- Checkout gathers all cart data to build invoice.
- A unique invoice number is generated.
- Invoice data is stored in the database and linked to the customer and transaction.
- Final invoice is displayed for review and can be printed or saved.

Process Flow Description:

1. Staff initiates checkout.
2. System gathers shopping cart details and calculates totals.
3. System generates a unique invoice number.
4. Invoice and transaction details are saved to the database.
5. Invoice is displayed on screen for staff.

6. Staff prints or saves invoice as PDF.

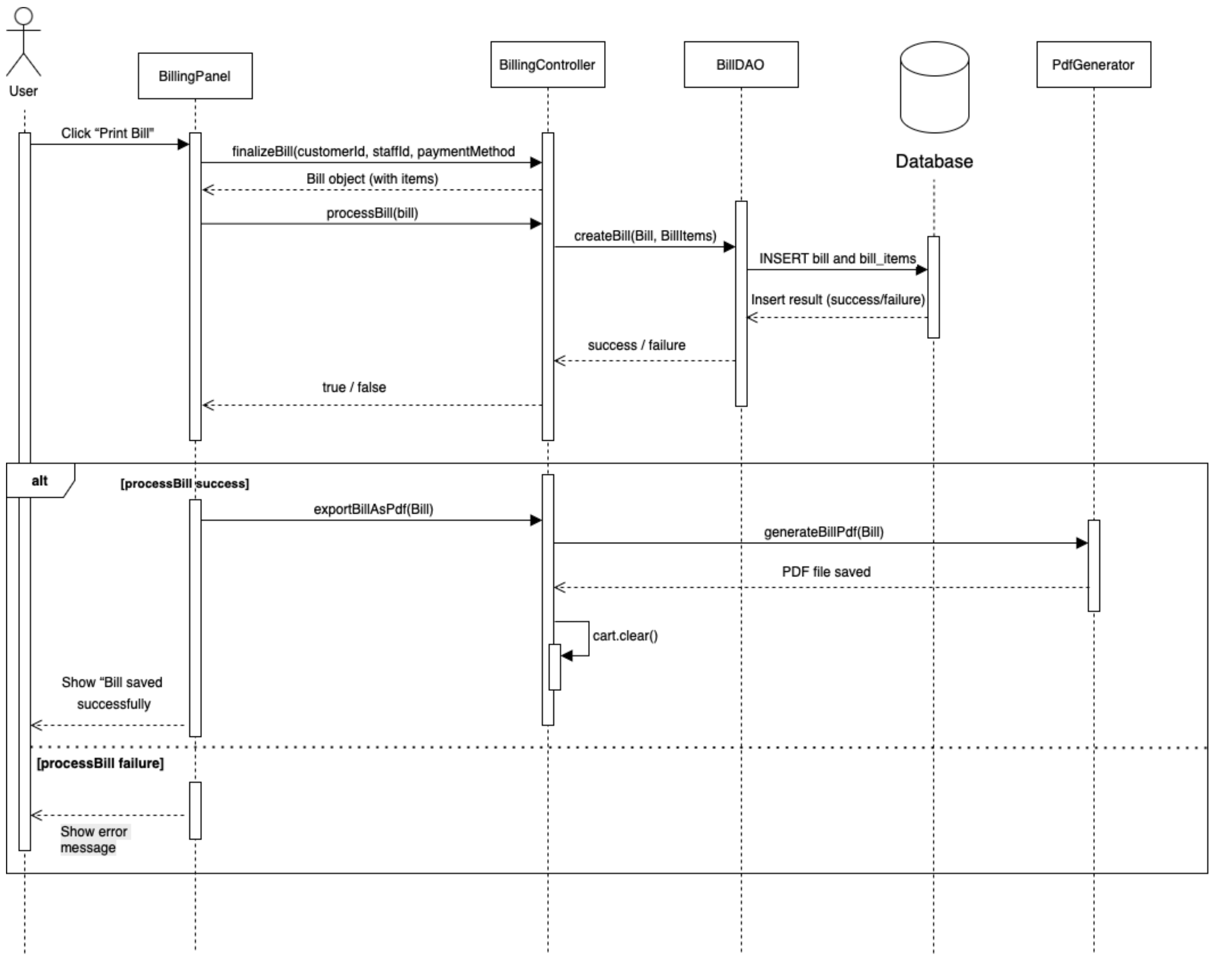


Figure 3.3: Sequence Diagram of Invoice Generation

Sample Invoice Format:

```
Pet Shop Invoice

Transaction Time: 2025-06-29 12:31:10

Staff ID: 19

Customer ID: 1

Payment Method: CASH

Items Purchased:

[PET] Whiskers x1 - $400.00

Total Amount: $400.00
```

Figure 3.4: The details of pet shop invoice

The invoice format clearly includes:

- Timestamp of transaction
- Staff ID and Customer ID
- Payment method
- Itemized purchase list
- Discounts, tax, and total amount

3.8 CSV Reporting

Access Control:

Manager and staff members have permission to generate and access CSV reports related to invoices and sales, ensuring sensitive financial data remains secure.

Key Features:

- **Professional CSV Report Generation:** Users can effortlessly create visually well-structured CSV reports, including invoices, daily summaries, or customizable date-range sales reports, suitable for internal review or official documentation.
- **Branding and Layout Customization:** Reports support company branding elements such as logos and customizable invoice layouts to maintain a consistent and professional appearance across all documents.
- **Separate File Export for Each Report Type:** The system allows exporting individual report types into separate CSV files, such as Financial Report, Sales Report, Pet Statistics, Inventory Report, Customer Report, and Staff Report.
- **Export, Save, and Print:** CSV reports can be exported and saved locally or on the system, or printed for physical record-keeping, providing flexible options for users.
- **Report Storage and Retrieval:** Generated reports are systematically stored within the system, enabling easy access and retrieval for future reference, auditing, or analysis.

Function Description:

Authorized personnel select report parameters such as date ranges, customer filters, or specific report types. The system then processes and formats stored invoice and sales data into clean, readable CSV files containing all relevant details. Users can save these reports locally, email them directly to stakeholders, or print official copies. The built-in storage system ensures reports remain well-organized and accessible over time.

Sample Report Format:

Pet Shop Report - Sales Report	
Generated: 2025-06-29 13:17:29	
Metric	Value
Total Orders	1
Total Revenue	$400.00
Average Order Value	$400.00
Best Month	Jun ($400.00)

Figure 3.5: The details of sales report

3.9 Reporting Dashboard

The Reporting Dashboard provides a comprehensive, real-time overview of key metrics to effectively manage store operations. It consolidates data across areas such as pets, products, staff, revenue, and inventory, delivering valuable insights to support timely and accurate decision-making.

Key Features:

- **Summary Metrics:** Displays aggregated information such as total registered pets, total products in inventory, active staff count, total orders, average revenue per order, and total revenue over selectable time periods. These metrics are retrieved dynamically from the system's controllers or DAOs.For example:

```
int totalProducts =
productController.getAllProducts().size();

int totalPets = petController.getAllPets().size();

int totalCustomers =
CustomerController.getAllCustomers().size();

int totalorders = billing-getTotalOrders();

BigDecimal totalRevenue = billing.getTotalRevenue ();

if (totalRevenue == null)

    totalRevenue = BigDecimal.ZERO;

int totalStaff = 0;

try
{
totalStaff = userDAO.getAllStaff().size();
}
catch (Exception e)
{
totalStaff = 0;
}

// Calculate additional stats
BigDecimal avgOrderValue
= totalorders > 0 && totalRevenue. compareTo (BigDecimal.ZERO
   ) > 0 ?

totalRevenue.divide(BigDecimal.valueOf(totalOrders), 2,
   BigDecimal.ROUND_HALF_UP) : BigDecimal.ZERO;
```

```

int lowStockProducts = getLowStockCount();
```

This code illustrates how the dashboard calculates the average order value for display.

- **Low Stock Alerts:** Highlights products with inventory levels below safety thresholds, enabling timely restocking to prevent business interruptions.
- **Sales Reports:** Interactive charts showing sales trends by day, week or month; revenue breakdown by product type, pet services and staff performance analysis.
- **Order Analysis:** Tracks the number of orders by status (completed, pending, canceled) with filtering options by date range and order status.
- **Detailed Navigation:** The Quick Action section within the dashboard enables users to quickly access frequently used management functions, including:
  - Pet Management
  - Product Management
  - Order Management
  - Revenue Reports

Access Control:

The Reporting Dashboard is accessible to users with roles of `Staff` or `Manager`. Since these are the only user roles available, all authorized users have permission to view and manage the dashboard, ensuring proper control over sensitive data.

Implementation Notes:

These quick actions are implemented as buttons or clickable icons that redirect users to corresponding modules. Data aggregation and calculations (e.g. average order value) are performed through controller-service-DAO layers to ensure up-to-date and accurate information.

Chapter 4

Database Design

4.1 Entity-Relationship Diagram (ERD)

The Pet Shop Management System is designed using a relational database model, which structures data into well-defined entities and establishes relationships through foreign keys. The core entities in the system include:

- `Staff`
- `Customers`
- `Pets`
- `Products`
- `Bills`
- `Bill_Items`

These entities are interconnected to ensure referential integrity, data consistency, and to support efficient querying for core operations such as order processing, billing, reporting, and customer tracking.

Key Relationships

- A `staff` member can create multiple `bills` (1:N).
- A `customer` may be linked to multiple `bills`.

- Each `bill` can contain multiple entries in the `bill_items` table.
- A `bill_item` can reference either a `pet` or a `product`.

This ERD structure facilitates the modular a nd scalable nature of the database, making it adaptable to future enhancements such as service modules or promotional offerings.

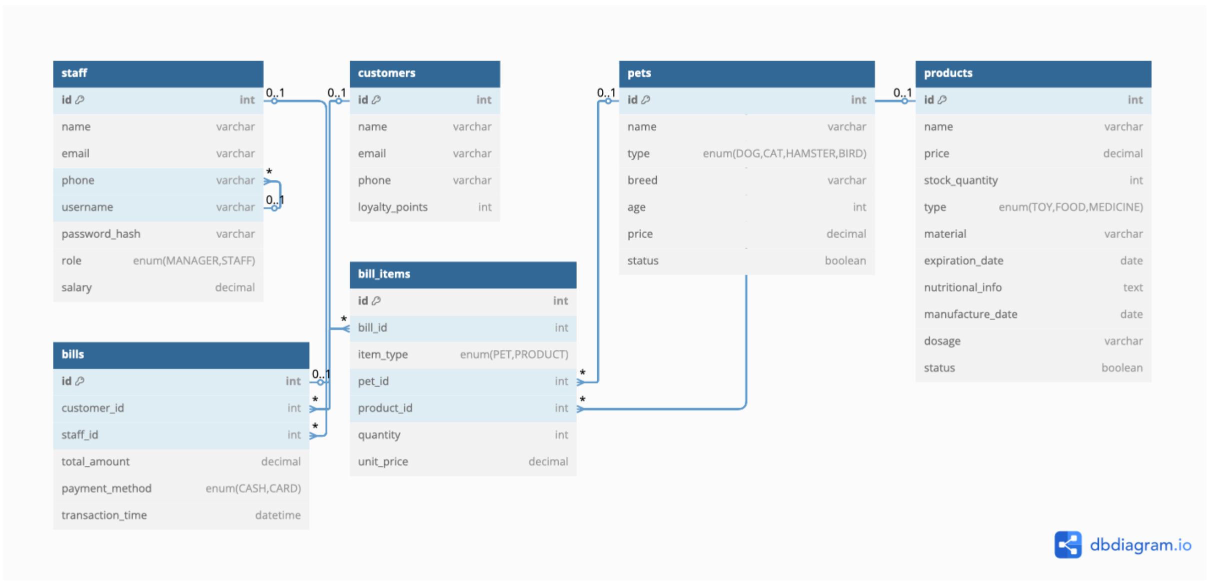
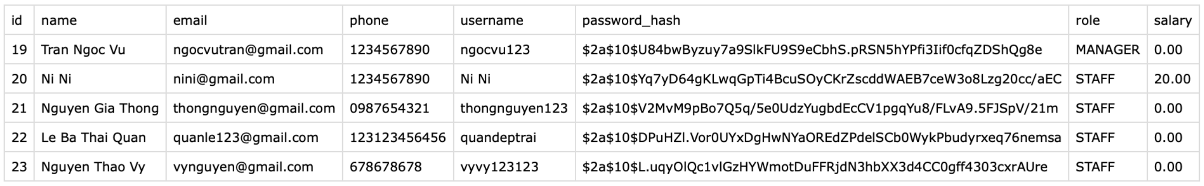


Figure 4.1: Entity-Relationship Diagram of the Pet Shop Management System

4.2 Table Specifications

4.2.1 Staff Table



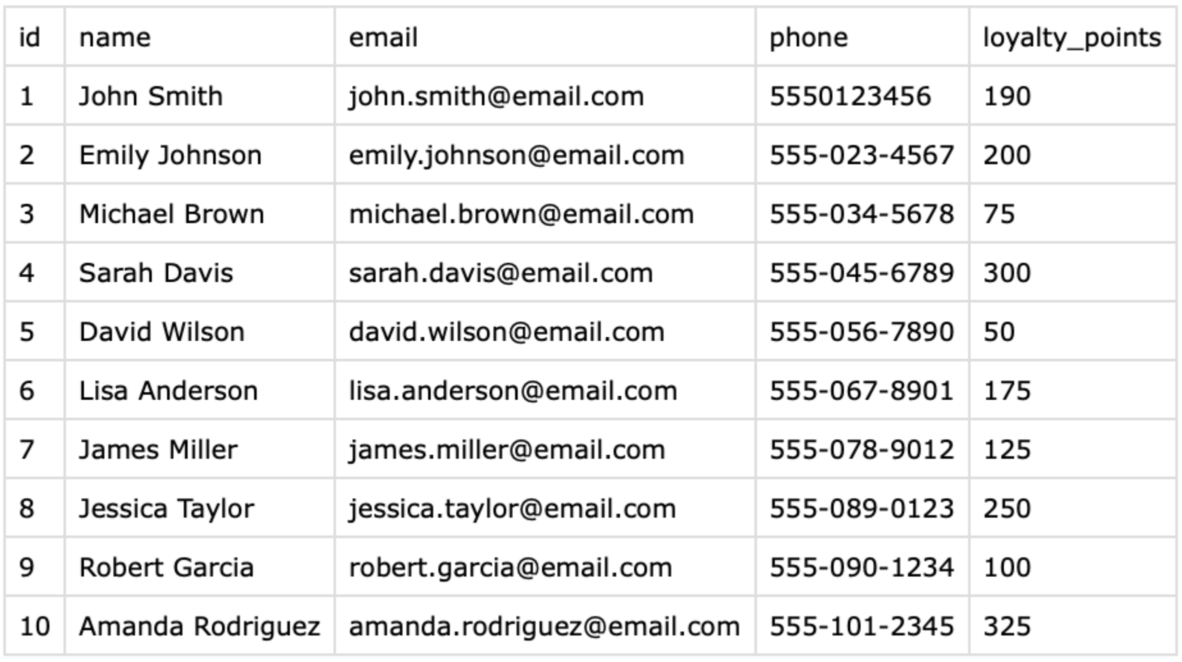
id	name	email	phone	username	password_hash	role	salary
19	Tran Ngoc Vu	ngocvutran@gmail.com	1234567890	ngocvu123	$2a$10$U84bwByzuy7a9SlkFU9S9eCbhS.pRSN5hYPfi3Iif0cfqZDShQg8e	MANAGER	0.00
20	Ni Ni	nini@gmail.com	1234567890	Ni Ni	$2a$10$Yq7yD64gKLwqGpTi4BcuSOyCKrZscddWAEB7ceW3o8Lzg20cc/aEC	STAFF	20.00
21	Nguyen Gia Thong	thongnguyen@gmail.com	0987654321	thongnguyen123	$2a$10$V2MvM9pBo7Q5q/5e0UdzYugbdEcCV1pgqYu8/FLvA9.5FJSpV/21m	STAFF	0.00
22	Le Ba Thai Quan	quanle123@gmail.com	123123456456	quandeptrai	$2a$10$DPuHZl.Vor0UYxDgHwNYaOREdZPdelSCb0WykPbudyrxeq76nemsa	STAFF	0.00
23	Nguyen Thao Vy	vynguyen@gmail.com	678678678	vyvy123123	$2a$10$L.uqyOlQc1vlGzHYWmotDuFFRjdN3hbXX3d4CC0gff4303cxrAUre	STAFF	0.00

Figure 4.2: Staff Table

Field	Type and Description
id	INT, Primary Key, Auto-increment
name	VARCHAR(255)
email	VARCHAR(255), Unique
phone	VARCHAR(20)
username	VARCHAR(50), Unique
role	ENUM('STAFF', 'MANAGER')
salary	DECIMAL(10,2)

Table 4.1: Staffs Table Schema

4.2.2 Customers Table



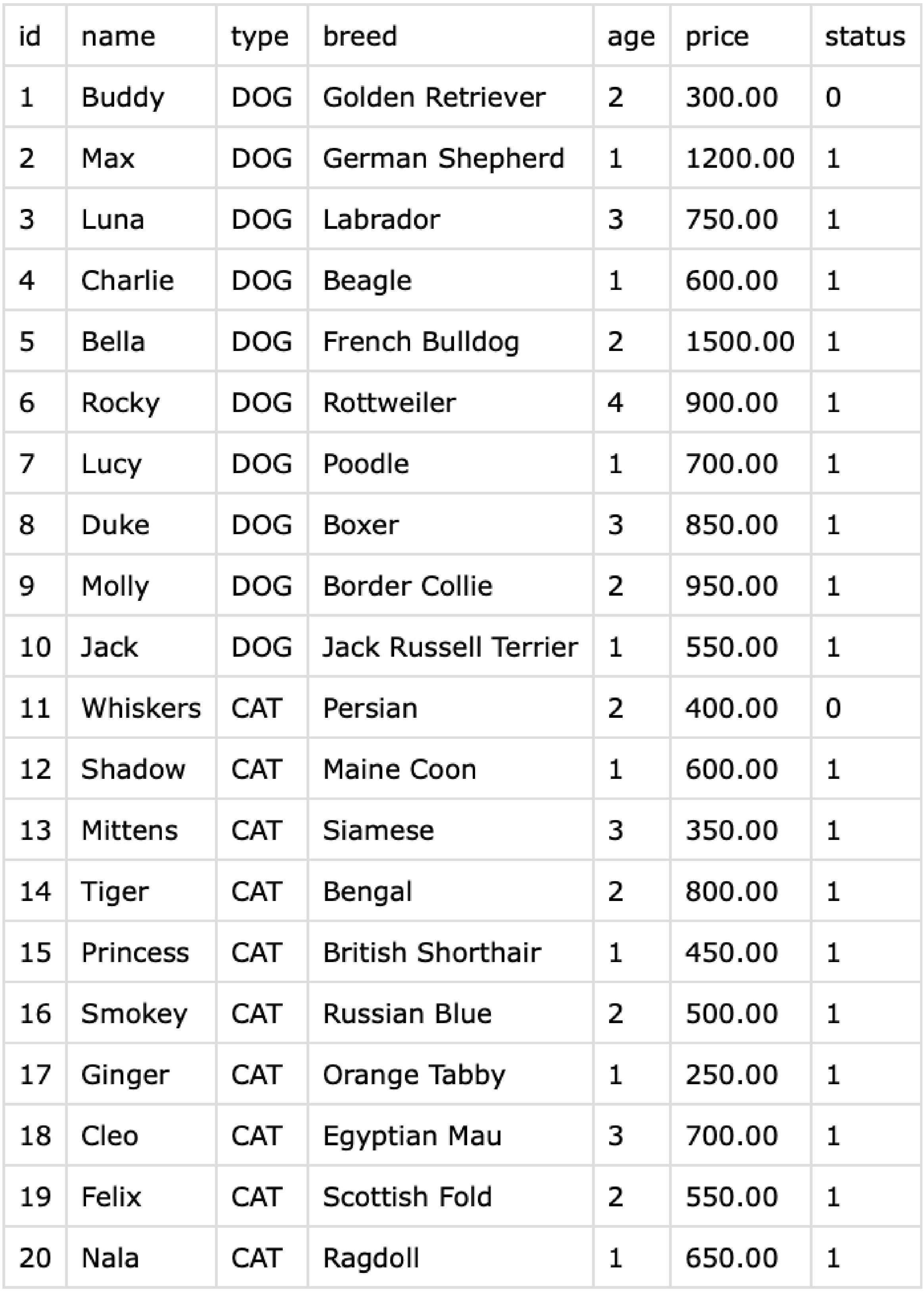
id	name	email	phone	loyalty_points
1	John Smith	john.smith@email.com	5550123456	190
2	Emily Johnson	emily.johnson@email.com	555-023-4567	200
3	Michael Brown	michael.brown@email.com	555-034-5678	75
4	Sarah Davis	sarah.davis@email.com	555-045-6789	300
5	David Wilson	david.wilson@email.com	555-056-7890	50
6	Lisa Anderson	lisa.anderson@email.com	555-067-8901	175
7	James Miller	james.miller@email.com	555-078-9012	125
8	Jessica Taylor	jessica.taylor@email.com	555-089-0123	250
9	Robert Garcia	robert.garcia@email.com	555-090-1234	100
10	Amanda Rodriguez	amanda.rodriguez@email.com	555-101-2345	325

Figure 4.3: Customers Table

Field	**Type and Description**
id	INT, Primary Key, Auto-increment
name	VARCHAR(255)
email	VARCHAR(255), Unique
phone	VARCHAR(20)
loyalty_points	INT

Table 4.2: Customers Table Schema

4.2.3 Pets Table



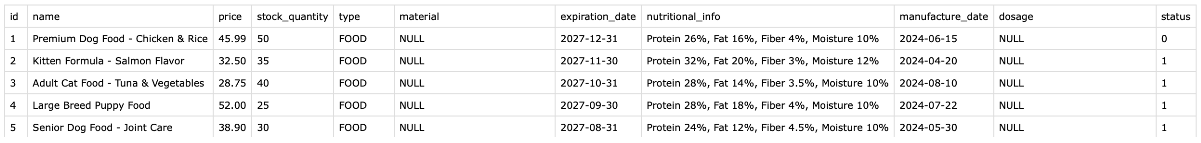
id	name	type	breed	age	price	status
1	Buddy	DOG	Golden Retriever	2	300.00	0
2	Max	DOG	German Shepherd	1	1200.00	1
3	Luna	DOG	Labrador	3	750.00	1
4	Charlie	DOG	Beagle	1	600.00	1
5	Bella	DOG	French Bulldog	2	1500.00	1
6	Rocky	DOG	Rottweiler	4	900.00	1
7	Lucy	DOG	Poodle	1	700.00	1
8	Duke	DOG	Boxer	3	850.00	1
9	Molly	DOG	Border Collie	2	950.00	1
10	Jack	DOG	Jack Russell Terrier	1	550.00	1
11	Whiskers	CAT	Persian	2	400.00	0
12	Shadow	CAT	Maine Coon	1	600.00	1
13	Mittens	CAT	Siamese	3	350.00	1
14	Tiger	CAT	Bengal	2	800.00	1
15	Princess	CAT	British Shorthair	1	450.00	1
16	Smokey	CAT	Russian Blue	2	500.00	1
17	Ginger	CAT	Orange Tabby	1	250.00	1
18	Cleo	CAT	Egyptian Mau	3	700.00	1
19	Felix	CAT	Scottish Fold	2	550.00	1
20	Nala	CAT	Ragdoll	1	650.00	1

Figure 4.4: Pets Table

Field	**Type and Description**
id	INT, Primary Key, Auto-increment
type	ENUM('DOG', 'CAT', 'HAMSTER', 'BIRD')
breed	VARCHAR(100)
age	INT
price	DECIMAL(10,2)
status	BOOLEAN (1 = available, 0 = sold)

Table 4.3: Pets Table Schema

4.2.4 Products Table



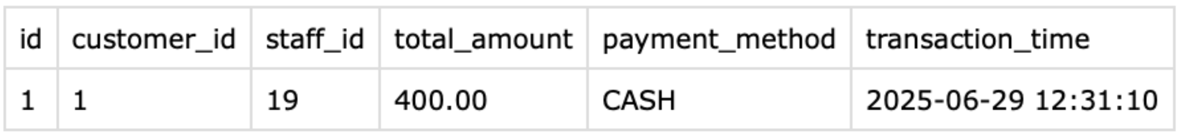
id	name	price	stock_quantity	type	material	expiration_date	nutritional_info	manufacture_date	dosage	status
1	Premium Dog Food - Chicken & Rice	45.99	50	FOOD	NULL	2027-12-31	Protein 26%, Fat 16%, Fiber 4%, Moisture 10%	2024-06-15	NULL	0
2	Kitten Formula - Salmon Flavor	32.50	35	FOOD	NULL	2027-11-30	Protein 32%, Fat 20%, Fiber 3%, Moisture 12%	2024-04-20	NULL	1
3	Adult Cat Food - Tuna & Vegetables	28.75	40	FOOD	NULL	2027-10-31	Protein 28%, Fat 14%, Fiber 3.5%, Moisture 10%	2024-08-10	NULL	1
4	Large Breed Puppy Food	52.00	25	FOOD	NULL	2027-09-30	Protein 28%, Fat 18%, Fiber 4%, Moisture 10%	2024-07-22	NULL	1
5	Senior Dog Food - Joint Care	38.90	30	FOOD	NULL	2027-08-31	Protein 24%, Fat 12%, Fiber 4.5%, Moisture 10%	2024-05-30	NULL	1

Figure 4.5: Products Table

Field	**Type and Description**
id	INT, Primary Key, Auto-increment
type	ENUM('TOY', 'FOOD', 'MEDICINE')
stock_quantity	INT
status	BOOLEAN (1 = in stock, 0 = out of stock)
expiration_date	DATE (nullable)
nutritional_info	TEXT (nullable)
dosage	VARCHAR(100) (nullable)
material	VARCHAR(100) (nullable)

Table 4.4: Products Table Schema

4.2.5 Bills Table



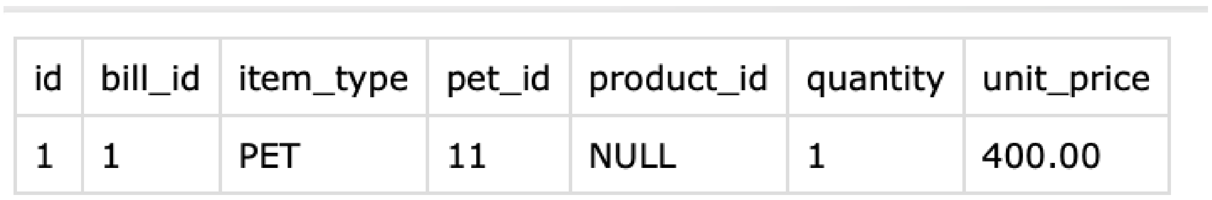
id	customer_id	staff_id	total_amount	payment_method	transaction_time
1	1	19	400.00	CASH	2025-06-29 12:31:10

Figure 4.6: Bills Table

Field	Type and Description
id	INT, Primary Key, Auto-increment
customer_id	INT, Foreign Key referencing Customers(id)
staff_id	INT, Foreign Key referencing Staff(id)
total_amount	DECIMAL(10,2)
payment_method	ENUM('CASH', 'CARD')
transaction_time	DATETIME

Table 4.5: Bills Table Schema

4.2.6 Bill_Items Table



id	bill_id	item_type	pet_id	product_id	quantity	unit_price
1	1	PET	11	NULL	1	400.00

Figure 4.7: Bill Items Table

Field	Type and Description
id	INT, Primary Key, Auto-increment
bill_id	INT, Foreign Key referencing Bills(id)
item_type	ENUM('PET', 'PRODUCT')
pet_id	INT (nullable, used when item_type = 'PET')
product_id	INT (nullable, used when item_type = 'PRODUCT')
quantity	INT
unit_price	DECIMAL(10,2)

Table 4.6: Bill_Items Table Schema

4.3 Data Relationships

- One `Staff` → Many `Bills` (1:N)
- One `Customer` → Many `Bills` (1:N)
- One `Bill` → Many `Bill_Items` (1:N)
- One `Bill_Item` → One `Pet` or One `Product` (1:1)

Design Advantages

- **Scalability:** Easily extendable to future modules such as pet grooming or appointment booking.
- **Role-Based Access Control:** Enforced through the `role` field in the `Staff` table.
- **Efficient Reporting:** Optimized via indexing and normalization for fast data analysis.

Chapter 5

Front-End

5.1 Overview

The **front-end** serves as the **client-side interface** for the **Pet Shop Management System**, providing an **interactive** and **user-friendly** experience for `STAFF` and `MANAGER` to manage the pet store's operations.
It acts as the primary user interaction layer, enabling users to:

- Log in
- Manage `Pets, Customers, Orders`
- Invoices through well-organized management panels

Built using `Java Swing`, the **front-end** provides a ***desktop-based*** graphical user interface with intuitive navigation across different modules such as `Dashboard, Billing, Reports, and Inventory Management`. The interface responds to user actions like:

- Adding new `Pets`
- Editing `Customer` details
- Generating `Invoices`

while communicating with the **back-end services** to retrieve and update data in real-time. This design ensures a smooth and efficient workflow for daily store operations.

5.2 The Front-End Technology Stack

The **front-end** of the Pet Shop Management System is implemented using `Java Swing`, a part of the ***Java Foundation Classes (JFC)***, to provide a graphical desktop-based user interface.
It serves as the primary interaction layer for the application users, mainly `STAFF` and `MANAGER`, offering a clean and intuitive environment to perform daily operations such as:

- Logging in
- Managing `Pets`
- Handling `Customers` and `Orders`
- Generating `Invoices` or `Reports`.

`Java Swing` components like `JJTable, JFrame, JPanel, JButton, and JTextField` are used to build **dynamic forms** and **interactive management panels**. The user interface responds to various **user actions** (e.g., button clicks, form submissions) using event listeners (`ActionListener, MouseListener, etc.`).
Although it's a desktop application, the design emphasizes modular, responsive interaction, ensuring ease of use and smooth navigation across different functional areas such as `Dashboard, Pet Management, Billing, and Reporting`.

5.2.1 Structure & Routing

Structure:

The **front-end** of the Pet Shop Management System is built using `Java Swing`, organized into modular `Java classes` corresponding to *each screen and functionality* in the application. The folder structure follows a feature-based approach, with each form representing a specific module or operation.

Key components include:

- LoginForm.java
- Dashboard.java `Pets`

- PetForm.java `Customers` and `Orders`
- CustomerForm.java `Invoices` or `Reports`.
- `LoginForm.java`: User login interface
- `Dashboard.java`: Main menu after login
- `PetForm.java`: Add, edit, and manage pet information
- `CustomerForm.java`: Manage customer data
- `InvoiceForm.java`: Create and manage invoices
- `ReportForm.java`: View statistical reports
- `Main.java`: Entry point to launch the application

Additional reusable **UI components (e.g. buttons, tables, dialogs)** are implemented as helper classes or within the forms themselves. This separation enhances **code readability** and supports **future scalability**.

Routing

In `Java Swing desktop applications`, routing is not handled by a `URL` system like in web frameworks. Instead, screen navigation is managed through the creation, visibility control, or layout switching of `JFrame and JPanel` components.
Routing behavior in the Pet Shop Management System is handled as follows: - Upon successful login from `LoginForm`, the application transitions to the `Dashboard` window.
- Inside `Dashboard`, menu buttons or sidebar options trigger the loading of different panels such as:

- Pet Management (`PetForm`)
- Customer Management (`CustomerForm`)
- Billing (`InvoiceForm`)
- Reports (`ReportForm`)

This transition is commonly implemented by 2 ways:

- Calling `setVisible(false)` on the current window and opening a new `JFrame`, or
- Using a `CardLayout` within the dashboard to swap visible `JPanels`.

This approach allows **smooth, event-driven navigation** between application modules while keeping each screen logically separated.

5.2.2 UI-Flow

The `STAFF` logs into the system and navigates to the main dashboard to manage `Pets, Customers, Products, Invoices, and User Accounts`.

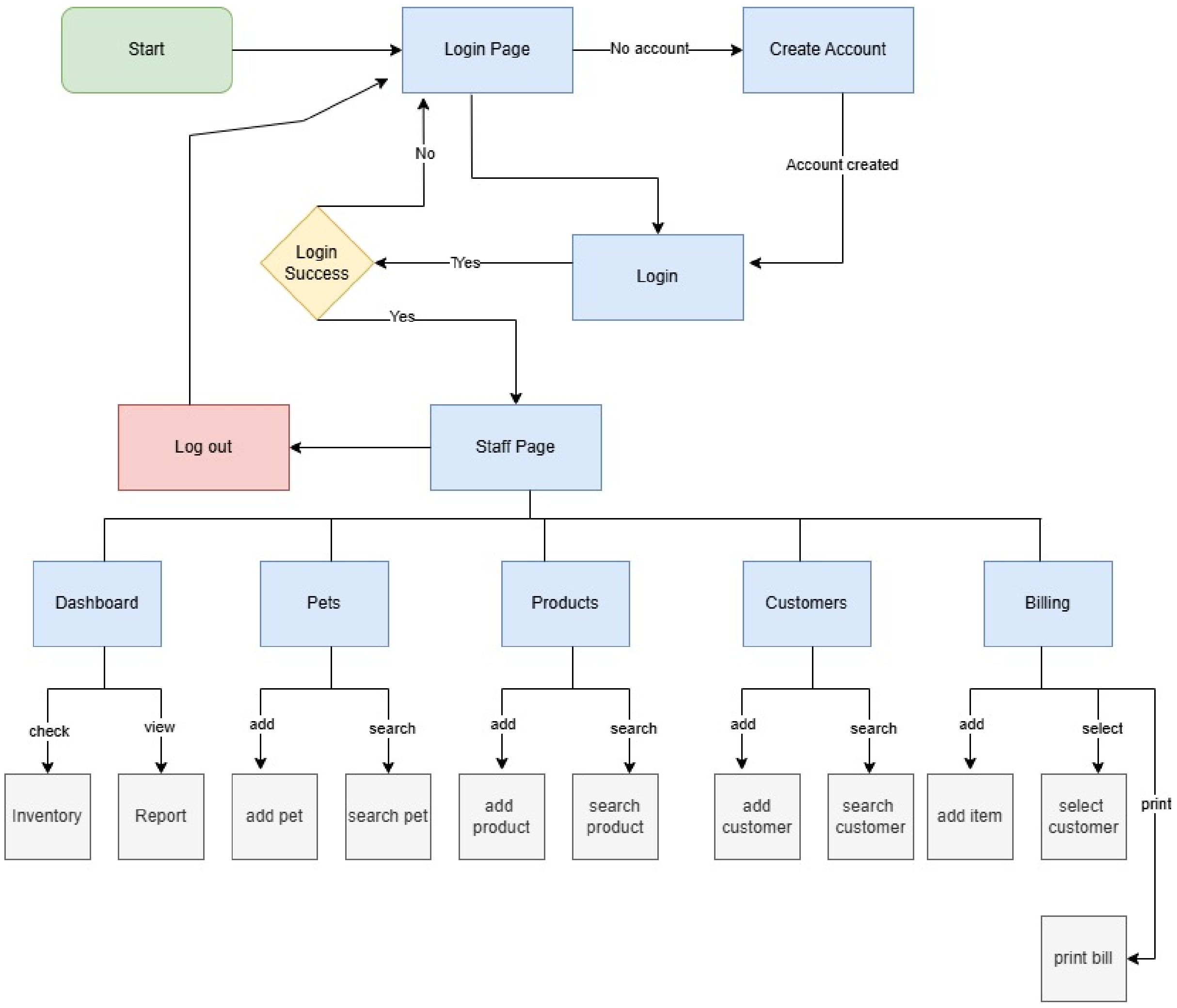


Figure 5.1: Staff UI Flow

5.2.3 Key Functionalities

The **front-end** implements several core functionalities:

- **Login and Authentication**: Allows staff to log into the system securely.
- **Pet Management**: Add, update, delete, and view pet information.
- **Customer Management**: Handle customer data and search for existing customers.
- **Product Management**: Manage pet-related products including food, accessories, etc.
- **Billing and Invoice**: Create and print invoices for customer purchases.
- **User Account Management**: Managing staff accounts and their roles.
- **Navigation UI**: Provides a user-friendly interface for switching between different modules.

5.3 Staff Interface

The Login Screen

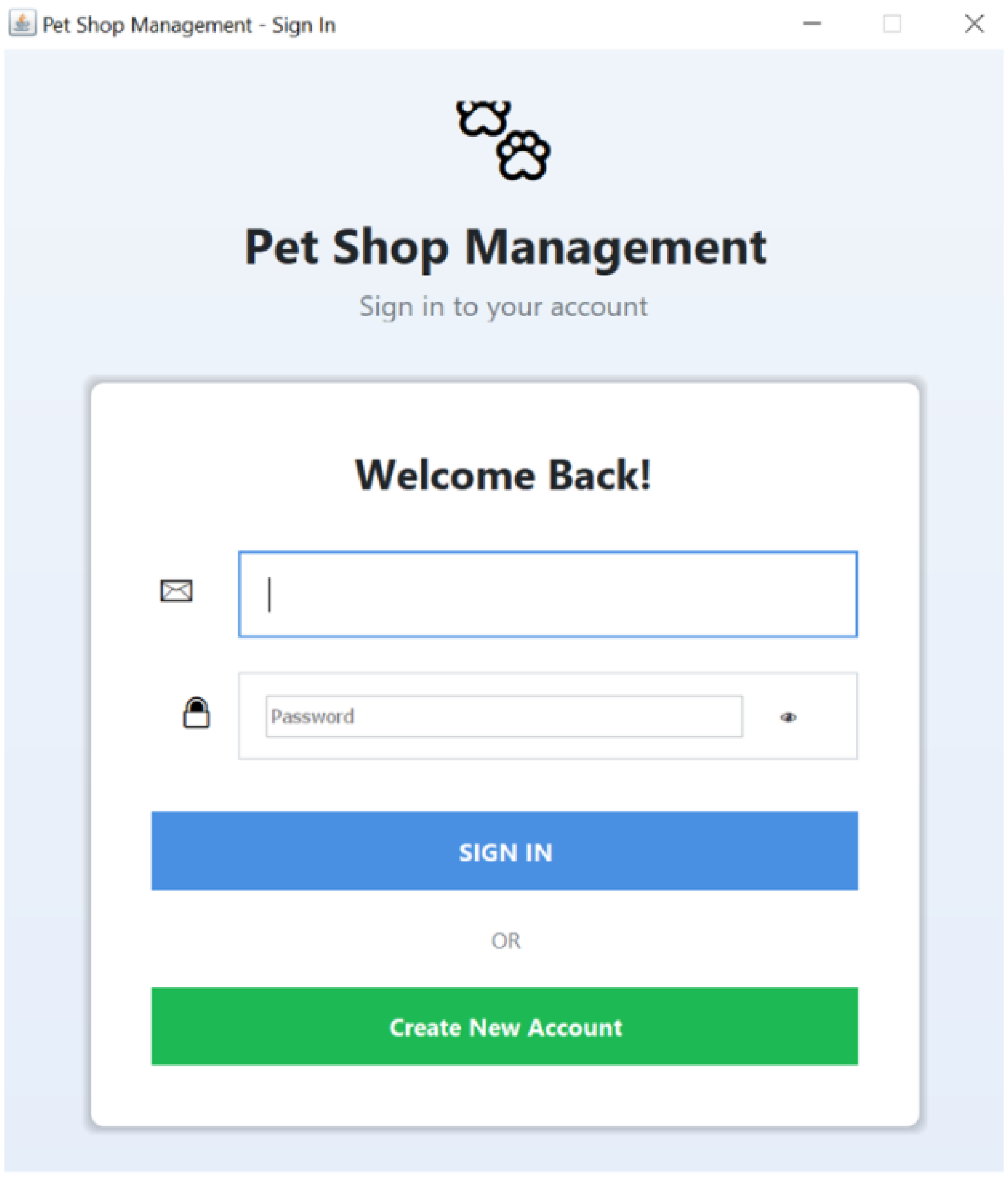
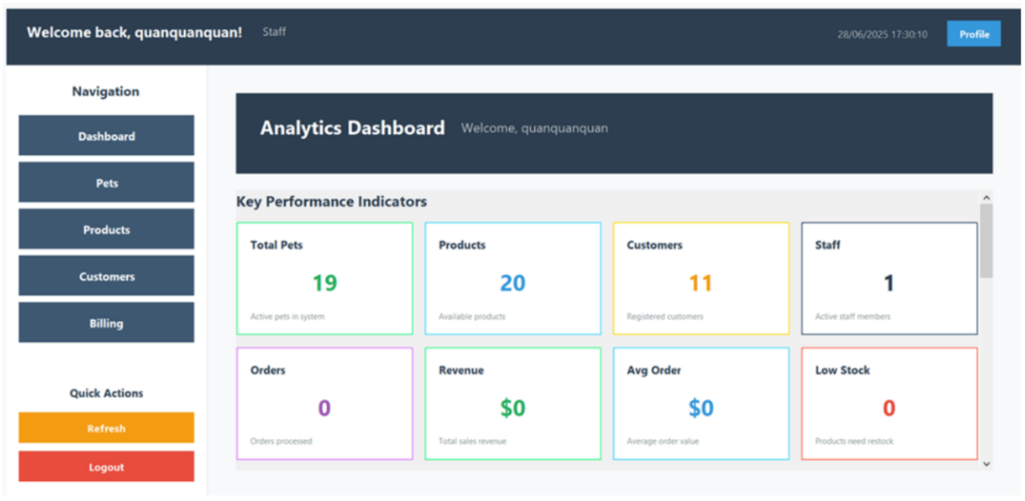

Pet Shop Management - Sign In

Figure 5.2: Login

The Main Dashboard Screen



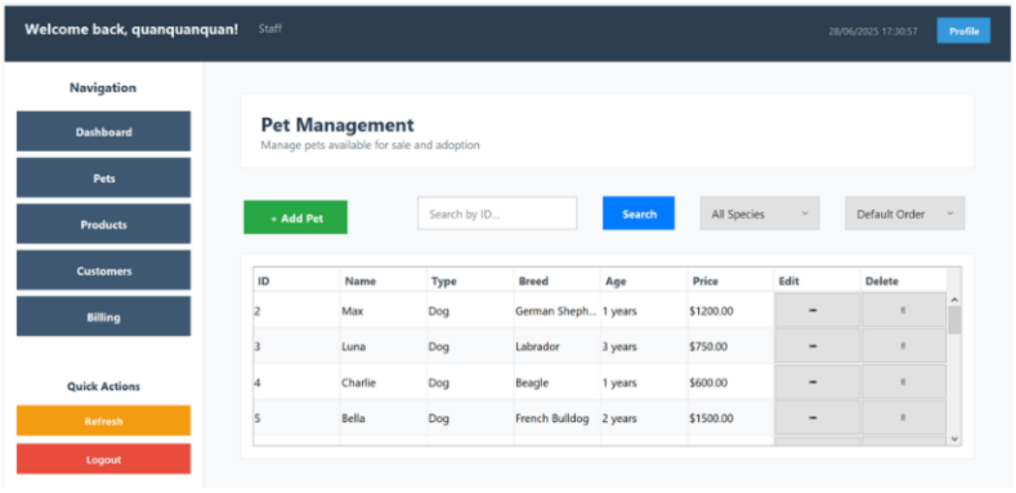
Analytics Dashboard Welcome, quanquanquan

Key Performance Indicators

Total Pets 19 Active pets in system	Products 20 Available products	Customers 11 Registered customers	Staff 1 Active staff members
Orders 0 Orders processed	Revenue $0 Total sales revenue	Avg Order $0 Average order value	Low Stock 0 Products need restock

Figure 5.3: Main DashBoard

The Pets Screen



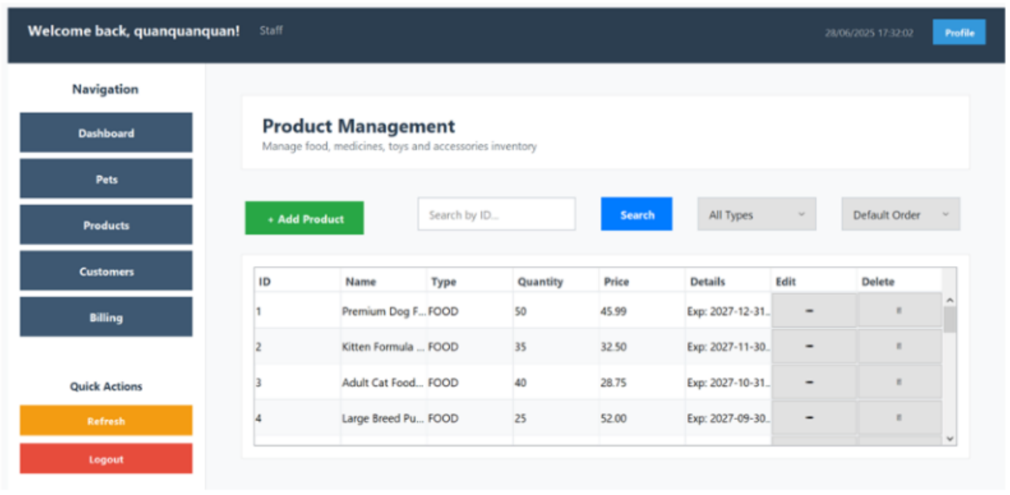
Pet Management

Manage pets available for sale and adoption

ID	Name	Type	Breed	Age	Price	Edit	Delete
2	Max	Dog	German Sheph...	1 years	$1200.00		
3	Luna	Dog	Labrador	3 years	$750.00		
4	Charlie	Dog	Beagle	1 years	$600.00		
5	Bella	Dog	French Bulldog	2 years	$1500.00		

Figure 5.4: Pets

The Products Screen


Welcome back, quanquanquan! Staff 28/06/2025 17:32:02

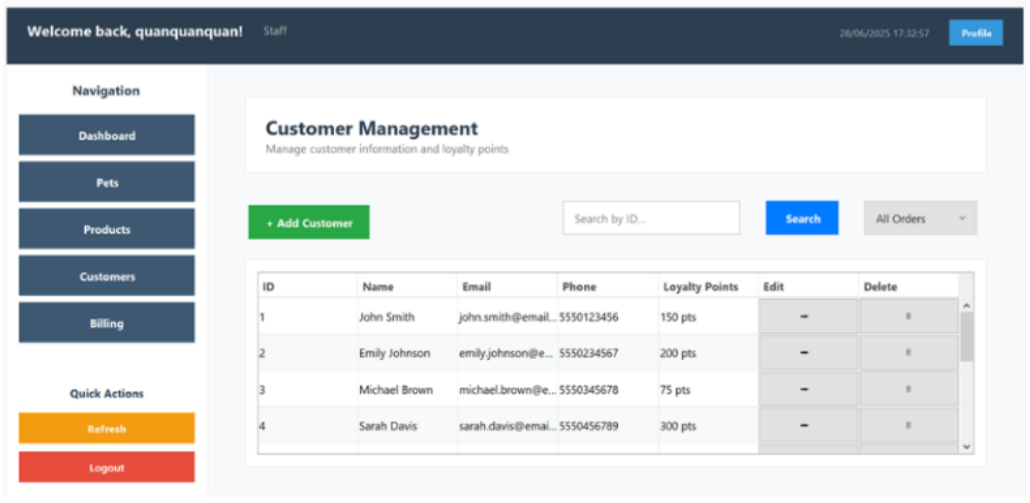
Product Management

Manage food, medicines, toys and accessories inventory

ID	Name	Type	Quantity	Price	Details	Edit	Delete
1	Premium Dog F...	FOOD	50	45.99	Exp: 2027-12-31...		
2	Kitten Formula ...	FOOD	35	32.50	Exp: 2027-11-30...		
3	Adult Cat Food...	FOOD	40	28.75	Exp: 2027-10-31...		
4	Large Breed Pu...	FOOD	25	52.00	Exp: 2027-09-30...		

Figure 5.5: Products

The Customer Screen


Welcome back, quanquanquan! Staff 28/06/2025 17:32:57

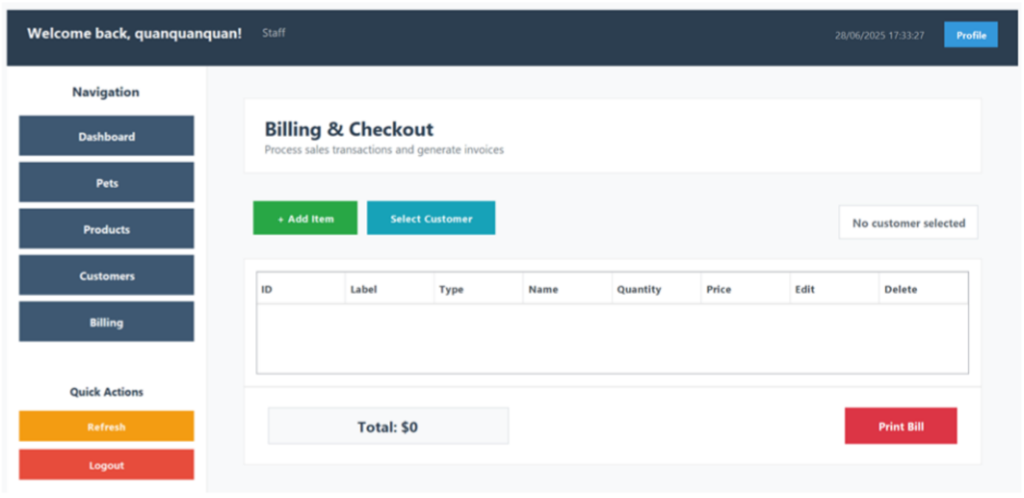
Customer Management

Manage customer information and loyalty points

ID	Name	Email	Phone	Loyalty Points	Edit	Delete
1	John Smith	john.smith@email...	5550123456	150 pts		
2	Emily Johnson	emily.johnson@e...	5550234567	200 pts		
3	Michael Brown	michael.brown@e...	5550345678	75 pts		
4	Sarah Davis	sarah.davis@emai...	5550456789	300 pts		

Figure 5.6: Customer

The Billing Screen



ID	Label	Type	Name	Quantity	Price	Edit	Delete

Figure 5.7: Billing

The Report Screen

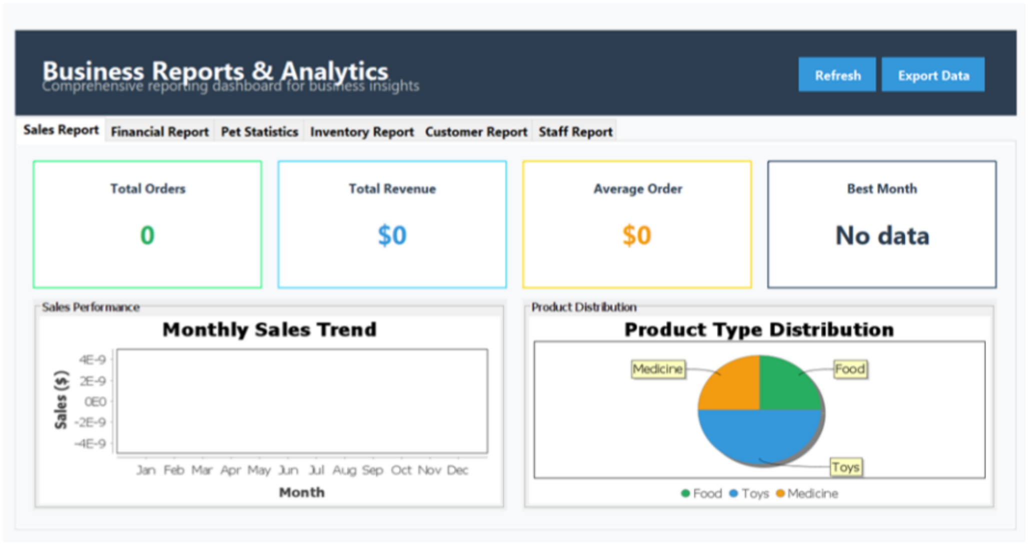


Figure 5.8: Report

Chapter 6

Back-End

6.1 Overview

The **back-end** of the **Pet Shop Management System** is the core engine that drives all business logic, data processing, and database interaction for the application. It is responsible for performing operations requested by the front-end interface, such as retrieving pet information, storing customer orders, processing invoices, and managing staff accounts. The system is architected using the ***Model-View-Controller (MVC)*** design pattern, which promotes a clear separation of concerns:

- Adding new `Pets`
- Editing `Customer` details
- Generating `Invoices`

6.2 Responsibilities of MVC Components

- `Model`: Represents the application's **core data** and **logic**. Each class in this layer corresponds to a real-world entity, such as Pet, `Customer`, `Invoice`, or `User`. These classes define the structure of the data and may include validation logic.
- `View`: Although the view is technically part of the **front-end** (`Java Swing UI`), it interacts closely with the **back-end** by sending user actions to the **Controller** and **displaying** the results. The **back-end** ensures that the data provided to the view is **accurate** and **up to date**.

- `Controller` **(Service layer)**: Acts as the intermediary between the `Model` and the `View`. It receives input from the `UI`, processes the data (using business rules), and calls the appropriate `DAO` methods to interact with the database. It ensures that the operations are valid before ***modifying data*** or ***returning results*** to the `UI`.

The back-end is further divided into:

- `DAO (Data Access Object)` classes, which encapsulate all database operations using `SQL` via `JDBC`.

- `Service classes`, which enforce application logic and serve as the core controller logic for each functional area.

By using MVC, the system achieves **modularity, testability, and maintainability**. Changes to the database, business rules, or `UI` logic can be made with **minimal impact** on other layers, making the system easier to evolve and maintain.

Key Technologies:

The back-end of the Pet Shop Management System is implemented using a combination of mature, reliable technologies to ensure robustness, performance, and maintainability. The following are the core technologies and components used:

Component	**Technology Used**	**Description**
Programming Language	Java (`JDK 17 or later`)	Primary language used for building business logic, data access, and controllers.
Database	`MySQL`	Relational database system used to store persistent data related to pets, customers, orders, etc.
Database Connectivity	`JDBC (Java Database Connectivity)`	Allows `Java` applications to connect and execute `SQL` queries on the `MySQL database`.
Architecture Pattern	***MVC (Model–View–Controller)***	Ensures separation of concerns, modularity, and maintainability.
Logging	`Log4j`	Used for logging events, errors, and system behavior for debugging and monitoring.
Utilities	`Apache Commons Lang / IO`	Provides helper functions for string handling, file manipulation, and other tasks.
Build Tool	`Apache Maven / Manual build`	Dependency management and build configuration.
IDE	`NetBeans / IntelliJ IDEA`	**IDEs** used for writing, debugging, and managing source code.

Table 6.1: Back-end Technology Stack

6.2.1 Back-End Architecture

The **back-end** of the **Pet Shop Management System** follows a layered architecture, with clear separation between data access, business logic, and data representation. This design ensures ***maintainability, modularity, and reusability*** across the application.**The core layers include:**

Model Layer:

Contains `Java` classes that represent the core business entities, such as `Pet, Customer, Invoice, User, and Product`.
These classes define fields that map directly to the database schema and may include basic validation or formatting logic.
Example: `Pet.java, Customer.java`

DAO Layer (Data Access Object):

Handles all interactions with the relational database using `JDBC`.
Each `DAO class` is responsible for implementing ***Create, Read, Update, Delete (CRUD)*** operations for its associated model.
This layer **encapsulates** `SQL queries`, database connection handling, and result parsing.
Example: `PetDAO.java, CustomerDAO.java`

Service Layer:

- Implements the business logic and acts as **a bridge** between the `front-end UI` and the `DAO layer`. - Validates data, coordinates between `multiple DAOs`, and prepares responses for the user interface. **Example:** `PetService.java, InvoiceService.java`

Controller/UI Communication (Handled via Event Listeners):

Although this is part of the `front-end`, it invokes methods from the service layer in response to user actions such as **button clicks, form submissions**, or **navigation events**.
This layered approach aligns with the ***MVC (Model–View–Controller)*** paradigm and ensures scalability, as future changes to the `database` or `UI` will not affect the core business logic.

6.2.2 Back-End Code Structure

The backend codebase is organized into clearly defined packages and folders to align with the layered architecture.

```
|
|-- model/
|   |-- Pet.java
|   |-- Customer.java
|   |-- Billing.java
|   |-- Product.java
|
|-- dao/
|   |-- PetDAO.java
|   |-- CustomerDAO.java
|   |-- BillDao.java
|   |-- UserDAO.java
|
|-- service/
|   |-- PetService.java
|   |-- CustomerService.java
|   |-- BillService.java
|   |- AuthService.java
|
|-- util/
|   |-- factory.java
|   |-- hash.java
|
|-- main/
    |-- AppLauncher.java
```

- `model/`: Contains ***POJO (Plain Old Java Object)*** classes representing database entities.
- `dao/`: Contains `DAO classes` that handle `SQL` operations.
- `util/`: Contains ***utility classes*** for common functions like **database connection**

pooling or **data validation**.

- `main/`: Contains the main entry point of the **back-end** (if executed independently or used for testing logic **without** `UI`).

This structure ensures that each responsibility is **encapsulated** in its own component, making it easier to **test, debug, and extend** the system.

6.2.3 Models

The Pet Shop Management System employs a model-driven architecture to represent and manage its core data entities. Each model class **encapsulates** the structure and behavior of a specific domain object, including `Pets, Users, Products`, and `Billing` information. These models define key methods used to ***perform database interactions*** and ***enforce logic*** related to their respective roles.

Pet Model (`Pet.java`)

The `Pet model` encapsulates data and operations related to animals in the store.

Method Name	**Description**
`addPet()`	Add a new pet to the system.
`updatePet()`	Update existing pet details.
`deletePet()`	Remove a pet from the database.
`getPetById()`	Retrieve pet details using its unique ID.
`getAllPets()`	Get a list of all pets.
`searchPetsByName()`	Search for pets using a keyword in their name.

Table 6.2: Methods of Pet Model

User Model (`User.java`)

This model manages data related to `Users` or `STAFF` who interact with the system.

Method Name	**Description**
`addUser()`	Register a new user or staff account.
`updateUser()`	Modify user information (e.g., name, role).
`deleteUser()`	Remove a user from the system.
`getUserById()`	Fetch a user's details using their ID.
`getAllUsers()`	Get a list of all registered users.
`loginUser()`	Authenticate a user's login credentials.

Table 6.3: Methods of User Model

Product Model (`Product.java`)

The Product model defines items such as `FOOD, TOY, MEDICINE` available in the system.

Method Name	**Description**
`addProduct()`	Add a new product to the inventory.
`updateProduct()`	Update existing product details.
`deleteProduct()`	Remove a product from the inventory.
`getProductById()`	Retrieve details of a product using its ID.
`getAllProducts()`	List all available products in the system.
`searchProducts()`	Find products by keyword (name or category).

Table 6.4: Methods of Product Model

Bill Model (`Bill.java`)

This model manages `Invoice` and `Billing` information associated with purchases.

Method Name	**Description**
`createInvoice()`	Generate a new invoice or bill.
`getInvoiceById()`	Retrieve details of a bill using its ID.
`getAllInvoices()`	Fetch a list of all invoices in the system.
`deleteInvoice()`	Remove an invoice record from the system.
`printInvoice()`	Print or export a bill for the customer.

Table 6.5: Methods of Bill Model

6.2.4 Controllers

`Controllers` in the **Pet Shop Management System** act as intermediaries between the user interface and the model layer:

- Handle inputs
- Perform logical checks
- Invoke model methods
- Return results to the **front-end**

$\longrightarrow$ **The following sections describe the key controllers and their responsibilities.**

AuthController (`controller.user.AuthController`)

This controller handles account authentication and registration logic. It keeps track of the currently authenticated user and manages permissions.

Method Name	**Description**
`login(String, String)`	Authenticate a user using email and password.
`signup(Staff, String)`	Register a new staff account and store the raw password securely.
`isManager()`	Check whether the currently logged-in user holds manager privileges.

Table 6.6: Methods of AuthController

UserController (`controller.user.UserController`)

The `UserController` manages all actions related to staff and user accounts, including creation, updates, and deletion.

Method Name	**Description**
`addStaff(Staff)`	Add a new staff member to the system.
`getAllStaff()`	Retrieve a list of all staff accounts.
`updateStaff(Staff)`	Update staff information. (Manager-only action)
`updateUser(SysUser)`	Modify general system user profile information.
`deleteStaff(int)`	Remove a staff account by ID. (Manager-only action)

Table 6.7: Methods of UserController

ProductController

This controller governs interactions with the product inventory, including **adding, updating, filtering, and deleting** products.

Method Name	**Description**
`addProduct(Product)`	Adds a new product to the database. Logs errors if any.
`updateStock(int, int)`	Adjust the stock quantity of a product based on the given change.
`getAllProducts()`	Retrieves the complete list of products from the database.
`getProductsByFilter(String, String)`	Fetch products by category and sort by price order.
`getAvailableProducts(String)`	Get all in-stock products of a given type (e.g., food, toy).
`updateProduct(Product)`	Update existing product information in the database.
`deleteProduct(int)`	Remove a product from the database by its ID.

Table 6.8: Methods of ProductController

Chapter 7

Security Implementation

7.1 Password Hashing (BCrypt)

The system implements secure password storage using `BCrypt` **hashing algorithm** to protect user credentials:

7.1.1 Implementation Details

- Uses `BCrypt` library for password hashing with salt generation
- Passwords are **never stored** in plaintext in the database
- Each password has a **unique salt** to prevent rainbow table attacks
- Adaptive cost factor provides future-proof security

Key Code Implementation

In `AuthController.java` - Password hashing during sign-up:

```java
staff.setPasswordHash(BCrypt.hashpw(rawPassword, BCrypt.gensalt()
  ));
```

Password verification during login:

```java
if (!BCrypt.checkpw(password, user.getPasswordHash())) {
        System.err.println("Login failed: Incorrect password.");
        return false;}
```

7.1.2 Security Benefits

- Protection against ***password cracking*** attempts
- Secure password verification ***without*** exposing plaintext
- Industry-standard ***cryptographic protection***
- Resistance to ***timing attacks***

7.2 Role-Based Access Control

The system implements hierarchical role-based access control to restrict system functionality based on user roles.

Role Hierarchy

- `MANAGER`: Full system access including **staff management, financial reports, system configuration**.
- `STAFF`: Limited access to daily operations like `Sales, Customer Management, Inventory`

Access Control Implementation

In `AuthController.java`:

```java
public static boolean isManager() {
        return currentUser instanceof Manager;
        }
```

Example usage in `UserController.java`:

```java
// Proceed with staff deletion
public boolean deleteStaff(int id) {
        if (!(AuthController.currentUser instanceof Manager))
            return false;
        }
```

Protected Operations

- STAFF CRUD operations (Manager only)
- Salary information access (Manager only)
- System configuration changes (Manager only)
- Advanced reporting features (Manager only)

7.3 Input Validation

Comprehensive input validation system **prevents** security vulnerabilities and **ensures** data integrity:

7.3.1 Validation Categories

- ***Format*** Validation: Email format, phone number format
- ***Business Rule*** Validation: Stock quantities, price ranges, age limits
- ***SQL Injection*** Prevention: Parameterized queries in all DAO operations
- Data ***Sanitization***: Trimming whitespace, preventing XSS

7.3.2 Implementation Example

In CustomerService.java:

```java
    public void validateCustomerData(String name, String email,
        String phone, int loyalty) {
    // Basic validation
    if (name == null || name.trim().isEmpty()) {
        throw new IllegalArgumentException("Customer name cannot
            be empty");
} if (email = = null | | email.trim().isEmpty()) throw new
    IllegalArgumentException ("Email cannot be empty");
}
if (phone == null || phone.trim().isEmpty()) {
    throw new IllegalArgumentException("Phone number cannot be
       empty");
}
```

Chapter 8

Business Logic

8.1 Loyalty Points System

Automated customer loyalty program that rewards frequent customers.

Point Calculation Logic:

- Customers earn *1 loyalty point* for every $10 spent
- Points are **automatically calculated** and **credited** after each purchase
- Point balance is **maintained** in customer records

Implementation:

In `BillingController.java`:

```java

public void applyLoyaltyPoints(Customer customer, BigDecimal
    total) throws SQLException
{
int points = total.divide(BigDecimal.TEN, RoundingMode.DOWN).
    intValue();
if (points > 0)
    {
    customer.addLoyaltyPoints(points);
    customerDao.updateCustomer(customer);
    }
}
```

Business Benefits:

- Encourages customer retention
- Increases average transaction value
- Provides customer behavior insights
- Automated point management reduces manual errors

8.2 Stock Management

Real-time inventory management system ensuring accurate stock levels:

Stock Control Features:

- Automatic stock deduction upon sale completion
- Stock validation before adding items to cart
- Real-time stock level checking
- Prevention of overselling

Implementation:

In `BillingController.java`:

```java
public void addProductToCart(Product product, int quantity){
    // Check if the product is already in the cart
    BillItem existingItem = cart.getAllItems().stream()
    .filter(i -> i.getItemType() == BillItem.ItemType.PRODUCT &&
       i.getProductId() == product.getId())
    .findFirst()
    .orElse(null);
    int currentInCart = existingItem != null ? existingItem.
       getQuantity() : 0 ;
    int remainingStock = product.getStockQuantity() -
       currentInCart;
    if (quantity > remainingStock)
    {throw new IllegalArgumentException("Only " + remainingStock
       + " items available for " + product.getName());
    }
    cart.addProduct(product, quantity);
}
```

8.3 Revenue Calculation

A comprehensive revenue tracking system that uses **cart functionality** and **bill data** to provide real-time financial insights, order analytics, and business performance monitoring.

Revenue Tracking Features:

- **Automatic revenue update** upon successful payment processing
- **Revenue breakdown support**: by staff, product, time (weekly, monthly)
- **Total, staff-wise, and product-type revenue reporting**
- **Seamless integration** with billing and reporting modules

Implementation:

In `BillingController.java`:

```
public boolean processBill(Bill bill) throws SQLException{
    List<BillItem> items = bill.getItems();
```

Stock validation before finalizing bill:

```
for (BillItem item : items)
{
    if (item.getItemType() == BillItem.ItemType.PRODUCT)
    {
        Product product $=$ productDao.getById(item.getProductId
           ());
        if ( product == null || product.getStockQuantity () <
           item.getQuantity())
        {
            throw new IllegalStateException("Not enough stock for
                " + (product != null ? product.getName() : "
               Unknown product") +
            ". Required: " + item.getQuantity() + ", Available: "
                + (product != null ? product.getStockQuantity() :
                0));
        }
    }
}
```

Payment processing:

```
boolean success = billingService.processPayment(bill, items);
    if (!success) return false;
```

Update Stock and Pet availability:

```
for (BillItem item : items)
{
            if (item.getItemType() == BillItem.ItemType.PET)
            {
                petDao.deletePet(item.getPetId());
            }
        }
        return true;
```

In Bill.java:

```
public BigDecimal getTotalAmount() {
    return items.stream()
    .map(BillItem::getTotal)
    .reduce(BigDecimal.ZERO, BigDecimal::add);
    }
```

Sample Reporting Methods in `BillingController.java`:

```java
public BigDecimal getRevenueByStaff(int staffId)
    {
        return billDao.getRevenueByStaff(staffId);
    }
public BigDecimal getTotalStaffRevenue()
    {
        return billDao.getTotalStaffRevenue();
    }
```

`Customer` activity methods:

```java
public java.util.Map<String, Integer> getCustomerActivityByMonth
   ()
{
    return billDao.getCustomerActivityByMonth();
}
public java.util.Map<String, Integer> getCustomerActivityByWeek()
{
    return billDao.getCustomerActivityByWeek();
}
```

8.4 PDF Invoice Generation

Invoice Features:

- Professional invoice layout
- Detailed line item breakdown
- Customer and business information
- Total calculations with taxes
- Digital signature capability

Implementation:

In `BillingController.java`

```
    public void exportBillAsPdf(Bill bill)
    {
        PdfGenerator.generateBillPdf(bill);
        cart.clear();
    }
```

Chapter 9

Deployment & Configuration

9.1 Database Setup

9.1.1 Prerequisites

- `MySQL Server 8.0 or higher`
- `Java Development Kit 11 or higher`
- `Maven 3.6 or higher`

9.1.2 Database Installation:

1. Install `MySQL Server`
2. Create database using **provided schema**
3. Configure `connection parameters`
4. Execute initial `data scripts`

Database creation:

Listing 9.1: Create Petshop database

```sql
DROP DATABASE IF EXISTS petshop_db;
CREATE DATABASE petshop_db;
USE petshop_db;
```

Table creation and initial data:

Listing 9.2: Create staff table

```sql
CREATE TABLE staff (
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  phone VARCHAR(20) NOT NULL,
  username VARCHAR(100) NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role ENUM('MANAGER','STAFF') NOT NULL,
  salary DECIMAL(10,2) NOT NULL DEFAULT '0.00',
  PRIMARY KEY (id),
  UNIQUE KEY email (email),
  UNIQUE KEY username (username),
  KEY idx_staff_email (email)
);
```

Listing 9.3: Create bills table

```sql
CREATE TABLE bills (
  id INT NOT NULL AUTO_INCREMENT,
  customer_id INT NOT NULL,
  staff_id INT NOT NULL,
  total_amount DECIMAL(10,2) NOT NULL,
  payment_method ENUM('CASH','CARD') NOT NULL,
  transaction_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  FOREIGN KEY (customer_id) REFERENCES customers(id),
  FOREIGN KEY (staff_id) REFERENCES staff(id)
);
```

Execute complete `petshop.sql` script

9.2 Application Configuration

9.2.1 Configuration Files:

- Database connection properties
- Application settings
- Security configurations
- Logging configurations

9.2.2 Connection Configuration:

Listing 9.4: Configuring Database Connection in Java

```java
public class DatabaseConfig {
    public static final String URL = "jdbc:mysql://localhost
       :3306/petshop_db";
    public static final String USER = "root";
    public static final String PASSWORD = "root";
}
```

9.3 Build Process (Maven)

9.3.1 Maven Configuration:

Listing 9.5: Maven POM Configuration

```xml
<project>
  <groupId>com.petshop</groupId>
  <artifactId>petshop-management</artifactId>
  <version>1.0.0</version>
  <dependencies>
    <dependency>
```

```
      <groupId>mysql</groupId>
      <artifactId>mysql-connector</artifactId>
      <version>8.0.33</version>
    </dependency>
    <!-- Additional dependencies -->
  </dependencies>
</project>
```

9.3.2 Build Commands:

- `mvn clean compile` - ***Clean and compile*** source code
- `mvn test` - ***Run*** all unit and integration tests
- `mvn package` - ***Create*** executable `JAR` file
- `mvn install` - ***Install*** artifact to local repository

Chapter 10

User Manual

10.1 System Requirements

10.1.1 Hardware Requirements

- **Processor**: `Intel Core i3` or `AMD equivalent ( 2.0 GHz minimum)`
- **Memory**: `4GB RAM` minimum, `8GB` recommended
- **Storage**: `1 GB` free disk space for application and data
- **Display**: `1024 × 768` minimum resolution
- **Network**: **Internet connection** for updates (optional)

10.1.2 Software Requirements

- **Operating System**: `Windows 10/11, macOS 10.14+, Linux Ubuntu 18.04+`
- **Java Runtime**: `JRE 11` or higher
- **Database**: `MySQL Server 5.7` or higher
- **Additional Software**: `PDF` viewer for invoice viewing

10.2 Installation Guide

Step-by-Step Installation Process:

1. **Java Installation**

   a. Download textbf`JRE 11+` from `Oracle` or `OpenJDK`

   b. Install with **default settings**

   c. Verify: Open *command prompt*, type `java -version`

2. **MySQL Installation**

   a. Download `MySQL Community Server`

   b. Install with secure configuration

   c. Set `root password` and remember it

   d. Start `MySQL service`

3. **Database Setup**

   a. Open `MySQL command line` or `Workbench`

   b. Execute `petshop.sql` script

   c. Verify **all tables** are created successfully

4. **Application Installation**

   a. Extract application files to desired location

   b. Update database connection settings

   c. Run `PetshopApp.jar` file

10.3 User Guide for Manager

Manager System Access and Capabilities:

10.3.1 Login Process:

- **Step 1**: Launch application

- **Step 2**: Enter manager credentials
- **Step 3**: Access full system functionality

10.3.2 Key Manager Functions:

1. **Staff Management**
   a. ***Add*** new `STAFF` members with role assignment
   b. ***Update*** `STAFF` information and `salaries`
   c. ***Remove*** `STAFF` members from system
   d. ***View*** `STAFF` performance `reports`
2. **Financial Reporting**
   a. ***Generate*** `revenue reports` by date range
   b. ***View*** `STAFF` performance metrics
   c. ***Analyze*** `customer` spending patterns
   d. ***Export*** reports to `PDF` format
3. **System Administration**
   a. ***Configure*** system settings
   b. ***Manage*** user access permissions
   c. ***Monitor*** system performance
   d. ***Backup*** and ***maintenance*** procedures

Daily Manager Tasks:

- Review daily sales `reports`
- Monitor inventory levels
- Check `STAFF` performance metrics
- Handle `customer` escalations

10.4 User Guide for Staff

Staff System Access and Daily Operations:

10.4.1 Login Process:

- **Step 1**: Launch application with `STAFF` credentials
- **Step 2**: Access assigned functional areas
- **Step 3**: Limited administrative capabilities

10.4.2 Core Staff Functions:

1. Customer Management

   a. ***Register*** new `customers`

   b. ***Update*** `customers` information

   c. ***View*** `customers` purchase history

   d. ***Manage*** `loyalty point` balances

2. Sales Processing

   a. ***Create*** new `sale transactions`

   b. ***Add*** `pets` and `products` to cart

   c. ***Process*** `payments (cash/card)`

   d. ***Generate*** customer `invoices`

3. Inventory Management

   a. ***Add*** new `pets` to inventory

   b. ***Update*** `pet` information and `status`

   c. ***Manage*** `product` catalog

   d. ***Update*** `stock quantities`

   e. ***Check*** `inventory` levels

Daily Staff Workflow:

- Check inventory levels at start of shift
- Process customer transactions
- Update pet and product information
- Generate end-of-day sales summary

Chapter 11

Maintenance & Future Enhancements

11.1 Known Issues

11.1.1 Performance Issues

- Single *database connection* may cause **bottlenecks** under high load
- Large `dataset queries` may experience slower response times
- `UI` responsiveness decreases with extensive data operations

11.1.2 Functional Limitations

- No automated **backup system** implemented
- Limited **multi-user** concurrent access
- Basic **error logging** without advanced monitoring
- No**email notification system** for low stock alerts

11.1.3 Workarounds

- Regular **manual** database backups recommended
- Limit **concurrent users** during peak operations

- Monitor **application logs** manually for errors
- Implement **external backup** procedures

11.2 Planned Features

11.2.1 Short-term Enhancements (Next 6 months)

- Advanced dashboard with ***real-time analytics***
- Automated email notifications for low stock
- `Barcode scanning` integration for products
- Enhanced reporting with textbfgraphical charts
- Mobile-responsive `web interface`

11.2.2 Medium-term Roadmap (6-12 months)

- Multi-store management capabilities
- Customer mobile application
- Integration with accounting software
- Advanced **inventory forecasting**
- `Cloud deployment` options

11.2.3 Long-term Vision (1-2 years)

- `AI-powered` inventory optimization
- **Customer behavior** analytics
- **Automated** marketing campaigns
- **Supply chain** management integration
- `IoT device` integration for pet monitoring

11.3 System Maintenance

11.3.1 Daily Maintenance

- **Monitor** application performance
- **Check** database connectivity
- **Review** error logs for issues
- **Verify** backup procedures

11.3.2 Weekly Maintenance

- Database **optimization and cleanup**
- Log file **rotation and archival**
- System performance analysis
- User access review

11.3.3 Monthly Maintenance

- Complete `database backup verification`
- `Security updates and patches`
- System performance tuning
- User training and support **review**

11.3.4 Annual Maintenance

- System architecture review
- Security audit and penetration testing
- Hardware and software upgrade planning
- Disaster recovery testing

Chapter 12

Conclusion

The Pet Shop Management System represents a comprehensive solution for pet retail **business operations**. Built with modern `Java` technologies and following industry best practices, the system provides robust functionality for **managing all aspects** of a pet store business.

12.1 Key Achievements

- **Secure**, **role-based** access control system
- **Comprehensive** business logic implementation
- **User-friendly** interface design
- **Scalable** architecture for future growth
- Complete transaction processing **capabilities**

12.2 Business Impact

- **Streamlined** operations and improved **efficiency**
- **Enhanced** customer relationship management
- **Accurate** inventory and financial tracking
- **Professional** invoice generation and reporting

- **Reduced** manual errors and improved data accuracy

12.3 Technical Excellence

- **Clean, maintainable** code architecture
- **Comprehensive** security implementation
- **Robust** error handling and validation
- **Well-documented** system components

The system successfully addresses the core requirements of pet retail management while providing a foundation for future enhancements and scalability. With proper maintenance and continued development, this system will serve as ***a reliable business management platform*** for pet retail operations.

Chapter 13

Appendices

A. Detailed Data Models

A.1 Staff Model (‘staff‘ table)

- `id`: `Integer`, `auto-increment`, `primary key`.
- `name`: `String`, required.
- `email`: `String`, required.
- `username`: `String`, required, unique.
- `password_hash`: `String`, required (hashed using `BCrypt`).
- **role**: `Enum (STAFF, MANAGER)` , values: `MANAGER`, `STAFF`, required.
- `salary`: `Decimal`, default: 0.00.
- *Note*: Password is securely stored using salted `BCrypt` hash. *Role determines access level.*

A.2 Customer Model (‘customers‘ table)

- `id`: `Integer`, `auto-increment`, `primary key`.
- `name`: `String`, required.
- `email`: `String`, required, unique.

- phone: String, required.
- loyalty_points: Integer, default: 0.

A.3 Pet Model (Pets table)

- id: Integer, auto-increment, primary key.
- name: String, required.
- type: Enum (DOG, CAT), required.
- breed: String, required.
- age: Integer, required.
- price: Decimal, required.
- status: Boolean (1 = available), default: 1.

A.4 Product Model (Products table)

- id: Integer, auto-increment, primary key.
- name: String, required.
- price: Decimal, required.
- stock_quantity: Integer, default: 0.
- type:Enum (TOY, FOOD, MEDICINE), required.
- material: String (for toys), optional.
- expiration_date: Date (for food/medicine), optional.
- nutritional_info: Text (for food), optional.
- manufacture_date: Date, optional.
- dosage: String (for medicine), optional.
- status: Boolean, default: 1.

A.5 Bill Model (Bills table)

- `id`: `Integer`, `auto-increment`, `primary key`.
- `customer_id`: Foreign key → `customers(id)`, required.
- `staff_id`: Foreign key → `staff(id)`, required.
- `total_amount`: `Decimal`, required.
- `payment_method`: `Enum (CASH, CARD)`, required.
- `transaction_time`: Timestamp, default: CURRENT_TIMESTAMP.

A.6 BillItem Model (bBill_Items table)

- `id`: `Integer`, `auto-increment`, `primary key`.
- `bill_id`: Foreign key → `bills(id)`, required.
- `item_type`: `Enum (PET, PRODUCT, MEDICINE)`, required.
- `pet_id`: `Integer`, optional.
- `product_id`: `Integer`, optional.
- `quantity`: `Integer`, required.
- `unit_price`: `Decimal`, required.

B. Sample SQL Data Inserts

B.1 Sample Staff

```sql
INSERT INTO staff (name, email, phone, username, password_hash,
    role, salary)
VALUES
('Vy Manager', 'Vy@gmail.com', '0123456789', 'vy', 'hashed_pwd1',
     'MANAGER', 1500.00),
('Vu Staff', 'Vu@gmail.com', '0987654321', 'vu', 'hashed_pwd2', '
    STAFF', 800.00);
```

B.2 Sample Customers

```sql
INSERT INTO customers (name, email, phone, loyalty_points)
VALUES
('Quan Le', 'Quan@gmail.com', '0901234567', 15),
('Thong Nguyen', 'Thong@gmail.com', '0912345678', 30);
```

B.3 Sample Pets

```sql
INSERT INTO pets (name, type, breed, age, price)
VALUES
('Max', 'DOG', 'Labrador', 3, 700.00),
('Luna', 'CAT', 'Maine Coon', 2, 500.00);
```

B.4 Sample Products

```sql
INSERT INTO products (name, price, stock_quantity, type)
VALUES
('Dog Food - Premium', 50.00, 40, 'FOOD'),
('Cat Toy - Laser Pointer', 12.00, 75, 'TOY');
```

C. API Documentation

C.1 AuthController

- `login(String username, String password)`: Verifies user and starts session.
- `isManager()`: Checks if current session user has `MANAGER` role.
- `logout()`: Ends user session.

C.2 CustomerService

- `addCustomer(Customer customer)`: Validates and saves `customer`.
- `updateCustomer(Customer customer)`: Modifies `customer` information.

- `getCustomerById(int id)`: Retrieves a `customer` by ID.

C.3 BillingController

- `addProductToCart(Product p, int quantity)`: Adds `product` to current cart.
- `applyLoyaltyPoints(Customer c, BigDecimal total)`: Applies points based on total.
- `processBill(Bill bill)`: Validates `stock`, finalizes transaction.
- `exportBillAsPdf(Bill bill)`: Generates `PDF invoice`.

C.4 StaffController (Manager Only)

- `addStaff(Staff s)`: Creates new `STAFF` member.
- `deleteStaff(int id)`: Removes `STAFF` from system.
- `getRevenueByStaff(int id)`: Revenue summary for individual `STAFF`.

D. Error Code Reference

Code	**Description**	**Cause or Context**
1001	Invalid credentials	Login failure due to wrong username/password
1002	Access denied	Staff attempting manager-only action
2001	Missing required fields	Input validation failed
3001	Insufficient stock	Product quantity requested exceeds availability
3002	Pet unavailable	Pet already sold or marked inactive
4001	Database error	Failed query, table issue, or disconnected DB
5001	Internal application error	Uncaught exceptions, logic errors