PROJECT REPORT
Veterinary Appointment System

Ankara University, Computer Engineering Department
May 26, 2025

Contributors
Ahmet ATAR — 22290230
Yiğit GÜLBEYAZ — 22290725
Onur Yiğit KOCATÜRK — 22290617
Ömerfaruk SARIBAL — 22290042

Introduction

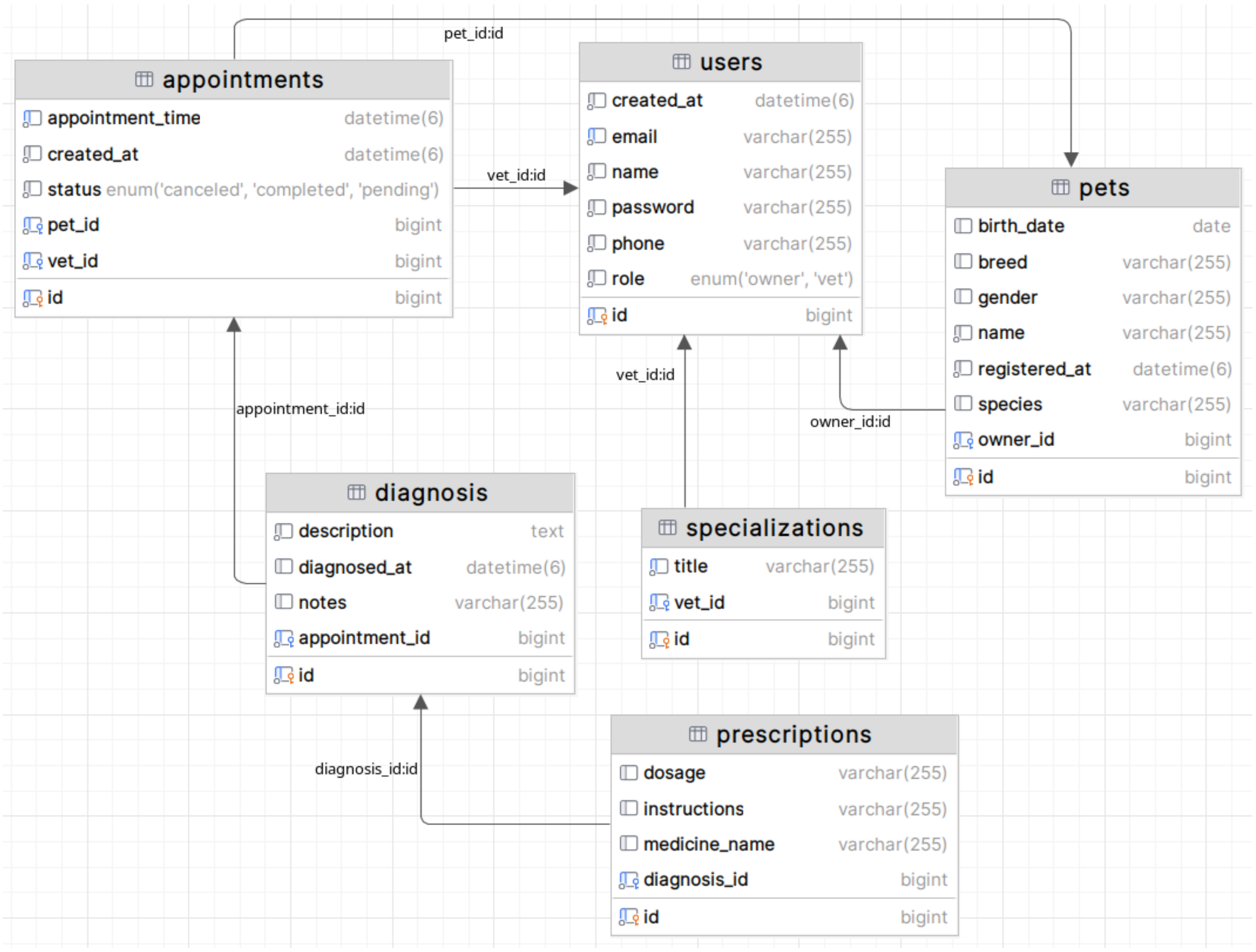
The Veterinary Appointment System is a relational database-backed software solution designed to streamline operations in veterinary clinics. It addresses the need for a structured and digitized workflow where pet owners can register their animals, schedule appointments with veterinarians, and access medical history and prescriptions with ease.

The system targets two main user roles: Pet Owners and Veterinarians. Pet Owners are able to manage pet profiles and initiate appointments, while Veterinarians can track their schedules, add diagnosis, and issue prescriptions based on clinical evaluations. The system ensures data integrity, role-based access, and a clean separation of responsibilities through a well-designed relational schema.

The backend of the system is built using Java (Spring Boot) with MySQL as the underlying relational database, managed via Docker Compose for consistent deployment. The frontend is implemented as an iOS mobile application using Swift and UIKit, which interacts with the backend through a RESTful API.

This project demonstrates the application of fundamental DBMS concepts such as relational modeling, entity relationships, normalization, indexing strategies, and integrity constraints, all within the context of a real-world use case.

Relational Schema of the System



Generated SQL Queries (via Spring Data JPA)

During the initialization of the application, Hibernate automatically generates the database schema based on the annotated JPA entity definitions. The following SQL statements were produced and executed to construct the relational schema for the Veterinary Appointment System:

```
CREATE TABLE users (
    id BIGINT NOT NULL AUTO_INCREMENT,
    email VARCHAR(255) NOT NULL,
    name VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    phone VARCHAR(255) NOT NULL,
    role ENUM ('OWNER', 'VET') NOT NULL,
    PRIMARY KEY (id)
```

```sql
);

CREATE TABLE pets (
    id BIGINT NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    species VARCHAR(255),
    breed VARCHAR(255),
    gender VARCHAR(255),
    birth_date DATE,
    owner_id BIGINT NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE appointments (
    id BIGINT NOT NULL AUTO_INCREMENT,
    appointment_time DATETIME(6) NOT NULL,
    status ENUM ('CANCELED', 'COMPLETED', 'PENDING') NOT NULL,
    pet_id BIGINT NOT NULL,
    vet_id BIGINT NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE diagnosis (
    id BIGINT NOT NULL AUTO_INCREMENT,
    description TEXT NOT NULL,
    notes VARCHAR(255),
    diagnosed_at DATETIME(6),
    appointment_id BIGINT NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE prescriptions (
    id BIGINT NOT NULL AUTO_INCREMENT,
    medicine_name VARCHAR(255),
    dosage VARCHAR(255),
    instructions VARCHAR(255),
    diagnosis_id BIGINT NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE specializations (
    id BIGINT NOT NULL AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    vet_id BIGINT NOT NULL,
    PRIMARY KEY (id)
);

CREATE INDEX idx_user_email ON users (email);
CREATE INDEX idx_pet_owner ON pets (owner_id);
CREATE INDEX idx_appointment_pet ON appointments (pet_id);
CREATE INDEX idx_appointment_vet ON appointments (vet_id);
CREATE INDEX idx_appointment_time ON appointments (appointment_time);
CREATE INDEX idx_prescription_diagnosis ON prescriptions (diagnosis_id);
CREATE INDEX idx_specialization_vet ON specializations (vet_id);

ALTER TABLE users
    ADD CONSTRAINT uq_user_email UNIQUE (email);

ALTER TABLE diagnosis
    ADD CONSTRAINT uq_diagnosis_appointment UNIQUE (appointment_id),
   ADD CONSTRAINT FK_diagnosis_appointment FOREIGN KEY (appointment_id)
REFERENCES appointments (id);

ALTER TABLE pets
    ADD CONSTRAINT FK_pet_owner FOREIGN KEY (owner_id) REFERENCES users (id);

ALTER TABLE appointments
    ADD CONSTRAINT FK_appointment_pet FOREIGN KEY (pet_id) REFERENCES pets (id),
    ADD CONSTRAINT FK_appointment_vet FOREIGN KEY (vet_id) REFERENCES users (id);

ALTER TABLE prescriptions
    ADD CONSTRAINT FK_prescription_diagnosis FOREIGN KEY (diagnosis_id)
REFERENCES diagnosis (id);

ALTER TABLE specializations
    ADD CONSTRAINT FK_specialization_vet FOREIGN KEY (vet_id) REFERENCES users (id);
```

Schema Characteristics and Design Justification

The automatically generated schema demonstrates a robust and normalized relational design. Below is a detailed breakdown of the schema's structural strengths and DBMS principles applied:

1. Use of Primary Keys for Entity Uniqueness

- Every table contains an `id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY.`
- Ensures each record has a unique identifier and enables efficient indexing and lookup.
- Follows First Normal Form (1NF) by uniquely identifying each row.

2. Strong Use of Foreign Key Constraints

- Foreign key constraints maintain referential integrity between:
    - `pets.owner_id → users.id`
    - `appointments.pet_id → pets.id`
    - `appointments.vet_id → users.id`
    - `diagnosis.appointment_id → appointments.id`
    - `prescriptions.diagnosis_id → diagnosis.id`
    - `specializations.vet_id → users.id`
- Prevents insertion of inconsistent or orphaned data.

3. One-to-One Relationship Enforcement

- The constraint `uq_diagnosis_appointment` enforces a 1:1 relationship between `appointments` and `diagnosis`.
- Ensures a single diagnosis per appointment, reflecting real-world constraints.

4. Normalization of Entity Structures

- Tables are well-normalized up to 3NF:
    - No redundant columns.
    - Each non-key attribute depends only on the primary key.
    - Relationships (e.g., pets, diagnosis, appointments) are separated across dedicated tables.

5. ENUM Columns for Controlled Data Domains

- ENUMs used for `users.role ('OWNER', 'VET')` and `appointments.status ('CANCELED', 'COMPLETED', 'PENDING')` restrict values.
- Ensures data consistency, avoids invalid states, and facilitates safer query logic.

6. Strategic Indexing for Query Optimization

- Indexes added to commonly queried fields:
    - `users.email`
    - `appointments.pet_id, appointments.vet_id, appointments.appointment_time`
    - `prescriptions.diagnosis_id`
    - `specializations.vet_id`
- Speeds up `JOIN`, `WHERE`, and `ORDER BY` operations, particularly on foreign keys and frequently filtered columns.

7. Unique Constraint on Email

- `uq_user_email` ensures each email in the `users` table is unique.
- Critical for login systems and identity management.

8. Use of DATETIME(6) for High-Precision Timestamps

- `appointments.appointment_time` and `diagnosis.diagnosed_at` use `DATETIME(6)` for microsecond-level accuracy.
- Useful in medical systems where exact timing matters (e.g., audit logs or appointment conflicts).

9. Logical Table Design with Domain Semantics

- Tables are semantically well-separated:
  - `pets` do not mix with `appointments`
  - `diagnosis` is not stored directly in `appointments`
  - `prescriptions` are tied strictly to `diagnosis`, not directly to `appointments`
- This separation enhances modularity, testability, and maintainability of the schema.

10. Support for Specializations in a Separate Table

- `specializations` table maps veterinarians to multiple areas of expertise (e.g., orthopedics, nutrition).
- Implies potential 1:N or M:N relationship scalability, even if currently modeled as 1:N.

11. Enforcing Composite Uniqueness on Diagnosis Table

The constraint:

```
ADD CONSTRAINT uq_diagnosis_appointment UNIQUE (appointment_id)
```

- effectively ensures a 1:1 mapping between `appointments` and `diagnosis`.
- While this is not a composite primary key, it serves as a composite uniqueness constraint in conjunction with the `id` primary key.
- This design:
  - Allows the use of a surrogate primary key (`id`)
  - While still enforcing business-level uniqueness (one diagnosis per appointment)
- Similar strategies could be extended in the future (e.g., ensuring a vet cannot have two identical specializations).

Entity Descriptions & Data Dictionary

Table Name	Description
`users`	Stores information about both pet owners and veterinarians. Differentiation is made using the `role` ENUM (`OWNER`, `VET`).
`pets`	Represents pets owned by users. Each pet is linked to an owner via `owner_id`.
`appointments`	Captures scheduled appointments between pets and vets, including status and time.
`diagnosis`	Holds diagnostic information issued by a veterinarian for a specific appointment. One-to-one with `appointments`.
`prescription`	Contains prescribed medication data, dosage, and instructions, linked to a diagnosis.
`specialization`	Lists the special fields of expertise for each veterinarian (e.g., surgery, dermatology).

Each table includes relevant constraints such as foreign keys and unique constraints to ensure referential integrity and logical consistency across the system.

Application Architecture Overview

The Veterinary Appointment System is composed of the following major components:

- **Backend:**
  - Language: Java 17
  - Frameworks: Spring Boot, Spring Data JPA
  - Database: MySQL (Dockerized)
  - DevOps: Docker Compose for environment management
- **Frontend:**
  - Language: Swift
    Framework: UIKit
  - Architecture: MVVM
- **Data Flow:**
  - The iOS app communicates with the REST API to perform CRUD operations.
  - The Spring Boot backend handles business logic and interacts with the MySQL database through Hibernate ORM.
- **Deployment Architecture (Simplified):**

  [Swift iOS App] ⇄ [REST API – Spring Boot] ⇄ [MySQL DB in Docker]

Sample Use Case Flows

Below are selected functional use cases that describe typical user interactions with the system:

a) Pet Owner Registers and Adds Pet

- The user signs up and selects role `OWNER`.
- Adds one or more pets through a simple form (name, species, birth date, etc.).

b) Pet Owner Books an Appointment

- Selects a pet and a veterinarian.
- Chooses available date and time.
- System creates an appointment with status `PENDING`.

c) Veterinarian Confirms Appointment and Adds Diagnosis

- Vet sees upcoming appointments.
- After examination, adds a diagnosis linked to the appointment.
- The appointment status is updated to `COMPLETED`.

d) Veterinarian Issues Prescription

- From the diagnosis view, the vet creates a prescription.
- Includes medicine name, dosage, and instructions.

These flows are implemented through coordinated API endpoints and reflect transactional consistency across the underlying tables.

Containerization and Deployment Strategy

To ensure consistent, reproducible, and platform-independent deployment of the Veterinary Appointment System, the entire backend infrastructure was containerized using Docker. This containerization encapsulates the Spring Boot application and its required MySQL database within isolated, yet interconnected environments. The orchestration and lifecycle management of these containers are handled by Docker Compose, which simplifies multi-service setups for development and production alike.

Backend Container (Spring Boot)

The backend application is built using a multi-stage `Dockerfile`. In the first stage, Maven is used to compile and package the application into a standalone `.jar` file. In the second stage, a lightweight JDK image is used to run the packaged application efficiently. This results in a minimal and optimized runtime image, reducing both build time and attack surface.

The backend container:

- Exposes port `8080` for API access.
- Reads database connection details from injected environment variables (e.g., `SPRING_DATASOURCE_URL`).
- Waits for the database service to become available before starting (using startup delay or optional wait scripts).

Database Container (MySQL)

The MySQL container is configured with:

- A predefined root password and application-specific user credentials.
- A mounted volume to persist data across container restarts.
- Port `3307` exposed to the host for optional debugging or inspection.

Networking and Isolation

Both services are attached to a custom Docker bridge network (`vet_network`), allowing them to communicate using container names as hostnames (e.g., the backend connects to the database via `jdbc:mysql://vet_mysql:3306/vet_db`). This avoids hardcoding IP addresses and enhances portability across systems.

Compose-Orchestrated Deployment

The `docker-compose.yml` file defines and coordinates the lifecycle of both services. A simple command:

```
docker compose up --build
```

initializes the entire backend infrastructure from scratch. This one-line setup:

- Builds the Spring Boot application if needed.
- Launches the MySQL service and waits for readiness.
- Starts the backend container in a reliable, repeatable manner.

Benefits of This Architecture

- **Portability**: The application can run identically across different developer machines and deployment servers.
- **Isolation**: The backend and database run in isolated environments, eliminating cross-dependency issues.
- **Version control**: Images and configurations are explicitly versioned and documented within the repository.
- **Optimization**: Multi-stage builds and runtime minimization reduce image size and boot time.

This deployment strategy demonstrates modern DevOps practices, reinforcing the system's robustness, maintainability, and alignment with production-ready software engineering standards.

Appendix A: Source Code Repositories

A.1 Backend Repository

- **Title:** Veterinary Appointment System - Backend
- **Technologies Used:** Java 17, Spring Boot, Spring Data JPA, MySQL, Hibernate, Docker
- **Repository URL:** https://github.com/CAPELLAX02/vet-apt-sys

This repository contains the complete backend implementation of the system. It includes RESTful endpoints for user registration, appointment management, diagnosis and prescription handling. The system uses Hibernate ORM for database interaction and is fully containerized using Docker Compose.

A.2 iOS Application Repository

- **Title:** Veterinary Appointment System - iOS App
- **Technologies Used:** Swift, UIKit, URLSession, MVVM
- **Repository URL:** https://github.com/omerfaruksaribal/VetApp

This repository provides the iOS mobile application that allows pet owners and veterinarians to interact with the backend system. The app includes user registration, pet management, appointment booking, and diagnosis viewing functionalities. It communicates with the backend via secure RESTful APIs.