Національний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни «Об’єктно-орієнтоване програмування»

Тема «Система обліку домашніх улюбленців у ветклініці»

Студентки__2__ курсу AI-233 групи
Спеціальності 122 – «Комп’ютерні науки»
_____Струкової К.С._____
(прізвище та ініціали)
Керівник доцент, к.т.н. Годовиченко М.А.
_____
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії

_____ _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)

м. Одеса – 2025 рік

Національний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра інформаційних систем

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ

студенці Струковій Карині Сергіївні група АІ-233

1. Тема роботи
«Система обліку домашніх улюбленців у ветклініці»

2. Термін здачі студентом закінченої роботи 13.06.2024

3. Початкові дані до проекту (роботи)
Варіант 20
Система дозволяє обліковувати тварин, їх власників, візити до ветеринара, процедури та призначення. Одна тварина може мати багато візитів. Система має реєстрацію користувача за допомогою логіну та паролю, автентифікацію за допомогою логіну та паролю, автентифікацію за допомогою jwt-токену

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити)
Вступ. Аналіз предметної області. Проєктування програмного забезпечення. Реалізація програмного продукту. Тестування та налагодження. Висновки.

Завдання видано 14.03.2024 _____
(підпис викладача)

Завдання прийнято до виконання 14.03.2024 _____
(підпис студента)

АНОТАЦІЯ

Струкова К.С. Система обліку домашніх улюбленців у ветклініці: курсова робота з дисципліни «Об’єктно-орієнтоване програмування» / Карина Струкова Сергіївна ; керівник Микола Анатолійович Годовиченко. – Одеса : Нац. ун-т «Одес. політехніка», 2025. – 46 с.

Розглянуто підходи та особливості розробки багаторівневих вебзастосунків із використанням архітектури Spring. Виконано аналіз основних етапів створення програмного продукту: від проєктування структури даних до реалізації REST API, автентифікації користувачів та тестування. Реалізовано логіку обробки ключових сутностей (власник, тварина, візит, процедура, призначення) з повною підтримкою CRUD-операцій. Запропоновано рішення щодо побудови безпечного доступу з розмежуванням ролей (адміністратор/користувач) та інтеграцію з JWT для захисту API.

Ключові слова: вебзастосунок, Spring Boot, REST API, база даних, автентифікація, ветеринарія, облік візитів, багаторівнева архітектура, рольова модель.

ABSTRACT

Strukova K.S. Pet Clinic Management System: course paper in the discipline “Object-Oriented Programming” / Karina Serhiivna Strukova ; supervisor Mykola Anatoliiovych Hodovychenko. – Odesa : Odesa Polytech. Nat. Univ., 2025. – 30 p.

The work explores approaches and features of developing multi-tier web applications using the Spring architecture. The main stages of software development are analyzed, from data structure design to REST API implementation, user authentication, and system testing. The logic for managing key entities (owner, pet, visit, procedure, prescription) is implemented with full CRUD support. A solution for secure role-based access (admin/user) is proposed along with JWT integration for API protection.

Keywords: web application, Spring Boot, REST API, database, authentication, veterinary, visit tracking, multi-tier architecture, role-based model.

ЗМІСТ

ВСТУП

Мета курсової роботи: систематизація, поглиблення та практичне закріплення знань студентів з дисципліни «Об'єктно-орієнтоване програмування», а також набуття навичок самостійної розробки серверної частини прикладного програмного забезпечення з використанням мови програмування Java та фреймворку Spring.

У сучасному світі цифрові технології відіграють важливу роль у забезпеченні ефективної роботи підприємств та установ, зокрема у сфері ветеринарної медицини. Ветеринарні клініки потребують автоматизованих систем для обліку пацієнтів, відстеження історії лікування, планування візитів та зберігання контактної інформації власників тварин. Традиційне ведення обліку на папері є незручним, трудомістким і схильним до помилок, що знижує якість обслуговування клієнтів. Тому актуальним є розроблення програмного забезпечення для автоматизації цих процесів.

Метою даної курсової роботи є розробка інформаційної системи для обліку домашніх улюбленців у ветеринарній клініці, яка дозволить ефективно управляти інформацією про тварин, їхніх власників, медичні процедури та візити до лікаря.

Основними завданнями роботи є:

–аналіз предметної області та визначення функціональних вимог до системи;

–розроблення об'єктно-орієнтованої моделі системи;

–реалізація програмного забезпечення з використанням мови програмування Java;

–використання фреймворку Spring для організації архітектури застосунку;

–застосування JPA (Java Persistence API) для збереження даних у реляційній базі даних;

–тестування функціональності системи.

У рамках курсової роботи розглядається предметна область, що охоплює основні аспекти діяльності ветеринарної клініки: реєстрація тварин, зберігання їхніх медичних карток, запис на прийом, облік процедур і лікарів.

Для реалізації проєкту обрано мову програмування Java завдяки її популярності, об'єктно-орієнтованій природі та підтримці широкого спектра бібліотек і фреймворків. Фреймворк Spring забезпечує гнучку модульну архітектуру, спрощує керування залежностями та дозволяє легко масштабувати застосунок. Технологія JPA забезпечує зручну роботу з базою даних, автоматизуючи збереження та відновлення об’єктів.

Таким чином, розробка системи обліку домашніх улюбленців у ветклініці спрямована на покращення якості обслуговування клієнтів, зменшення навантаження на персонал та забезпечення точності збереження медичних даних.

1 АНАЛІЗ ПРЕДМЕТОЇ ОБЛАСТІ

1.1 Аналіз системи

У ветеринарній клініці важливою складовою успішної роботи є точний облік тварин, їхніх власників, історії хвороб і процедур, а також своєчасне призначення лікування. Відсутність централізованої системи зберігання цих даних ускладнює доступ до інформації, призводить до дублювання записів, помилок у призначеннях та втрати даних про візити. З огляду на це виникає потреба в автоматизованій інформаційній системі, яка дозволить ефективно керувати всіма аспектами взаємодії клініки з клієнтами та пацієнтами.

Розроблювана система охоплює ключові об’єкти предметної області. Основною одиницею є тварина, для якої зберігається ім’я, вид, дата народження та інформація про власника. Кожна тварина може мати багато візитів до клініки, під час яких фіксується дата звернення, причина візиту, а також призначаються процедури та медикаменти. У межах одного візиту може бути виконано кілька процедур, таких як вакцинація, аналізи чи діагностичні дослідження, а також надані відповідні призначення із зазначенням ліків та дозування.

Процес використання системи починається з автентифікації користувача — працівника клініки — через логін і пароль. Після входу в систему, йому надається токен (JWT), який дозволяє здійснювати подальші запити до API. Основна взаємодія з базою даних відбувається через REST-запити, що охоплюють додавання, перегляд, редагування та видалення власників, тварин, візитів, процедур і призначень. Передбачено також функціонал для формування аналітичної інформації: розрахунок витрат по тварині, пошук візитів за датою, статистика за видами тварин, виявлення найпоширеніших процедур, перегляд останнього візиту.

Таким чином, система підтримує як повний життєвий цикл обслуговування пацієнта (від первинного візиту до завершення лікування), так і інформаційні потреби персоналу та адміністраторів клініки. Крім базового функціоналу,

реалізована можливість реєстрації нових користувачів і контроль доступу до ресурсів, що забезпечує безпечну роботу в багатокористувацькому середовищі.

Сутності системи взаємопов’язані між собою відповідно до логіки реального бізнес-процесу. Наприклад, кожен візит прив’язується до конкретної тварини, а та, у свою чергу, — до власника. Це дозволяє отримати повну історію звернень для кожного клієнта. Така структура даних забезпечує цілісність і узгодженість збереженої інформації, а також гнучкість для майбутнього масштабування системи.

У ветеринарній клініці важливою складовою успішної роботи є точний облік тварин, їхніх власників, історії хвороб і процедур, а також своєчасне призначення лікування. Відсутність централізованої системи зберігання цих даних ускладнює доступ до інформації, призводить до дублювання записів, помилок у призначеннях та втрати даних про візити. З огляду на це виникає потреба в автоматизованій інформаційній системі, яка дозволить ефективно керувати всіма аспектами взаємодії клініки з клієнтами та пацієнтами.

Розроблювана система охоплює ключові об’єкти предметної області. Основною одиницею є тварина, для якої зберігається ім’я, вид, дата народження та інформація про власника. Кожна тварина може мати багато візитів до клініки, під час яких фіксується дата звернення, причина візиту, а також призначаються процедури та медикаменти. У межах одного візиту може бути виконано кілька процедур, таких як вакцинація, аналізи чи діагностичні дослідження, а також надані відповідні призначення із зазначенням ліків та дозування.

Процес використання системи починається з автентифікації користувача — працівника клініки — через логін і пароль. Після входу в систему, йому надається токен (JWT), який дозволяє здійснювати подальші запити до API. Основна взаємодія з базою даних відбувається через REST-запити, що охоплюють додавання, перегляд, редагування та видалення власників, тварин, візитів, процедур і призначень. Передбачено також функціонал для формування аналітичної інформації: розрахунок витрат по тварині, пошук візитів за датою, статистика за видами тварин, виявлення найпоширеніших процедур, перегляд останнього візиту.

Таким чином, система підтримує як повний життєвий цикл обслуговування пацієнта (від первинного візиту до завершення лікування), так і інформаційні потреби персоналу та адміністраторів клініки. Крім базового функціоналу, реалізована можливість реєстрації нових користувачів і контроль доступу до ресурсів, що забезпечує безпечну роботу в багатокористувацькому середовищі.

Сутності системи взаємопов’язані між собою відповідно до логіки реального бізнес-процесу. Наприклад, кожен візит прив’язується до конкретної тварини, а та, у свою чергу, — до власника. Це дозволяє отримати повну історію звернень для кожного клієнта. Така структура даних забезпечує цілісність і узгодженість збереженої інформації, а також гнучкість для майбутнього масштабування системи.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Опис програми

Програма представляє собою систему обліку домашніх тварин у ветеринарній клініці. Вона дозволяє зберігати, переглядати та редагувати інформацію про тварин, їхніх власників, візити до клініки, а також медичні процедури та призначення. Користувачу доступна можливість додавати нових власників і тварин, фіксувати відвідування ветеринара, додавати процедури (наприклад, вакцинацію, діагностику) та призначення ліків для кожного візиту.

Програма надає зручні інструменти для редагування та видалення даних, отримання списку тварин певного власника, аналізу медичних витрат, пошуку останнього візиту, виводу візитів за датою або за назвою процедури, а також виведення найпопулярніших процедур та статистики по видам тварин.

Додатково реалізована система автентифікації — користувач повинен пройти реєстрацію з логіном та паролем, а для подальшого доступу до захищених ресурсів використовується JWT-токен.

Програма буде корисною для невеликих приватних ветеринарних клінік, де необхідно швидко організувати облік пацієнтів та зберігати історію лікування у зручному вигляді. Уся інформація може бути збережена в базі даних та відновлена або оновлена у разі потреби.

2.2 Опис використоуваної структури

У програмі використано набір класів, які відображають основні сутності предметної області. Усі вони об’єднані в однозв'язну логічну структуру через відповідні зв’язки між об’єктами. Наприклад, в об'єкті Pet є посилання на Owner, у Visit — на Pet, у Prescription — на Visit. Такий підхід забезпечує природну модель даних, що відповідає реальній структурі роботи клініки.

Усі сутності мають анотації JPA, що забезпечує збереження в базі даних, а також взаємодію з нею через рівень репозиторіїв. Це дає змогу шукати, оновлювати, обробляти дані, використовуючи об'єктно-орієнтований підхід.

2.3 Опис сутностей

У програмі було використано шість основних сутностей: Owner, Pet, Visit, Procedure, Prescription, User. Кожна з них відповідає реальному об’єкту або процесу в предметній області та містить набір полів, необхідних для коректної роботи системи обліку у ветеринарній клініці.

Сутність Owner (власник тварини) описує клієнта ветеринарної клініки — власника однієї або кількох тварин. Містить такі поля, як id, name, phone. Сутність Pet (домашня тварина) – сутність, що містить інформацію про пацієнта клініки — домашню тварину. Вона має поля id, name, species, birthDate, owner: Посилання на власника (Owner). Сутність Visit відображає факт звернення тварини до клініки, їй властиві поля id, date, reason та pet. Procedure є в свою чергу сутністю, що описує процедуру, проведену під час візиту. В неї є поля id, name, cost, visit тощо. Стуність Prescription (призначення) описувтаиме медикаментозне призначення, зроблене під час візиту. В неї є поля id, medication, dosage, visit: Остання сутність User (користувач системи) описує обліковий запис користувача (адміністратор або персонал клініки) і використовується для автентифікації. У цієї сутності є поля id, password: Захешований пароль (тип String) та role: Роль користувача в системі (наприклад, «ADMIN», «USER»).

Сутність User пов’язана з системою автентифікації (JWT, Spring Security) і не має прямого зв’язку з медичною моделлю, але забезпечує контроль доступу до REST API.

2.4 Опис архітектури застосунку

Застосунок реалізовано за класичною багаторівневою архітектурою, яка складається з трьох основних рівнів: Controller, Service та Repository. Такий підхід забезпечує чітке розділення відповідальностей, гнучкість та зручність підтримки коду.

Контролери (Controller) відповідають за обробку HTTP-запитів, отриманих від клієнта (наприклад, через REST API). Вони приймають вхідні дані, викликають відповідні методи сервісного рівня і формують HTTP-відповіді. Реалізують бізнес-логіку не безпосередньо, а через виклики сервісів, забезпечують валідацію вхідних параметрів і трансформацію даних між HTTP-моделями та внутрішніми об’єктами.

Сервіси (Service) містять основну бізнес-логіку застосунку. Вони координують роботу з кількома репозиторіями, реалізують перевірки, обчислення та інші операції, необхідні для коректного виконання функцій. Сервіси викликають методи репозиторіїв для доступу до даних. Вони не повинні залежати від деталей HTTP або бази даних — вони оперують доменними моделями.

Репозиторії (Repository) відповідають за безпосередню взаємодію із базою даних. Використовуються засоби JPA (Java Persistence API) для зручного мапінгу об’єктів на записи таблиць. Репозиторії надають методи для CRUD-операцій (створення, читання, оновлення, видалення). Вони ізольовані від бізнес-логіки і не містять її.

Controller залежить від Service — викликає його методи для обробки бізнес-логіки. Service залежить від Repository — використовує репозиторії для доступу до даних. Repository працює напряму з базою даних і не залежить від інших рівнів. Завдяки такому поділу кожен рівень можна тестувати окремо (наприклад, юніт-тести для сервісів із заміною репозиторіїв моками).

2.5. Опис REST API

У програмі реалізовано REST API, що надає клієнтам (наприклад, веб- або мобільному застосунку) доступ до основних функцій системи. API реалізовано з використанням стандартних HTTP-методів: GET, POST, PUT, DELETE. Усі запити передбачають обмін даними у форматі JSON.

Наприклад, додавання власника виконується через метод POST на /owner, отримання всіх власників через метод GET на /owner, додавання тварини відбувається через метод POST на /pet, додавання візитів через метод POST на /visit, додавання процедури до візіиту через метод POST на /procedures, отримання загальних витрат по тварині можна здійснити через метод GET на /procedure/expenses/total/{petId}, а, наприклад, авторизація користувача відбувається через метод POST на /api/login. Ці запити реалізують ключові функції системи: управління власниками, тваринами, візитами, процедурами та доступом до системи через JWT-автентифікацію. REST API дозволяє зручно інтегрувати систему з іншими клієнтськими інтерфейсами.

3 РЕАЛІЗАЦІЯ ПРОДУКТУ

3.1 Опис реалізації моделей

У програмному продукті використовуються Java-класи, що відповідають сутностям предметної області. Для зберігання даних у базі реалізовані entity-класи, а для передавання даних через REST API.

Було створено клас Owner

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Owner {

    @Id
    @GeneratedValue (strategy = GenerationType.AUTO)
    private Long id;

    private String name;
    private String phone;

    @OneToMany(mappedBy = "owner", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<Pet> pets;
}
```

Створено клас Pet

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Pet {

    @Id
    @GeneratedValue (strategy = GenerationType.AUTO)
    private Long id;
    private String name;
```

```
    private String species;
    private LocalDate birthDate;

    @ManyToOne
    @JoinColumn(name = "owner_id")
    @JsonBackReference
    private Owner owner;

    @OneToMany(mappedBy = "pet", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<Visit> visits;
}
```

Створено клас Visit

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Visit {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "pet_id")
    @JsonBackReference
    private Pet pet;

    private LocalDate date;
    private String reason;

    @OneToMany(mappedBy = "visit", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<Procedure> procedures;

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

```
    @JsonManagedReference
    private List<Prescription> prescriptions;
}
```

Створено клас Procedure

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Procedure {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "visit_id")
    @JsonBackReference
    private Visit visit;

    private String name;
    private BigDecimal cost;
}
```

Створено клас Prescription

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Prescription {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "visit_id")
    @JsonBackReference
```

```
    private Visit visit;

    private String medication;
    private String dosage;
}
```

Створено клас User

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "accounts")
public class User implements UserDetails {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @Column(unique = true, nullable = false)
  private String username;
  private String password;

  private String role = "USER";

  public Long getId() {
    return id;
  }

  @Override
  public Collection<? extends GrantedAuthority> getAuthorities() {
    return Collections.singleton(() -> role);
  }
}
```

3.2 Бізнес-логіка (Сервіси)

Усі основні бізнес-операції в застосунку реалізовані в сервісному шарі. Цей шар виконує обробку вхідних даних, перевірку, трансформацію моделей, взаємодію з репозиторіями та формування результатів для контролерів.

OwnerService: управління власниками тварин.

Основними методами є addNewOwner(Owner owner) — створення нового власника, updateOwner(Long id, Owner owner) — оновлення даних власника., getAllOwners() — повертає список усіх власників, deleteOwner(Long id) — видалення власника

```
@Service
public class OwnerService {

   private OwnerRepository ownerRepository;

   @Autowired
   public OwnerService(OwnerRepository ownerRepository) {
     this.ownerRepository = ownerRepository;
   }

   public Owner addNewOwner(Owner owner) {
      return ownerRepository.save(owner);
   }

   public List<Owner> getAllOwners(){
      return ownerRepository.findAll();
   }

   public Owner updateOwner(Long id, Owner owner) {
      Owner existingOwner = ownerRepository.findById(id).orElseThrow(() -> new RuntimeException("Owner not found"));
      existingOwner.setName(owner.getName());
      existingOwner.setPhone(owner.getPhone());
      return ownerRepository.save(existingOwner);
   }
```

```
    public void deleteOwner(Long id) {
        ownerRepository.deleteById(id);
    }
}
```

PetService: управління тваринами

Основними методами є addNewPet(Pet pet) — додавання тварини до певного власника, updatePet(Long id, Pet pet) — оновлення інформації про тварину, getPetsByOwner(Long id) — отримання списку тварин за ідентифікатором власника, deletePet(Long id) — видалення тварини, getPetTypeStatistics() — отримання статистики по видам тварин

```
@Service
public class PetService {

    private PetRepository petRepository;

    @Autowired
    public PetService(PetRepository petRepository) {
        this.petRepository = petRepository;
    }

    public Pet addNewPet(Pet pet) {
        return petRepository.save(pet);
    }

    public List<Pet> getPetsByOwner(Long id) {
        return petRepository.findByOwnerId(id);
    }

    public Pet updatePet(Long id, Pet pet) {
        Pet existingPet = petRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Pet not found"));
        existingPet.setName(pet.getName());
        existingPet.setSpecies(pet.getSpecies());
        existingPet.setBirthDate(pet.getBirthDate());
```

```
        existingPet.setOwner(pet.getOwner());
        return petRepository.save(existingPet);
    }

    public void deletePet(Long id) {
        petRepository.deleteById(id);
    }

    public List<Object[]> getPetTypeStatistics() {
        return petRepository.getPetTypeStatistics();
    }
}
```

VisitService: управління візитами тварин

Основні методи: addNewVisit(Visit visit) — створення візиту для тварини, getVisitsByPet(Long id) — отримання списку візитів певної тварини, updateVisit(Long id, Visit visit) — оновлення інформації про візит, deleteVisit(Long id) — видалення візиту, getLastVisit(Long petId) — отримання останнього візиту, getVisitsByDate(LocalDate date) — всі візити за датою.

```
@Service
public class VisitService {

    private VisitRepository visitRepository;

    @Autowired
    public VisitService(VisitRepository visitRepository) {
        this.visitRepository = visitRepository;
    }

    public Visit addNewVisit(Visit visit) {
        return visitRepository.save(visit);
    }

    public List<Visit> getVisitsByPet(Long id) {
        return visitRepository.findAllByPetId(id);
```

```
    }

    @Autowired
    private PetRepository petRepository;

    public Visit updateVisit(Long id, Visit visit) {
        Visit existingVisit = visitRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Visit not found"));
        existingVisit.setDate(visit.getDate());
        existingVisit.setReason(visit.getReason());
        if (visit.getPet() != null && visit.getPet().getId() != null) {
            Pet pet = petRepository.findById(visit.getPet().getId())
                .orElseThrow(() -> new RuntimeException("Pet not found"));
            existingVisit.setPet(pet);
        }
        return visitRepository.save(existingVisit);
    }

    public void deleteVisit(Long id) {
        visitRepository.deleteById(id);
    }

    public List<Visit> getVisitsByDate(LocalDate date) {
        return visitRepository.findByDate(date);
    }

    public Visit getLastVisit(Long petId) {
        List<Visit> visits = visitRepository.findLastVisitByPetId(petId);
        if (visits.isEmpty()) {
            throw new RuntimeException("No visits found for pet");
        }
        return visits.get(0);
    }
}
```

ProcedureService: управління медичними процедурами

Основні методи: addNewProcedure(Procedure procedure) — додавання процедури до візиту, getProceduresByVisit(Long visitId) — список процедур певного візиту, updateProcedure(Long id, Procedure procedure) — оновлення процедури, deleteProcedure(Long id) — видалення процедури, getTotalExpensesByPet(Long petId) — отримати загальні витрати по тварині, getPopularProcedures() — отримати популярні процедури

```
@Service
public class ProcedureService {

   private ProcedureRepository procedureRepository;

   @Autowired
   public ProcedureService(ProcedureRepository procedureRepository) {
      this.procedureRepository = procedureRepository;
   }

   public Procedure addNewProcedure(Procedure procedure) {
      return procedureRepository.save(procedure);
   }

   public List<Procedure> getProceduresByVisit(Long visitId) {
      return procedureRepository.findAllByVisitId(visitId);
   }

   @Autowired
   private VisitRepository visitRepository;

   public Procedure updateProcedure(Long id, Procedure procedure) {
      Procedure existingProcedure = procedureRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Procedure not found"));
      existingProcedure.setName(procedure.getName());
      existingProcedure.setCost(procedure.getCost());
      if (procedure.getVisit() != null && procedure.getVisit().getId() != null) {
         Visit visit = visitRepository.findById(procedure.getVisit().getId())
            .orElseThrow(() -> new RuntimeException("Visit not found"));
```

```
            existingProcedure.setVisit(visit);
        }
        return procedureRepository.save(existingProcedure);
    }

    public void deleteProcedure(Long id) {
        procedureRepository.deleteById(id);
    }

    public BigDecimal getTotalExpensesByPet(Long petId) {
        return Optional.ofNullable(procedureRepository.getAllExpensesPerPet(petId))
            .orElse(BigDecimal.ZERO);
    }

    public List<Object[]> getPopularProcedures(){
        return procedureRepository.getAllPopularProcedures();
    }
}
```

PrescriptionService: керування призначеннями (ліками)

Основні методи: addNewPrescription(Prescription prescription) — додати призначення, getPrescriptionsByVisit(Long visitId) — отримати призначення за візитом, updatePrescription(Long id, Prescription prescription) — оновлення призначення, deletePrescription(Long id) — видалення призначення

```
@Service
public class PrescriptionService {

    private PrescriptionRepository prescriptionRepository;

    @Autowired
    public PrescriptionService(PrescriptionRepository prescriptionRepository) {
        this.prescriptionRepository = prescriptionRepository;
    }

    public Prescription addNewPrescription(Prescription prescription) {
        return prescriptionRepository.save(prescription);
```

```
    }

    public List<Prescription> getPrescriptionsByVisit(Long visitId) {
        return prescriptionRepository.findAllByVisitId(visitId);
    }

    public Prescription updatePrescription(Long id, Prescription prescription) {
        Prescription existingPrescription = prescriptionRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Prescription not found"));
        existingPrescription.setMedication(prescription.getMedication());
        existingPrescription.setDosage(prescription.getDosage());
        existingPrescription.setVisit(prescription.getVisit());
        return prescriptionRepository.save(existingPrescription);
    }

    public void deletePrescription(Long id) {
        prescriptionRepository.deleteById(id);
    }
}
```

3.3 Контролери

Контролери є частиною архітектурного рівня Controller і відповідають за обробку HTTP-запитів, що надходять від клієнта (наприклад, frontend або Postman). Вони приймають запити, передають дані до відповідних сервісів для обробки, та повертають результати у вигляді HTTP-відповіді.

У програмі реалізовано окремі контролери для основних сутностей

OwnerController: обробляє запити, пов’язані з власниками тварин:

Основні контролери: додавання нового власника (POST /owner), отримання списку всіх власників (GET /owner), оновлення даних власника (PUT /owner/{id}), видалення власника (DELETE /owner/{id})

```
@RestController
@RequestMapping("/owner")
public class OwnerController {
```

```
    private OwnerService ownerService;

    @Autowired
    public OwnerController(OwnerService ownerService) {
      this.ownerService = ownerService;
    }

    @PostMapping
    public ResponseEntity<Owner> addNewOwner(@RequestBody Owner owner) {
      return new ResponseEntity<>(ownerService.addNewOwner(owner),
HttpStatus.CREATED);
    }

    @GetMapping
    public List<Owner> getAllOwners() {
      return ownerService.getAllOwners();
    }

    @PutMapping("/{id}")
    public Owner updateOwner(@PathVariable Long id, @RequestBody Owner owner) {
      return ownerService.updateOwner(id, owner);
    }

    @DeleteMapping("/{id}")
    public void deleteOwner(@PathVariable Long id) {
      ownerService.deleteOwner(id);
    }
  }
```

PetController: керує даними про домашніх тварин:

Основні контролери: додавання тварини (POST /pet/owner), отримання тварин певного власника (GET /pet/owner/{id}), оновлення тварин (PUT /pet/{id}), видалення тварини (DELETE /pet/{id}), отримати список видів тварин (GET /pet/statistics)

```
@RestController
@RequestMapping("/pet")
public class PetController {

    private PetService petService;

    @Autowired
    public PetController(PetService petService) {
        this.petService = petService;
    }

    @PostMapping
    public ResponseEntity<Pet> addNewPet(@RequestBody Pet pet) {
        return new ResponseEntity<>(petService.addNewPet(pet), HttpStatus.CREATED);
    }

    @GetMapping("/owner/{id}")
    public List<Pet> getPetsByOwner(@PathVariable Long id) {
        return petService.getPetsByOwner(id);
    }

    @PutMapping("/{id}")
    public Pet updatePet(@PathVariable Long id, @RequestBody Pet pet) {
        return petService.updatePet(id, pet);
    }

    @DeleteMapping("/{id}")
    public void deletePet(@PathVariable Long id) {
        petService.deletePet(id);
    }

    @GetMapping("/statistics")
    public List<Object[]> getPetTypeStatistics() {
        return petService.getPetTypeStatistics();
    }
}
```

VisitController: обробляє візити до ветеринара

Основні контролери: створення візиту (POST /visits), перегляд візитів тварини (GET /visit/pet/{id}), редагування візиту (PUT/visit/{id}), видалення візиту (DELETE /visit/{id}), отримати інформацію про візити по даті (GET /visit/{date}), інформація про останній візит тварини(GET /visit/pets/{petId}/last-visit)

```
@RestController
@RequestMapping("/visit")
public class VisitController {

   private VisitService visitService;

   @Autowired
   public VisitController(VisitService visitService) {
      this.visitService = visitService;
   }

   @PostMapping
   public ResponseEntity<Visit> addNewVisit(@RequestBody Visit visit) {
      return new ResponseEntity<>(visitService.addNewVisit(visit), HttpStatus.CREATED);
   }

   @GetMapping("/pet/{id}")
   public List<Visit> getVisitsByPet(@PathVariable Long id) {
      return visitService.getVisitsByPet(id);
   }

   @PutMapping("/{id}")
   public Visit updateVisit(@PathVariable Long id, @RequestBody Visit visit) {
      return visitService.updateVisit(id, visit);
   }

   @DeleteMapping("/{id}")
   public void deleteVisit(@PathVariable Long id) {
      visitService.deleteVisit(id);
```

```
    }

    @GetMapping("/{date}")
    public List<Visit> getVisitsByDate(@PathVariable LocalDate date) {
      return visitService.getVisitsByDate(date);
    }

    @GetMapping("/pets/{petId}/last-visit")
    public Visit getLastVisit(@PathVariable Long petId) {
      return visitService.getLastVisit(petId);
    }
}
```

ProcedureController: реалізує роботу з процедурами в межах візиту

Основні контролери: додавання процедури до візиту (POST /procedures), отримання процедур за візитом (GET /procedure/visit/{visitId}), оновлення, (PUT /procedures), видалення процедур (DELETE /procedures)

```
@RestController
@RequestMapping("/procedure")
public class ProcedureController {

    private ProcedureService procedureService;

    @Autowired
    public ProcedureController(ProcedureService procedureService) {
      this.procedureService = procedureService;

    }

    @PostMapping
    public ResponseEntity<Procedure> addNewProcedure(@RequestBody Procedure procedure)
{
      return new ResponseEntity<>(procedureService.addNewProcedure(procedure),
HttpStatus.CREATED);
    }
```

```
    @GetMapping("/visit/{visitId}")
    public List<Procedure> getProceduresByVisit(@PathVariable Long visitId) {
        return procedureService.getProceduresByVisit(visitId);
    }

    @PutMapping("/{id}")
    public Procedure updateProcedure(@PathVariable Long id, @RequestBody Procedure procedure) {
        return procedureService.updateProcedure(id, procedure);
    }

    @DeleteMapping("/{id}")
    public void deleteProcedure(@PathVariable Long id) {
        procedureService.deleteProcedure(id);
    }

    @GetMapping("/expenses/total/{petId}")
    public BigDecimal totalExpensesByPet(@PathVariable Long petId) {
        return procedureService.getTotalExpensesByPet(petId);
    }

    @GetMapping("/popular")
    public List<Object[]> popularProcedures() {
        return procedureService.getPopularProcedures();
    }
}
```

PrescriptionController: керує призначеннями (медикаментами):

Популярні контролери: додавання призначень до візиту (POST / prescription), отримання призначень за візитом (GET /prescription/visit/{visitId}), оновлення, (PUT / prescription), видалення призначень (DELETE /prescription)

```
@RestController
@RequestMapping("/prescription")
public class PrescriptionController {

    private PrescriptionService prescriptionService;
```

```
    @Autowired
    public PrescriptionController(PrescriptionService prescriptionService) {
        this.prescriptionService = prescriptionService;
    }

    @PostMapping
    public ResponseEntity<Prescription> addNewPrescription(@RequestBody Prescription prescription) {
        return new ResponseEntity<>(prescriptionService.addNewPrescription(prescription), HttpStatus.CREATED);
    }

    @GetMapping("/visit/{visitId}")
    public List<Prescription> getPrescriptionsByVisit(@PathVariable Long visitId) {
        return prescriptionService.getPrescriptionsByVisit(visitId);
    }

    @PutMapping("/{id}")
    public Prescription updatePrescription(@PathVariable Long id, @RequestBody Prescription prescription) {
        return prescriptionService.updatePrescription(id, prescription);
    }

    @DeleteMapping("/{id}")
    public void deletePrescription(@PathVariable Long id) {
        prescriptionService.deletePrescription(id);
    }

}
```

UserController: MVC контролер, повертає назву представлення (HTML-шаблону)
Контролер: повертає юзера (GET /user)

```
@Controller
public class UserController {
```

```
    @GetMapping("/user")
    public String user() {
        return "user";
    }
}
```

AdminController: MVC контролер, повертає назву представлення (HTML-шаблону)
Контролер: повертає юзера (GET /admin)

```
@Controller
public class AdminController {

    @GetMapping("/admin")
    public String admin() {
        return "admin";
    }
}
```

AuthController: MVC контролери при авторизацій та автентифікації на веб сервеісі

Основні контролери: регістрація (POST /register), повертання на сторінку (GET /register), повертання на сторінку (GET /login), редірект при успішному логіні (GET /success)

```
@Controller
public class AuthController {

    private final UserRepository userRepository;

    private final PasswordEncoder passwordEncoder;

    @Autowired
    public AuthController(UserRepository userRepository, PasswordEncoder passwordEncoder)
{
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }
```

```
    @PostMapping("/register")
    public String registerSubmit(@ModelAttribute("user") User user, @RequestParam String
role) {
        user.setRole("ROLE_" + role.toUpperCase());
        userRepository.save(user);
        return "redirect:/login";
    }

    @GetMapping("/register")
    public String registerForm(Model model) {
        model.addAttribute("user", new User());
        return "register";
    }

    @GetMapping("/login")
    public String loginForm() {
        return "login";
    }

    @GetMapping("/success")
    public String success(Authentication auth) {
        if(auth.getAuthorities().stream().anyMatch(a                                    ->
a.getAuthority().equals("ROLE_ADMIN"))){
            return "redirect:/admin";
        }
        return "redirect:/user";
    }
}
```

ApiController: MVC контролери при авторизацій та автентифікації на застосунках

Основні контролери: регістрація (POST /register), логін (POST/login)

```
@RestController
@RequestMapping("/api")
public class ApiController {
```

```
    private final UserRepository userRepository;

    private final PasswordEncoder passwordEncoder;

    private final AuthenticationManager authenticationManager;

    private final JwtProvider jwtProvider;

    public ApiController(UserRepository userRepository, PasswordEncoder passwordEncoder,
AuthenticationManager authenticationManager, JwtProvider jwtProvider) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
        this.authenticationManager = authenticationManager;
        this.jwtProvider = jwtProvider;
    }

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody AuthRequest request) {

        if(userRepository.findByUsername(request.getUsername()).isPresent()){
            return ResponseEntity.badRequest().body("Username is already taken");
        }

        User user = new User();
        user.setUsername(request.getUsername());
        user.setPassword(passwordEncoder.encode(request.getPassword()));

        String role = request.getRole();
        if(role == null || (!role.equalsIgnoreCase("user") && !role.equalsIgnoreCase("admin"))){
            role = "user";  // роль за замовчуванням
        }
        user.setRole("ROLE_" + role.toUpperCase());
        userRepository.save(user);
```

```
        return ResponseEntity.ok().body(user);
    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody AuthRequest request) {
        Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword()));

        String token = jwtProvider.generateToken(authentication);

        return ResponseEntity.ok(Map.of("token", token));
    }
}
```

3.4 Компоненти безпеки

JwtProvider: генерація токенів

```
@Component public class JwtProvider {

private final Key key =
Keys.hmacShaKeyFor("my_super_secure_secret_key_1234567890".getBytes(StandardCharsets.UTF_
8));

private final long expiration = 3600000;

public String generateToken(Authentication authentication) {
  Date now = new Date();
  Date expirationDate = new Date(now.getTime() + expiration);

  return Jwts.builder()
    .setSubject(authentication.getName())
    .setIssuedAt(now)
    .setExpiration(expirationDate)
    .signWith(key, SignatureAlgorithm.HS256)
    .compact();
```

```
}

public boolean validateToken(String token){
  try{
    Jwts.parser()
      .setSigningKey(key)
      .build()
      .parseClaimsJws(token)
      .getBody();
    return true;
  } catch (Exception e){
    System.out.println("EXCEPTION: " + e.getClass().getSimpleName() + " - " + e.getMessage());
    return false;
  }
}

public String getUsernameFromToken(String token){
  return Jwts.parser()
    .setSigningKey(key)
    .build()
    .parseClaimsJws(token)
    .getBody()
    .getSubject();}
      }
```

JwtAuthFilter: фільтрація HTTP запитів

```
@Component
public class JwtAuthFilter extends OncePerRequestFilter {

    private final JwtProvider jwtProvider;
    private final CustomUserDetailService customUserDetailService;

    @Autowired
    public JwtAuthFilter(JwtProvider jwtProvider, CustomUserDetailService
customUserDetailService) {
        this.jwtProvider = jwtProvider;
```

```
        this.customUserDetailService = customUserDetailService;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain) throws ServletException, IOException {

        String token = getToken(request);

        if (token == null){
            filterChain.doFilter(request, response);
            return;
        }

        if (!jwtProvider.validateToken(token)){
            filterChain.doFilter(request, response);
        return;
        }

        String username = jwtProvider.getUsernameFromToken(token);
        UserDetails userDetails = customUserDetailService.loadUserByUsername(username);

        UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
        SecurityContextHolder.getContext().setAuthentication(authentication);

        filterChain.doFilter(request, response);
    }


    private String getToken(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");

        if  (bearerToken == null || !bearerToken.startsWith("Bearer "))
            return null;
```

```
        return bearerToken.substring(7);
    }
}
```

CustomUserDetailService: керування інформацією аккаунта

Основний метод: a loadUserByUsername(String username) — пошук аккаунта по юзернейму

```
@Component
public class CustomUserDetailService implements UserDetailsService {

    private final UserRepository userRepository;

    public CustomUserDetailService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return userRepository.findByUsername(username).orElseThrow(() ->
            new UsernameNotFoundException("User " + username + " not found"));
    }
}
```

4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Протестуємо роботу авторизації та автентифікації спочатку на ВЕБ-сайті, а потім у застосунку Postman.

При вводі коректного юзернейму та паролю здійснюєтьс переадресація на відповідну сторінку(рис.1)

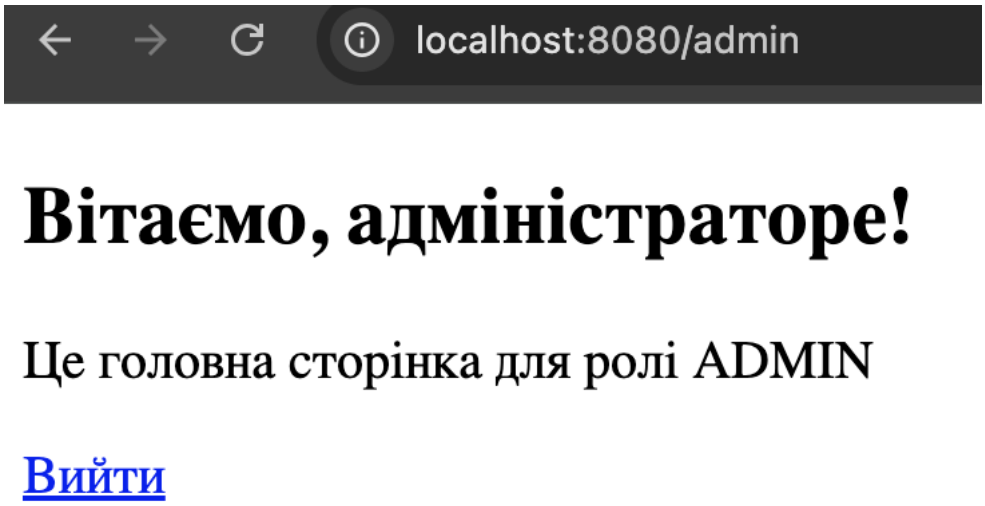

Рис. 1 успішна автентифікація користувача з ролю ADMIN

При спробі отримати доступ до сторінки, на яку не було надано повноважень, з’явиться повідомлення, що такої сторінки знайдено не було(рис.2)

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Jun 13 16:40:48 EEST 2025
There was an unexpected error (type=Not Found, status=404).

Рис.2 USER намагається зайти на сторінку для користувачів зі статусом ADMIN

Протестуємо роботу JWT, скориставшись Postman. Спочатку отримаємо наш токен, скориставшись Auth Type: Basik Auth. Введемо коректний юзер та пароль і отримуємо токен (рис3).Потім можемо скористатися їм, обравши Auth Type: Bearer Token, зазначивши коректні дані в боді, Payload і секретне слово, що зазначено в коді(рис.4)

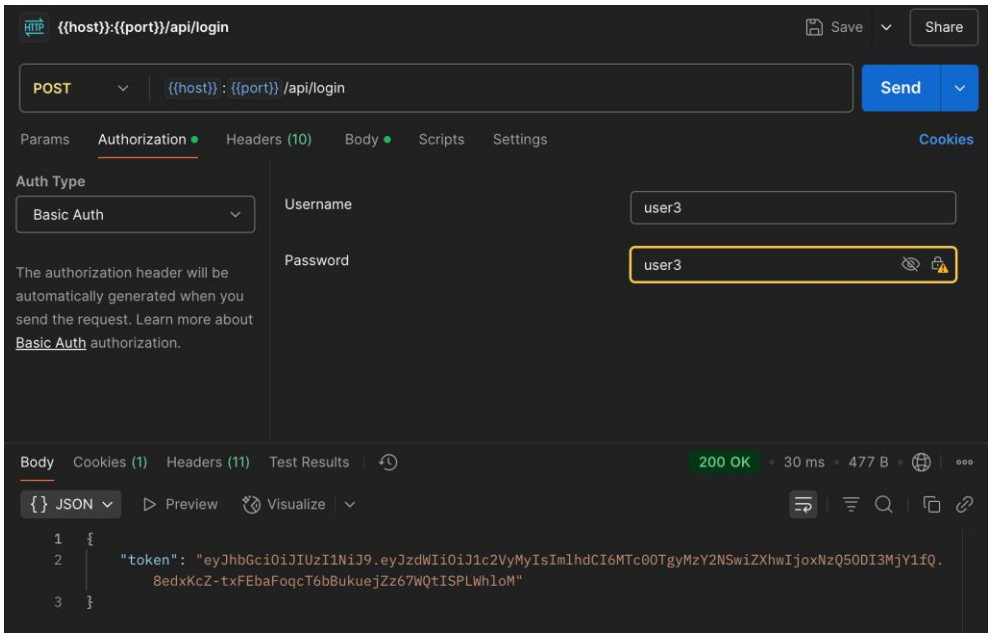

Рис.3 налаштування для отримання JWT токену

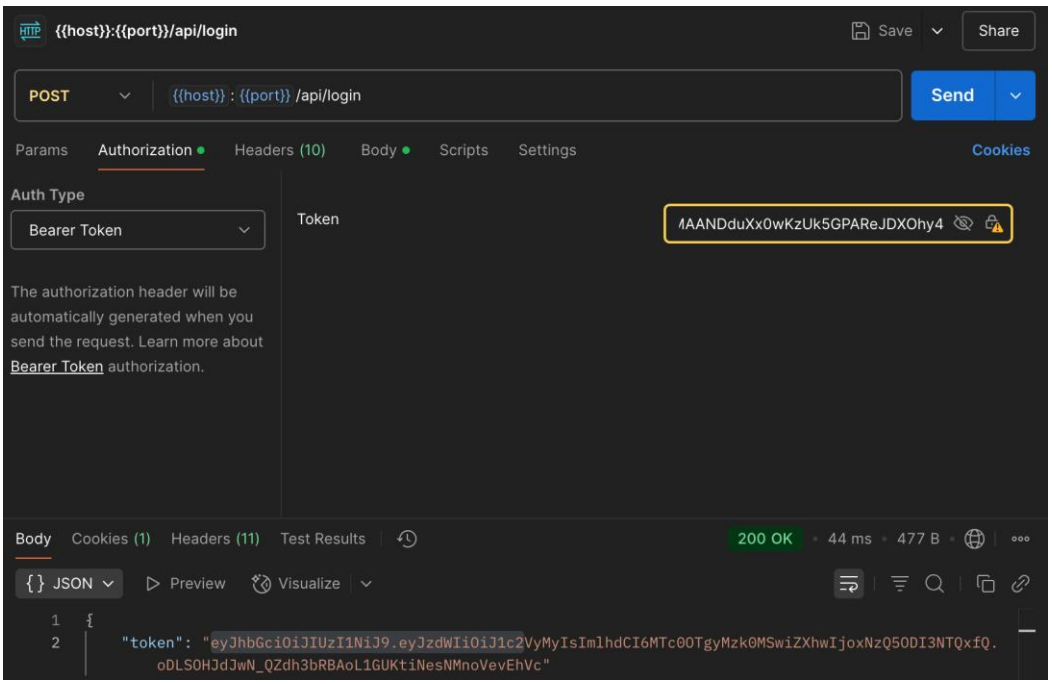

Рис.4 використання токену для логіну

Перевіримо роботу нашої БД, скориставшися REST-контролерами. Додамо власника, улюбленців, переконаємося, що запити виконуються коректно і зберігаються зі вкладеними улюбленцями та всією іншою інформацією(рис. 5)

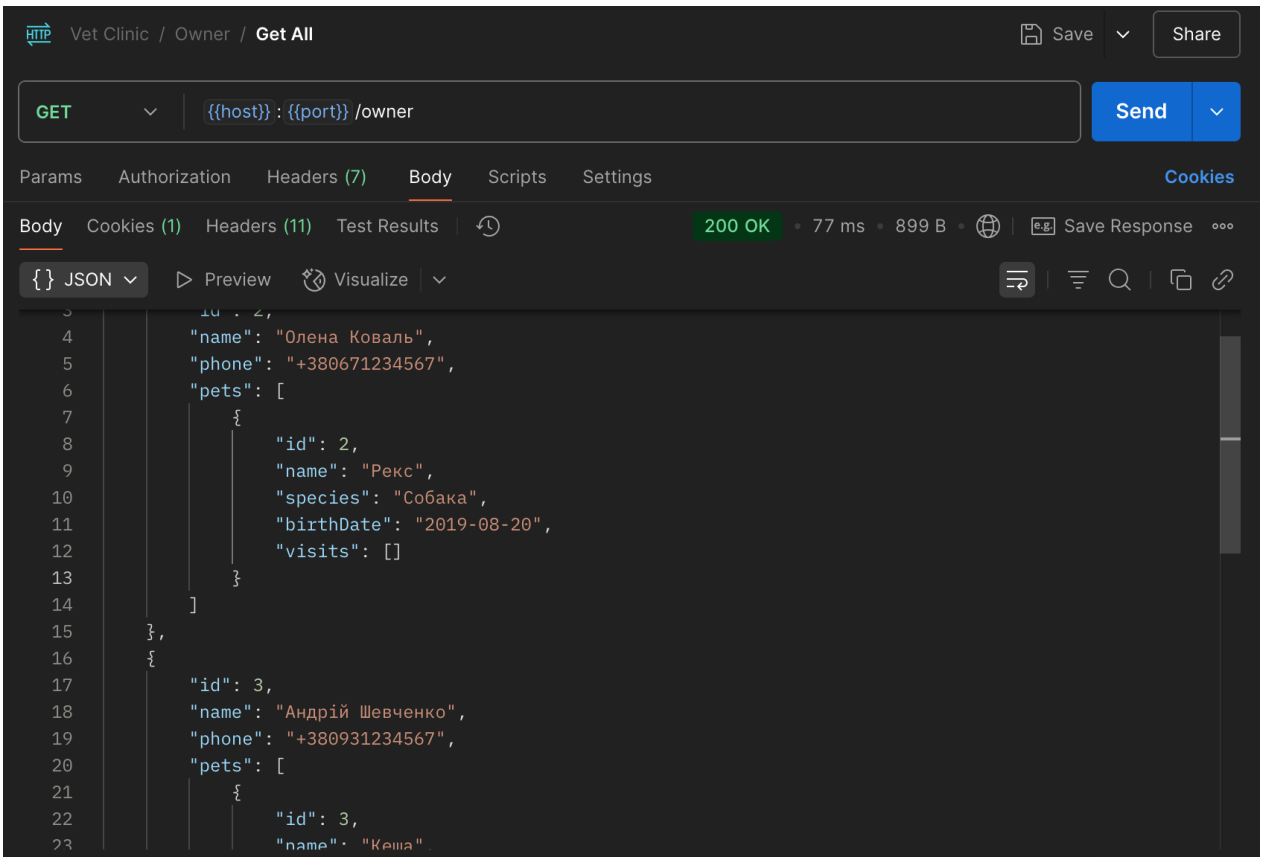

Рис.5 Перегляд всіх користувачів та їх улюбленців

Можемо також переглянути інші REST-запити, та переконатися, що ті працюють коректно(рис.6)

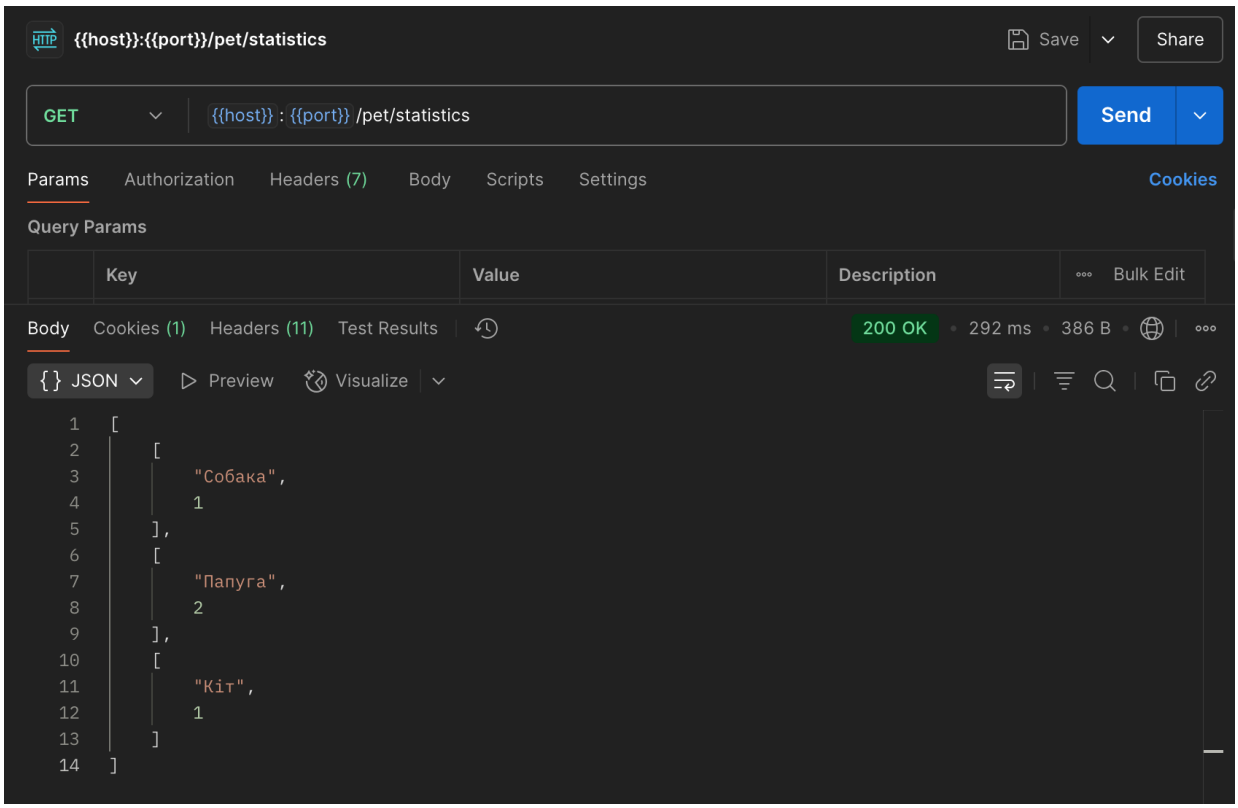

Рис.6 Демонстрація улюбленців та їх кількість, що зробили візит

ЗАГАЛЬНІ ВИСНОВКИ

У межах виконання даної роботи було реалізовано повнофункціональний вебзастосунок для ведення обліку ветеринарних візитів. Основною метою проєкту було створення системи, яка дозволяє реєструвати власників тварин, їхніх домашніх улюбленців, візити до ветеринара, проведені процедури та призначення медикаментів. Додатковою метою було впровадження автентифікації користувачів і розмежування прав доступу для адміністратора та звичайного користувача.

Розробку застосунку виконано з дотриманням принципів багаторівневої архітектури (MVC): створено моделі даних (entity-класи), контролери для обробки HTTP-запитів, сервіси для реалізації бізнес-логіки, а також репозиторії для взаємодії з базою даних. Реалізовано REST API з підтримкою основних CRUD-операцій для всіх ключових сутностей: Owner, Pet, Visit, Procedure, Prescription. Завдяки цьому застосунок може бути легко розширений або інтегрований з іншими клієнтами — наприклад, мобільним застосунком.

Для автентифікації та авторизації було впроваджено два підходи: класичний — через логін-форму, та сучасний — з використанням JWT (JSON Web Token). Адміністратор має доступ до керування всіма даними, в той час як звичайний користувач обмежений у правах доступу. Такий підхід дозволяє гнучко налаштовувати ролі та рівні доступу у майбутньому.

Розробка здійснювалася з використанням мови Java та фреймворку Spring Boot. Також було застосовано технології JPA/Hibernate для взаємодії з базою даних, Thymeleaf для шаблонів сторінок, Lombok для скорочення шаблонного коду, Postman — для перевірки REST-запитів, Swagger — для документування API. Було здійснено тестування всіх основних функцій застосунку, що дозволило виявити та виправити низку помилок, таких як некоректна обробка ролей користувачів, помилки в маршрутах та авторизації.

У ході роботи набуті практичні навички зі створення повнофункціонального CRUD-застосунку, проєктування архітектури багаторівневої системи, налаштування авторизації та безпеки у Spring, роботи з REST API, взаємодії з реляційними базами даних через ORM, а також навички налагодження,

тестування та документування API. Окремо варто відзначити важливість правильної організації коду за рівнями логіки — що суттєво полегшило масштабування та налагодження.

Загалом, розроблений застосунок задовольняє всі поставлені вимоги, реалізує основні функціональні сценарії, є масштабованим і придатним для подальшого розвитку як повноцінна система обліку ветеринарних візитів у клініці або ветеринарному центрі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spring Boot Documentation: веб-сайт. URL: https://docs.spring.io/spring-boot/index.html (дата звернення: 06.06.2025).

2. Spring Security Reference: веб-сайт. URL: https://docs.spring.io/spring-security/reference/ (дата звернення: 06.06.2025).

3. Introduction to JSON Web Tokens: веб-сайт. URL: https://jwt.io/introduction (дата звернення: 06.06.2025).