Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Інститут комп’ютерних систем

Кафедра інформаційних систем

Курсова робота

З дисципліни: «Об’єктно-орієнтоване програмування»

Варіант №20

Виконала:

Студентка групи АІ-232

Сидорова Т.С.

Перевірили:

Годовіченко М.А.

Одеса 2025

**ЗМІСТ**

[**ВСТУП**](#_Toc200882173) 3

[**АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**](#_Toc200882174) 5

[**ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**](#_Toc200882179) 8

[3.1 Проєктування структури даних 8](#_Toc200882180)

[3.2 Опис архітектури застосунку 10](#_Toc200882181)

[3.3 REST API 11](#_Toc200882183)

[**РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ** 14](#_Toc200882185)

[4.1 Моделі 14](#_Toc200882186)

[4.2 Репозиторії 20](#_Toc200882188)

[4.3 Сервіси 19](#_Toc200882189)

[4.4 Контролери 20](#_Toc200882190)

[4.5 Конфігурація 22](#_Toc200882191)

[4.6 Конфігурація, підключення до бази даних та хостинг 24](#_Toc200882197)

[**ТЕСТУВАННЯ** 28](#_Toc200882198)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 30](#_Toc200882205)

ВСТУП

У сучасному світі роль домашніх тварин у житті людей суттєво зросла — вони стали частиною родини, джерелом емоційної підтримки та об’єктом турботи. З цим зростає і потреба у якісному ветеринарному обслуговуванні. Проте багато клінік досі ведуть облік пацієнтів вручну або в неструктурованих таблицях, що ускладнює доступ до медичних даних та знижує ефективність роботи персоналу.

Актуальність проєкту полягає в необхідності створення зручної, надійної та безпечної системи обліку домашніх тварин, яка дозволить автоматизувати процеси зберігання й обробки даних про тварин, їхніх власників, візити, процедури та призначення.

Мета курсової роботи — розробити веб-застосунок для ветеринарної клініки з можливістю повноцінного обліку пацієнтів, візитів, процедур і призначень.

Основними завданнями даної курсової роботи є проведення детального аналізу предметної області з метою визначення ключових сутностей та їх взаємозв’язків. Далі було необхідно спроєктувати архітектуру застосунку разом із структурою бази даних, що дозволяє ефективно зберігати та опрацьовувати інформацію. Наступним кроком стало створення REST API з повною підтримкою CRUD-операцій для забезпечення можливості взаємодії користувачів із системою через вебінтерфейс. Особлива увага приділялася питанням безпеки, тому у проєкті реалізовано механізми автентифікації користувачів за допомогою JWT та OAuth2. Наприкінці було проведено комплексне тестування функціональності застосунку та його розгортання у хмарному середовищі для практичного використання.

Система реалізована мовою Java з використанням фреймворку Spring Boot. Для взаємодії з базою даних застосовується Spring Data JPA, а для збереження даних — PostgreSQL. Безпека забезпечується через JWT та інтеграцію з OAuth2. Проєкт розгорнуто у хмарному середовищі на платформі Render.

Розроблений застосунок дозволяє зручно керувати інформацією про тварин, відстежувати історію лікування, формувати статистику та спрощує повсякденну роботу ветеринарного персоналу.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Сфера ветеринарних послуг сьогодні активно розвивається, адже домашні улюбленці стали повноправними членами багатьох родин. Турбота про їхнє здоров’я вимагає не лише професіоналізму ветеринарів, а й якісного ведення обліку пацієнтів, історій візитів, призначень та процедур. У невеликих клініках це часто робиться вручну або в Excel, що з часом породжує безлад, втрату даних та знижує ефективність роботи персоналу.

Актуальність автоматизованої системи обліку тварин полягає в тому, що вона дозволяє централізовано керувати інформацією про власників, тварин, історію лікування, процедури та призначення, що суттєво спрощує роботу клініки, підвищує швидкість обслуговування клієнтів і дає змогу аналізувати зібрані дані для покращення сервісу.

Головною метою курсової роботи є створення сучасного веб застосунку, який реалізує повноцінний цикл обліку домашніх тварин у ветеринарній клініці. Система повинна охоплювати всі ключові процеси — від реєстрації власника до фінансової статистики витрат по кожній тварині. Особливу увагу приділяється гнучкості та масштабованості рішення: система має бути зручною, надійною, а головне — придатною для подальшого розвитку, наприклад, у напрямку додавання мобільного застосунку чи інтеграції з електронною чергою.

У процесі моделювання предметної області було виявлено кілька ключових сутностей, які формують основу системи:

* власник (owner) — фізична особа, яка є клієнтом клініки. Може мати одну

або декілька тварин. У записі зберігається ім’я та контактний номер;

* тварина (pet) — пацієнт ветеринарної клініки. Має унікальне ім’я, вид

(наприклад, кіт, собака, хом’як), дату народження та обов’язково прив’язується до свого власника;

* візит (visit) — окрема подія звернення тварини до клініки. Фіксується дата

та причина звернення, наприклад: “планова вакцинація” або “підозра на отруєння”;

* процедура (procedure) — медична маніпуляція, яка проводиться під час

візиту (наприклад, стерилізація, обробка ран, зняття швів тощо). Кожна процедура має назву та вартість;

* призначення (prescription) — фармакологічне лікування, рекомендоване

ветеринаром, із зазначенням медикаменту та дозування.

Зв'язки між цими сутностями:

* один власник може мати декількох тварин (зв’язок один до багатьох);
* одна тварина може мати декілька візитів;
* один візит може містити декілька процедур та декілька призначень.

У процесі створення вебзастосунку особливу увагу було приділено моделюванню поведінки користувача та розумінню типових сценаріїв його взаємодії із системою. Це дозволило сформувати не лише логіку функціонування програмного продукту, а й забезпечити зручність, ефективність і передбачуваність його використання в реальних умовах ветеринарної клініки.

Уявімо типову ситуацію, яка виникає у щоденній практиці закладу. До клініки звертається новий клієнт із домашнім улюбленцем. На етапі реєстрації адміністратор вводить основні дані про власника: ім’я та контактний номер телефону. Після цього створюється запис про тварину, що включає її ім’я, вид (наприклад, кішка, собака, папуга), дату народження та вказується зв’язок із відповідним власником. Таким чином, уже на стартовому рівні система формує пов’язані між собою сутності, які в подальшому забезпечують контекст і цілісність усіх наступних дій.

Коли тварина приходить на черговий візит до ветеринара, працівник клініки створює новий запис візиту, у якому зазначає дату та причину звернення. Під час візиту можуть бути проведені процедури — наприклад, ультразвукове дослідження, вакцинація чи чищення зубів — які фіксуються окремо із зазначенням назви та вартості. Якщо лікар вважає за потрібне призначити лікування, він додає відповідне призначення до того ж візиту, де вказується назва препарату та рекомендоване дозування.

Після завершення візиту адміністратор або лікар має змогу переглянути історію звернень цієї тварини, включно з усіма процедурами та призначеннями, що були виконані раніше. Такий підхід дозволяє сформувати повноцінну медичну картку пацієнта, яка зберігається в цифровому вигляді і доступна для пошуку чи аналітики у будь-який момент.

Важливо, що система передбачає як одноразову взаємодію з окремими елементами (наприклад, швидке додавання нового власника), так і більш комплексні дії, що вимагають обробки зв’язаних даних. Наприклад, під час перегляду візитів тварини працівник може миттєво побачити не лише загальну інформацію, а й усі проведені процедури, призначені препарати, а також сумарні витрати. Такі можливості суттєво спрощують роботу як для адміністративного персоналу, так і для лікарів, зменшуючи ризики помилок та дублювання даних.

Система також реалізує аналітичну функцію, яка дає змогу збирати зведену інформацію про діяльність клініки. Наприклад, можна згенерувати звіт про найбільш популярні процедури за певний період, або визначити, які види тварин найчастіше відвідують заклад. Крім того, існує можливість вивести статистику витрат на лікування по кожному пацієнту, що дозволяє формувати персоналізовані пропозиції або гнучко управляти ціноутворенням послуг.

Уся взаємодія із системою відбувається через REST API, що дає можливість у майбутньому інтегрувати інтерфейс для користувачів або мобільний застосунок. Але вже на цьому етапі вебсервіс повністю підтримує основні сценарії використання, зокрема створення, редагування, перегляд та видалення даних щодо власників, тварин, візитів, процедур і призначень.

ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проєктування структури даних

Проєктування структури даних є ключовим етапом створення інформаційної системи для ветеринарної клініки, оскільки воно визначає логіку збереження, обробки та взаємодії об’єктів предметної області. У контексті системи для обліку домашніх тварин було критично важливо змоделювати сутності, які відображають реальні процеси ветеринарної практики, враховуючи їх взаємозалежність та сценарії використання.

Перед початком кодування я ретельно проаналізувала предметну область і виділила основні сутності: власники тварин (Owner), тварини (Pet), візити до клініки (Visit), медичні процедури (Procedure), лікарські призначення (Prescription) та користувачі системи (User) для управління автентифікацією. Ці сутності пов’язані між собою, що вимагало ретельного моделювання для забезпечення цілісності та ефективності системи.

Для збереження даних обрано реляційний підхід із використанням PostgreSQL, який забезпечує узгодженість, нормалізацію та підтримку складних зв’язків. Кожна сутність реалізована як окремий клас Java, позначений анотацією @Entity для роботи з JPA (Java Persistence API). Щоб зменшити кількість шаблонного коду (гетерів, сетерів, конструкторів, методів toString()), використано бібліотеку Lombok із анотаціями @Data, @NoArgsConstructor, @AllArgsConstructor, що спрощує підтримку та читабельність коду.

Особливу увагу приділено моделюванню зв’язків між сутностями:

- Owner-Pet: Один власник може мати кілька тварин, що реалізовано через зв’язок @OneToMany у класі Owner і @ManyToOne у класі Pet. Це дозволяє навігацію від власника до його тварин і навпаки.

- Pet-Visit: Кожна тварина може мати кілька візитів, змодельованих як @OneToMany у Pet і @ManyToOne у Visit.

- Visit-Procedure/Prescription: Кожен візит може включати кілька процедур і призначень, що реалізовано через @OneToMany у Visit і @ManyToOne у Procedure та Prescription.

- User: Сутність для автентифікації, яка не має прямих зв’язків із іншими сутностями, але використовується для управління доступом.

Для автоматичного поширення операцій (наприклад, видалення) використано каскадні операції з параметрами cascade = CascadeType.ALL і orphanRemoval = true у @OneToMany. Наприклад, видалення візиту автоматично видаляє пов’язані процедури та призначення. Анотація @JoinColumn використовується для чіткого визначення зовнішніх ключів у дочірніх таблицях, що запобігає помилкам при генерації схеми бази даних.

Кожна сутність спроєктована з урахуванням:

- унікальності ідентифікаторів:

Використано @Id і @GeneratedValue(strategy = GenerationType.IDENTITY) для автоматичної генерації унікальних ідентифікаторів.

- типізації даних:

Наприклад, LocalDate для дат народження тварин (Pet.birthDate) і візитів (Visit.date), BigDecimal для вартості процедур (Procedure.cost), String для текстових полів, таких як Prescription.dosage.

- реального контексту:

Поля, як-от Prescription.dosage і Procedure.cost, відображають вимоги ветеринарної практики для точного обліку історії лікування та витрат.

Структура даних відображає реальні процеси ветеринарної клініки. Наприклад, Prescription містить поле dosage для точного збереження дозування ліків, а Procedure включає cost для фінансового обліку. Це забезпечує семантичну точність і практичну цінність системи.

Для майбутнього масштабування передбачено можливість додавання нових сутностей, таких як Veterinarian, ClinicRoom або Invoice, без порушення цілісності моделі. Наприклад, можна додати зв’язок @ManyToOne між Visit і Veterinarian для збереження інформації про лікаря, який проводив візит.

3.2 Опис архітектури застосунку

Архітектура застосунку побудована з урахуванням принципів багатошарової архітектури та шаблону MVC (Model-View-Controller) у контексті бекенд-розробки, що забезпечує чітке розділення обов’язків між компонентами. Це сприяє масштабованій, гнучкій і тестованій системі, яка відповідає вимогам Spring Boot 3.1.0.

Шари архітектури:

- Контролери (Controller):

Відповідають за обробку HTTP-запитів від клієнтів (наприклад, Postman або веб-додатків). Приймають параметри через URI або тіло запиту, валідують їх за допомогою @Valid і передають до сервісів. Наприклад, контролери обробляють запити на створення власників (POST /api/owners) або візитів (POST /api/pets/{petId}/visits), повертаючи відповіді у форматі JSON. Використовують анотації @RestController, @RequestMapping, @PostMapping, тощо.

- Сервіси (Service):

Реалізують бізнес-логіку, таку як створення сутностей, валідація даних і обробка помилок. Ізолюють контролери від деталей доступу до даних, забезпечуючи принцип слабкого зв’язку (low coupling). Наприклад, OwnerService обробляє створення власників, взаємодіючи з OwnerRepository. Позначені анотацією @Service і використовують ін’єкцію залежностей через @Autowired або @RequiredArgsConstructor.

- Репозиторії (Repository):

Забезпечують доступ до даних через Spring Data JPA, автоматично генеруючи SQL-запити для стандартних операцій (наприклад, findById у OwnerRepository). Для складних запитів використовуються анотації @Query з JPQL або native SQL. Позначені анотацією @Repository і розширюють JpaRepository.

Використання DTO

Для передачі даних між шарами використано DTO-класи (Data Transfer Object), такі як OwnerDTO, PetDTO, VisitDTO, ProcedureDTO і PrescriptionDTO. Вони запобігають прямій взаємодії контролерів із сутностями (@Entity), захищаючи цілісність моделі, містять анотації валідації (@NotBlank, @NotNull) для перевірки вхідних даних, спрощують форматування JSON у запитах і відповідях.

Ін’єкція залежностей

Spring Framework автоматично керує залежностями через @Autowired, @Service, @Repository і @RestController. Це зменшує зв’язність коду та дозволяє легко замінювати реалізації, наприклад, змінити базу даних.

Структура пакетів

Проєкт організовано за принципом feature-based structure:

example.petclinic.model: Сутності (User, Owner, Pet, Visit, Procedure, Prescription).

example.petclinic.repository: Репозиторії для доступу до даних.

example.petclinic.service: Сервіси з бізнес-логікою.

example.petclinic.controller: Контролери для обробки HTTP-запитів.

example.petclinic.dto: DTO-класи для передачі даних.

Такий підхід підвищує читабельність і полегшує командну роботу та масштабування.

REST-принципи

Усі запити обробляються через HTTP з дотриманням REST-принципів:

Використання методів GET, POST, PUT, DELETE для CRUD-операцій. HTTP-статуси: 200 OK, 201 Created, 400 Bad Request, 404 Not Found, 500 Internal Server Error. Логічні URL: /api/owners, /api/pets/{id}/visits, /api/visits/{id}/procedures.

Архітектура відповідає принципам чистої архітектури (Clean Architecture), забезпечуючи підтримуваність, тестованість і незалежність від фреймворків. Код є зрозумілим для інших розробників, що полегшує подальший розвиток.

3.3 Опис REST API

REST API є основним інтерфейсом взаємодії з серверною частиною системи ветеринарної клініки. Воно розроблене з дотриманням принципів REST, включаючи множинне іменування ресурсів, відповідні HTTP-методи, передбачувані URL і коректні статус-коди HTTP. API забезпечує інтуїтивну взаємодію для клієнтів і підтримує повне CRUD-управління сутностями (Owner, Pet, Visit, Procedure, Prescription) та автентифікацію користувачів.

Основні запити

- Автентифікація (User):

POST /api/auth/register: Реєстрація нового користувача. Очікує JSON із username, password, role. Повертає об’єкт із кодом 201.

POST /api/auth/login: Автентифікація користувача. Очікує JSON із username, password. Повертає токен чи повідомлення (код 200).

- Власники (Owner):

POST /api/owners: Створює нового власника. Очікує JSON із name, phone. Повертає створений об’єкт із ID (код 201).

GET /api/owners: Повертає список усіх власників (масив JSON, код 200).

PUT /api/owners/{id}: Оновлює власника за ID.

DELETE /api/owners/{id}: Видаляє власника за ID (код 204).

- Тварини (Pet):

POST /api/owners/{ownerId}/pets: Додає нову тварину для власника. Очікує JSON із name, species, birthDate, ownerId. Повертає створений об’єкт (код 201).

GET /api/owners/{ownerId}/pets: Повертає список тварин власника (код 200).

PUT /api/pets/{petId}: Оновлює дані тварини за ID.

DELETE /api/pets/{petId}: Видаляє тварину за ID (код 204).

- Візити (Visit):

POST /api/pets/{petId}/visits: Створює новий візит для тварини. Очікує JSON із petId, date, reason. Повертає створений об’єкт (код 201).

GET /api/pets/{petId}/visits: Повертає всі візити тварини (код 200).

PUT /api/visits/{visitId}: Оновлює візит за ID.

DELETE /api/visits/{visitId}: Видаляє візит за ID (код 204).

- Процедура (Procedure):

POST /api/visits/{visitId}/procedures: Додає процедуру до візиту. Очікує JSON із visitId, name, cost. Повертає створений об’єкт (код 201).

GET /api/visits/{visitId}/procedures: Повертає список процедур візиту (код 200).

PUT /api/procedures/{id}: Оновлює процедуру за ID.

DELETE /api/procedures/{id}: Видаляє процедуру за ID (код 204).

- Призначення (Prescription):

POST /api/visits/{visitId}/prescriptions: Додає призначення до візиту. Очікує JSON із visitId, medication, dosage. Повертає створений об’єкт (код 201).

GET /api/visits/{visitId}/prescriptions: Повертає список призначень візиту (код 200).

PUT /api/prescriptions/{id}: Оновлює призначення за ID.

DELETE /api/prescriptions/{id}: Видаляє призначення за ID (код 204).

- Аналітичні запити

GET /api/pets/{petId}/expenses: Повертає сумарні витрати на процедури для тварини (код 200).

GET /api/visits/by-date?date=2025-05-01: Повертає всі візити за вказану дату (код 200).

GET /api/procedures/popular: Повертає найпопулярніші процедури (код 200).

GET /api/statistics/species: Формує статистику за видами тварин (код 200).

GET /api/pets/{petId}/last-visit: Повертає останній візит тварини (код 200).

Валідація та статуси

Вхідні дані валідуються через DTO з анотаціями @NotBlank, @NotNull.

Успішні операції повертають коди 200 OK або 201 Created. Помилки повертають 400 Bad Request, 404 Not Found або 500 Internal Server Error.

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

Реалізація інформаційної системи для ветеринарної клініки здійснювалася з урахуванням попередньо спроєктованої архітектури, структури даних і REST API. Я дотримувалася підходу розділення відповідальностей за шаблоном Controller-Service-Repository, що забезпечує чисту архітектуру. Особливу увагу приділено зручності масштабування, підтримці коду та його зрозумілості.

4.1 Моделі (Entity-класи, DTO)

Моделі відображають об’єкти предметної області та забезпечують зв’язок із базою даних. Використано об’єктно-орієнтований підхід із JPA для роботи з PostgreSQL і Lombok для зменшення шаблонного коду.

* Entity-класи

1. User

package example.petclinic.model;

import jakarta.persistence.\*;

import lombok.Data;

@Entity

@Table(name = "users")

@Data

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(unique = true, nullable = false)

private String username;

@Column(nullable = false)

private String password;

@Column(nullable = false)

private String role;

}

Поля: id, username, password, role.

Особливості: Використовується для автентифікації. Поле username унікальне.

1. Owner

package example.petclinic.model;

import jakarta.persistence.\*;

import lombok.Data;

import java.util.ArrayList;

import java.util.List;

@Entity

@Table(name = "owners")

@Data

public class Owner {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@Column(nullable = false)

private String phone;

@OneToMany(mappedBy = "owner", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Pet> pets = new ArrayList<>();

}

Поля: id, name, phone, pets.

Зв’язки: @OneToMany із Pet.

1. Pet

package example.petclinic.model;

import jakarta.persistence.\*;

import lombok.Data;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.List;

@Entity

@Table(name = "pets")

@Data

public class Pet {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@Column(nullable = false)

private String species;

private LocalDate birthDate;

@ManyToOne

@JoinColumn(name = "owner\_id", nullable = false)

private Owner owner;

@OneToMany(mappedBy = "pet", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Visit> visits = new ArrayList<>();

}

Поля: id, name, species, birthDate, owner, visits.

Зв’язки: @ManyToOne із Owner, @OneToMany із Visit.

1. Visit

package example.petclinic.model;

import jakarta.persistence.\*;

import lombok.Data;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.List;

@Entity

@Table(name = "visits")

@Data

public class Visit {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private LocalDate date;

@Column(nullable = false)

private String reason;

@ManyToOne

@JoinColumn(name = "pet\_id", nullable = false)

private Pet pet;

@OneToMany(mappedBy = "visit", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Procedure> procedures = new ArrayList<>();

@OneToMany(mappedBy = "visit", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Prescription> prescriptions = new ArrayList<>();

}

Поля: id, date, reason, pet, procedures, prescriptions.

Зв’язки: @ManyToOne із Pet, @OneToMany із Procedure та Prescription.

1. Procedure

package example.petclinic.model;

import jakarta.persistence.\*;

import lombok.Data;

import java.math.BigDecimal;

@Entity

@Table(name = "procedures")

@Data

public class Procedure {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@Column(nullable = false)

private BigDecimal cost;

@ManyToOne

@JoinColumn(name = "visit\_id", nullable = false)

private Visit visit;

}

Поля: id, name, cost, visit.

Зв’язки: @ManyToOne із Visit.

1. Prescription

package example.petclinic.model;

import jakarta.persistence.\*;

import lombok.Data;

@Entity

@Table(name = "prescriptions")

@Data

public class Prescription {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String medication;

@Column(nullable = false)

private String dosage;

@ManyToOne

@JoinColumn(name = "visit\_id", nullable = false)

private Visit visit;

}

Поля: id, medication, dosage, visit.

Зв’язки: @ManyToOne із Visit.

* DTO-класи

1. AuthResponse

package example.petclinic.dto.auth;

import lombok.Data;

@Data

public class AuthResponse {

private String token;

private String role;

public AuthResponse(String token, String role) {

this.token = token;

this.role = role;

}

}

1. LoginRequest

package example.petclinic.dto.auth;

import jakarta.validation.constraints.NotBlank;

import lombok.Data;

@Data

public class LoginRequest {

@NotBlank(message = "Username is required")

private String username;

@NotBlank(message = "Password is required")

private String password;

}

RegisterRequest

package example.petclinic.dto.auth;

import jakarta.validation.constraints.NotBlank;

import lombok.Data;

@Data

public class RegisterRequest {

@NotBlank(message = "Username is required")

private String username;

@NotBlank(message = "Password is required")

private String password;

@NotBlank(message = "Role is required")

private String role;

}

1. OwnerDTO

package example.petclinic.dto;

import jakarta.validation.constraints.NotBlank;

import lombok.Data;

@Data

public class OwnerDTO {

@NotBlank(message = "Name is required")

private String name;

@NotBlank(message = "Phone is required")

private String phone;

}

1. PetDTO

package example.petclinic.dto;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.NotNull;

import lombok.Data;

import java.time.LocalDate;

@Data

public class PetDTO {

@NotBlank(message = "Name is required")

private String name;

@NotBlank(message = "Species is required")

private String species;

private LocalDate birthDate;

@NotNull(message = "Owner ID is required")

private Long ownerId;

}

1. VisitDTO

package example.petclinic.dto;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.NotNull;

import lombok.Data;

import java.time.LocalDate;

@Data

public class VisitDTO {

@NotNull(message = "Pet ID is required")

private Long petId;

@NotNull(message = "Date is required")

private LocalDate date;

@NotBlank(message = "Reason is required")

private String reason;

}

1. ProcedureDTO

package example.petclinic.dto;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.NotNull;

import lombok.Data;

import java.math.BigDecimal;

@Data

public class ProcedureDTO {

@NotNull(message = "Visit ID is required")

private Long visitId;

@NotBlank(message = "Name is required")

private String name;

@NotNull(message = "Cost is required")

private BigDecimal cost;

}

1. PrescriptionDTO

package example.petclinic.dto;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.NotNull;

import lombok.Data;

@Data

public class PrescriptionDTO {

@NotNull(message = "Visit ID is required")

private Long visitId;

@NotBlank(message = "Medication is required")

private String medication;

@NotBlank(message = "Dosage is required")

private String dosage;

}

Аналітичні DTO

* AnimalCostSummaryDTO: Містить сумарні витрати на тварину.
* PopularProcedureDTO: Повертає назву процедури та кількість.

- AnimalTypeStatsDTO: Статистика за видами тварин.

4.2 Репозиторії (інтерфейси доступу до даних)

Репозиторії забезпечують доступ до даних через Spring Data JPA.

1. UserRepository

package example.petclinic.repository;

import example.petclinic.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

User findByUsername(String username);

}

1. OwnerRepository

package example.petclinic.repository;

import example.petclinic.model.Owner;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface OwnerRepository extends JpaRepository<Owner, Long> {

List<Owner> findByNameContainingIgnoreCase(String name);

}

1. PetRepository

package example.petclinic.repository;

import example.petclinic.model.Pet;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import java.util.List;

import java.math.BigDecimal;

public interface PetRepository extends JpaRepository<Pet, Long> {

List<Pet> findByOwnerId(Long ownerId);

@Query("SELECT SUM(p.cost) FROM Procedure p WHERE p.visit.pet.id = :petId")

BigDecimal getTotalCostsForPet(@Param("petId") Long petId);

}

1. VisitRepository

package example.petclinic.repository;

import example.petclinic.model.Visit;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import java.util.List;

import java.time.LocalDate;

import java.util.Optional;

public interface VisitRepository extends JpaRepository<Visit, Long> {

List<Visit> findByPetId(Long petId);

List<Visit> findByDate(LocalDate date);

@Query("SELECT v FROM Visit v JOIN FETCH v.procedures WHERE v.id = :visitId")

Optional<Visit> findVisitWithProcedures(@Param("visitId") Long visitId);

}

1. ProcedureRepository

package example.petclinic.repository;

import example.petclinic.model.Procedure;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface ProcedureRepository extends JpaRepository<Procedure, Long> {

List<Procedure> findByVisitId(Long visitId);

@Query("SELECT p.name AS name, COUNT(p) AS count FROM Procedure p GROUP BY p.name ORDER BY count DESC")

List<Object[]> findMostPopularProcedures();

}

1. PrescriptionRepository

package example.petclinic.repository;

import example.petclinic.model.Prescription;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface PrescriptionRepository extends JpaRepository<Prescription, Long> {

List<Prescription> findByVisitId(Long visitId);

}

4.3 Сервіси (бізнес-логіка)

Сервіси інкапсулюють бізнес-логіку.

1. UserService

package example.petclinic.service;

import example.petclinic.dto.auth.LoginRequest;

import example.petclinic.dto.auth.RegisterRequest;

import example.petclinic.model.User;

import example.petclinic.repository.UserRepository;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Service;

@Service

@RequiredArgsConstructor

public class UserService {

private final UserRepository userRepository;

public void register(RegisterRequest request) {

if (userRepository.findByUsername(request.getUsername()) != null) {

throw new IllegalArgumentException("Username already exists");

}

User user = new User();

user.setUsername(request.getUsername());

user.setPassword(request.getPassword());

user.setRole(request.getRole());

userRepository.save(user);

}

public String login(LoginRequest request) {

User user = userRepository.findByUsername(request.getUsername());

if (user == null || !user.getPassword().equals(request.getPassword())) {

throw new IllegalArgumentException("Invalid credentials");

}

return "token";

}

}

1. OwnerService

package example.petclinic.service;

import example.petclinic.dto.OwnerDTO;

import example.petclinic.model.Owner;

import example.petclinic.repository.OwnerRepository;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Service;

import jakarta.persistence.EntityNotFoundException;

import java.util.List;

import java.util.stream.Collectors;

@Service

@RequiredArgsConstructor

public class OwnerService {

private final OwnerRepository ownerRepository;

public OwnerDTO createOwner(OwnerDTO dto) {

Owner owner = new Owner();

owner.setName(dto.getName());

owner.setPhone(dto.getPhone());

Owner saved = ownerRepository.save(owner);

return mapToDTO(saved);

}

public List<OwnerDTO> getAllOwners() {

return ownerRepository.findAll()

.stream()

.map(this::mapToDTO)

.collect(Collectors.toList());

}

public OwnerDTO getOwnerById(Long id) {

Owner owner = ownerRepository.findById(id)

.orElseThrow(() -> new EntityNotFoundException("Owner not found"));

return mapToDTO(owner);

}

private OwnerDTO mapToDTO(Owner owner) {

OwnerDTO dto = new OwnerDTO();

dto.setName(owner.getName());

dto.setPhone(owner.getPhone());

return dto;

}

}

1. PetService

package example.petclinic.service;

import example.petclinic.dto.PetDTO;

import example.petclinic.model.Pet;

import example.petclinic.model.Owner;

import example.petclinic.repository.PetRepository;

import example.petclinic.repository.OwnerRepository;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Service;

import jakarta.persistence.EntityNotFoundException;

import java.util.List;

import java.util.stream.Collectors;

@Service

@RequiredArgsConstructor

public class PetService {

private final PetRepository petRepository;

private final OwnerRepository ownerRepository;

public PetDTO createPet(PetDTO dto) {

Owner owner = ownerRepository.findById(dto.getOwnerId())

.orElseThrow(() -> new EntityNotFoundException("Owner not found"));

Pet pet = new Pet();

pet.setName(dto.getName());

pet.setSpecies(dto.getSpecies());

pet.setBirthDate(dto.getBirthDate());

pet.setOwner(owner);

Pet saved = petRepository.save(pet);

return mapToDTO(saved);

}

public List<PetDTO> getAllPets() {

return petRepository.findAll()

.stream()

.map(this::mapToDTO)

.collect(Collectors.toList());

}

public PetDTO getPetById(Long id) {

Pet pet = petRepository.findById(id)

.orElseThrow(() -> new EntityNotFoundException("Pet not found"));

return mapToDTO(pet);

}

public BigDecimal getTotalCostsForPet(Long petId) {

return petRepository.getTotalCostsForPet(petId);

}

private PetDTO mapToDTO(Pet pet) {

PetDTO dto = new PetDTO();

dto.setName(pet.getName());

dto.setSpecies(pet.getSpecies());

dto.setBirthDate(pet.getBirthDate());

dto.setOwnerId(pet.getOwner().getId());

return dto;

}

}

4.4 Контролери (обробка HTTP-запитів)

1. AuthController

package example.petclinic.controller;

import example.petclinic.dto.auth.AuthResponse;

import example.petclinic.dto.auth.LoginRequest;

import example.petclinic.dto.auth.RegisterRequest;

import example.petclinic.service.UserService;

import lombok.RequiredArgsConstructor;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/api/auth")

@RequiredArgsConstructor

public class AuthController {

private final UserService userService;

@PostMapping("/register")

public ResponseEntity<AuthResponse> register(@RequestBody RegisterRequest request) {

userService.register(request);

return ResponseEntity.status(201).body(new AuthResponse("token", request.getRole()));

}

@PostMapping("/login")

public ResponseEntity<AuthResponse> login(@RequestBody LoginRequest request) {

String token = userService.login(request);

return ResponseEntity.ok(new AuthResponse(token, "USER"));

}

}

2. PetController

package example.petclinic.controller;

import example.petclinic.dto.PetDTO;

import example.petclinic.service.PetService;

import lombok.RequiredArgsConstructor;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/pets")

@RequiredArgsConstructor

public class PetController {

private final PetService petService;

@PostMapping

public ResponseEntity<PetDTO> createPet(@RequestBody PetDTO petDTO) {

PetDTO createdPet = petService.createPet(petDTO);

return new ResponseEntity<>(createdPet, HttpStatus.CREATED);

}

@GetMapping

public ResponseEntity<List<PetDTO>> getAllPets() {

List<PetDTO> pets = petService.getAllPets();

return new ResponseEntity<>(pets, HttpStatus.OK);

}

@GetMapping("/{id}")

public ResponseEntity<PetDTO> getPetById(@PathVariable Long id) {

PetDTO pet = petService.getPetById(id);

return new ResponseEntity<>(pet, HttpStatus.OK);

}

@GetMapping("/{id}/expenses")

public ResponseEntity<BigDecimal> getTotalCostsForPet(@PathVariable Long id) {

BigDecimal totalCost = petService.getTotalCostsForPet(id);

return new ResponseEntity<>(totalCost, HttpStatus.OK);

}

}

4.5 Конфігурація (підключення БД, налаштування Spring Boot)

1. application.properties

spring.datasource.url=jdbc:postgresql://localhost:5432/vet\_clinic\_db

spring.datasource.username=postgres

spring.datasource.password=mypassword

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

server.port=8080

pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="[invalid url, do not cite]"

xmlns:xsi="[invalid url, do not cite]"

xsi:schemaLocation="[invalid url, do not cite] [invalid url, do not cite]">

<modelVersion>4.0.0</modelVersion>

<groupId>example.petclinic</groupId>

<artifactId>petclinic</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>petclinic</name>

<description>Pet Clinic application</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.1.0</version>

<relativePath/>

</parent>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

ТЕСТУВАННЯ

Тестування та налагодження є ключовими етапами розробки інформаційної системи для ветеринарної клініки, спрямованими на забезпечення її працездатності, стабільності та відповідності вимогам. Для перевірки системи застосовано комплексний підхід, що включає ручне тестування, автоматизовані тести та інструменти для тестування API. Нижче наведено методи перевірки, опис використання інструментів, результати тестування основних функцій, виявлені помилки та способи їх усунення. Ручне тестування виконувалося для перевірки основних CRUD-операцій для сутностей Owner, Pet, Visit, Procedure, Prescription та автентифікації User, перевірялася коректність відповідей API, обробка помилок при невалідних даних та поведінка системи під навантаженням. Автоматизоване тестування проводилося з використанням бібліотеки JUnit 5 із Spring Boot Test для модульних тестів сервісів і репозиторіїв, перевірено коректність збереження даних, валідацію DTO та обробку винятків. Тестування API здійснювалося за допомогою інструментів Postman і Swagger для перевірки ендпоінтів REST API, перевірялися HTTP-методи, статуси відповідей та структура JSON-відповідей. У Postman тестувалися ендпоінти, такі як створення сутностей, автентифікація та отримання даних, перевірялися запити з різними параметрами та тілами, а також обробка помилок при некоректних даних, тоді як у Swagger, за допомогою Springdoc OpenAPI, генерувалася автоматична документація API, і перевірялися ендпоінти через UI Swagger, що дозволило інтерактивно тестувати маршрути та валідацію полів. Результати перевірки основних функцій показали, що автентифікація User успішно пройшла для реєстрації та входу користувачів із коректною обробкою невалідних даних, CRUD-операції для Owner працюють коректно для всіх дій, включаючи повернення списку власників, CRUD для Pet забезпечує успішне створення тварини для власника з обробкою неіснуючих ідентифікаторів, CRUD для Visit коректно створює візити та повертає їх списки, CRUD для Procedure та Prescription функціонує при додаванні процедур і призначень із коректною агрегацією витрат, а аналітичні запити, такі як популярні процедури та статистика за видами тварин, повертають очікувані дані. Виявлені помилки включали внутрішню помилку сервера при невалідному ідентифікаторі тварини, що було усунено додаванням обробки винятку з поверненням відповідного статусу помилки, проблему з завантаженням пов’язаних даних через неефективні запити, яку усунули оптимізацією запитів у репозиторіях за допомогою JOIN FETCH, неправильну валідацію числових полів, що дозволяло некоректні значення, усунуту додаванням додаткової валідації до відповідних DTO, та повернення null при відсутності даних в агрегаційних запитах, що спричиняло помилки, усунуте додаванням обробки для повернення дефолтного значення. Висновки тестування підтвердили працездатність основних функцій системи, включаючи CRUD-операції та аналітичні запити, використання Postman і Swagger полегшило перевірку API, виявлені помилки, пов’язані з обробкою винятків, оптимізацією та валідацією, були успішно усунені, система готова до подальшого розвитку, але рекомендується додати інтеграційні тести для повної перевірки взаємодії між шарами.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Методичні вказівки до виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» для здобувачів інституту комп’ютерних систем за спеціальністю 122 Комп'ютерні науки, освітня програма «Комп’ютерні науки» / Укладачі: М.А. Годовиченко. – Одеса: Національний університет «Одеська Політехніка», 2024. – 59 с.
2. The OAuth 2.0 Authorization Framework. URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 27.06.2025).
3. JSON Web Tokens. URL: https://auth0.com/docs/secure/tokens/json-web-tokens (дата звернення: 29.06.2025).
4. Spring Data JPA Reference Documentation. URL: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/> (дата звернення: 28.06.2025).
5. Official OAuth 2.0 Authorization Framework. URL: <https://oauth.net/2/> (дата звернення: 28.06.2025).
6. Spring Boot Documentation Site. URL: <https://docs.spring.io/spring-boot/docs/current/reference/html/> (дата звернення: 29.06.2025).