🐾 Straylove - Community Stray Animal Tracker

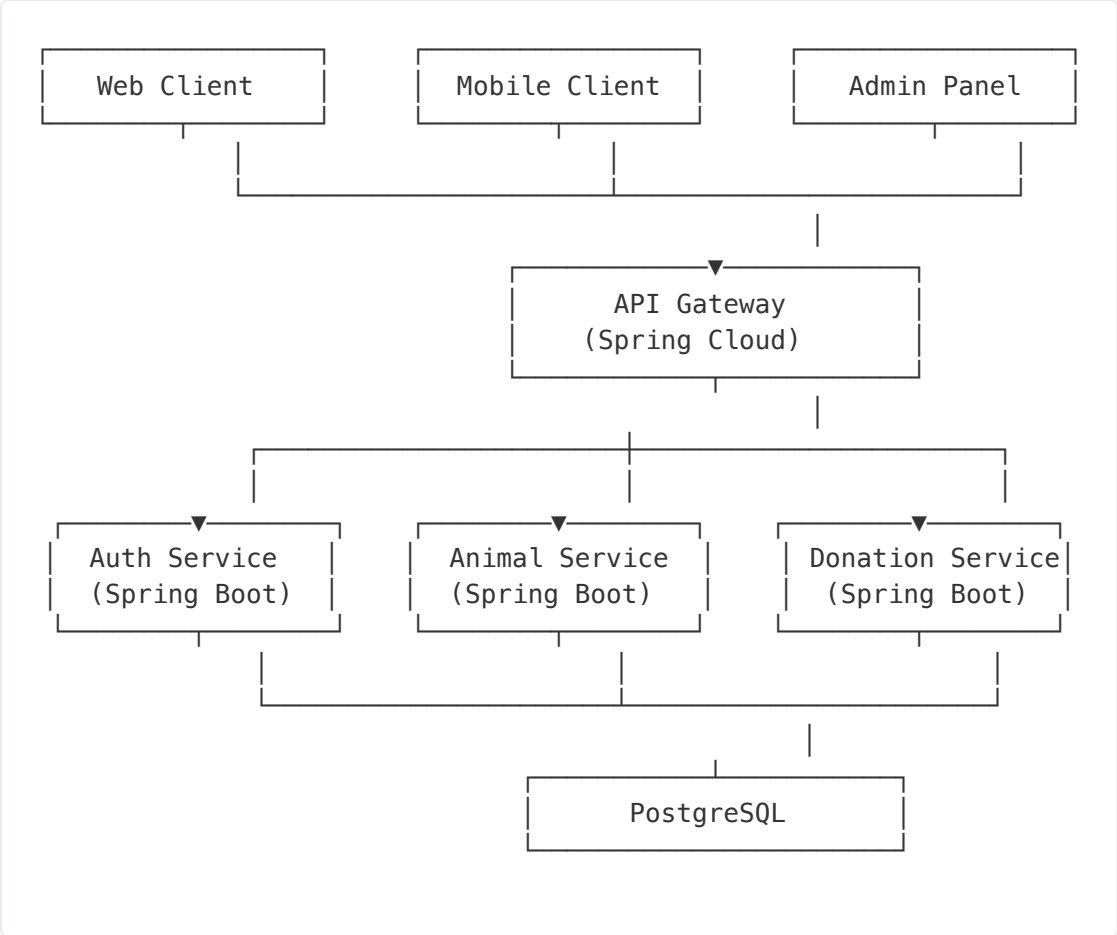
Low-Level Design Document

Version: 1.0
Date: June 2025
Author: System Architecture Team

Table of Contents

1. System Architecture

High-Level Architecture



Component Responsibilities

- **API Gateway:** Request routing, rate limiting, authentication
- **Auth Service:** User management, JWT generation, role management
- **Animal Service:** Stray animal tracking, care records, community logs
- **Donation Service:** Donation campaigns, payment processing

Technology Stack

Layer	Technology	Purpose

Backend Framework	Spring Boot 3.x	Core application framework
Security	Spring Security + JWT	Authentication & Authorization
Database	PostgreSQL	Primary data storage
ORM	Spring Data JPA	Database abstraction
Cache	Redis / Caffeine	Performance optimization
File Storage	Cloudinary	Image storage and CDN
Maps	Google Maps API	Location services
Documentation	OpenAPI 3.0 (Swagger)	API documentation

2. Database Design

Entity Relationship Overview

The database consists of 11 main tables with the following relationships:

- **users** - System users with roles (PUBLIC_USER, VOLUNTEER, ADMIN)
- **animals** - Core entity for tracking stray animals
- **animal_locations** - Location history for each animal
- **animal_images** - Multiple images per animal
- **care_records** - Vaccination, sterilization, and medical records
- **feeding_logs** - Daily feeding tracking
- **community_logs** - Community updates and alerts
- **log_upvotes** - Upvoting system for urgent logs
- **donation_campaigns** - Fundraising campaigns
- **donations** - Individual donation records

Core Tables Schema

Users Table

```
CREATE TABLE users (
    id BIGSERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(100),
    phone_number VARCHAR(20),
    role VARCHAR(20) NOT NULL CHECK (role IN ('PUBLIC_USER', 'VOLUNTE
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Animals Table

```
CREATE TABLE animals (
    id BIGSERIAL PRIMARY KEY,
    unique_identifier VARCHAR(50) UNIQUE NOT NULL,
    species VARCHAR(20) NOT NULL CHECK (species IN ('DOG', 'CAT', 'OT
    breed VARCHAR(50),
    color VARCHAR(50),
    approximate_age VARCHAR(20),
    gender VARCHAR(10) CHECK (gender IN ('MALE', 'FEMALE', 'UNKNOWN')
    temperament VARCHAR(20) CHECK (temperament IN ('FRIENDLY', 'AGGRE
    health_status VARCHAR(20) CHECK (health_status IN ('HEALTHY', 'IN
    is_vaccinated BOOLEAN DEFAULT false,
    is_sterilized BOOLEAN DEFAULT false,
    primary_image_url VARCHAR(500),
    reported_by BIGINT REFERENCES users(id),
    approved_by BIGINT REFERENCES users(id),
    approval_status VARCHAR(20) DEFAULT 'PENDING',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

> **Note:** All tables include proper foreign key constraints, indexes for performance, and audit fields for tracking changes.

3. Domain Models

Core Entity Classes

User Entity

```
 @Entity
@Table(name = "users")
@EntityListeners(AuditingEntityListener.class)
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false, length = 50)
    private String username;

    @Column(unique = true, nullable = false, length = 100)
    private String email;

    @Column(name = "password_hash", nullable = false)
    private String passwordHash;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    private UserRole role;

    @CreatedDate
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    // Additional fields, getters, setters...
}
```

Animal Entity

```
 @Entity
@Table(name = "animals")
public class Animal {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "unique_identifier", unique = true, nullable = fal
    private String uniqueIdentifier;

    @Enumerated(EnumType.STRING)
    private AnimalSpecies species;

    @OneToMany(mappedBy = "animal", cascade = CascadeType.ALL, fetch
    private List locations = new ArrayList<>();

    @OneToMany(mappedBy = "animal", cascade = CascadeType.ALL, fetch
    private List careRecords = new ArrayList<>();

    // Helper method
    public AnimalLocation getCurrentLocation() {
        return locations.stream()
            .filter(AnimalLocation::isCurrent)
            .findFirst()
            .orElse(null);
    }
}
```

Enumeration Types

- **UserRole:** PUBLIC_USER, VOLUNTEER, ADMIN
- **AnimalSpecies:** DOG, CAT, OTHER
- **Gender:** MALE, FEMALE, UNKNOWN
- **Temperament:** FRIENDLY, AGGRESSIVE, SHY, UNKNOWN
- **HealthStatus:** HEALTHY, INJURED, SICK, CRITICAL
- **CareType:** VACCINATION, STERILIZATION, FEEDING, MEDICAL_TREATMENT
- **LogType:** SIGHTING, CONCERN, UPDATE, ALERT

4. Service Layer Design

Animal Service

Core service for animal management with the following key methods:

Key Operations

```
 @Service
@Transactional
public class AnimalService {

    public AnimalResponseDto reportAnimal(AnimalReportDto reportDto,
                                          MultipartFile image,
                                          User reporter) {
        // 1. Generate unique identifier
        // 2. Upload image to Cloudinary
        // 3. Create animal entity with location
        // 4. Save to database
        // 5. Notify admins for approval
    }

    public Page searchAnimals(AnimalSearchCriteria criteria,
                                                  Pageable pageable) {
        // Dynamic search with specifications
        // Filter by species, area, health status, vaccination status
        // Only show approved animals to public users
    }

    public List getAnimalsNearLocation(BigDecimal latitude,
                                                     BigDecimal longi
                                                     double radiusKm)
        // 1. Calculate bounding box for initial filtering
        // 2. Query animals within bounding box
        // 3. Calculate exact distances using Haversine formula
        // 4. Filter by radius and return results
    }
}
```

Distance Calculation

```
 private double calculateDistance(double lat1, double lon1, double lat
    // Haversine formula implementation
    double R = 6371; // Earth's radius in km
    double dLat = Math.toRadians(lat2 - lat1);
    double dLon = Math.toRadians(lon2 - lon1);
    double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
               Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadia
               Math.sin(dLon/2) * Math.sin(dLon/2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    return R * c;
}
```

Care Service

Manages vaccination, sterilization, and feeding records:

- **recordCare():** Records vaccination/sterilization with automatic status updates
- **logFeeding():** Tracks daily feeding activities
- **getFeedingSchedule():** Returns feeding calendar for date range
- **getAnimalCareHistory():** Complete care timeline for an animal

Community Service

Handles community interaction features:

- **addCommunityLog():** Add sightings, concerns, or updates
- **upvoteLog():** Upvote urgent community logs
- **getUrgentLogs():** Retrieve high-priority logs

5. Repository Layer

Repository Interfaces

AnimalRepository

```
 @Repository
public interface AnimalRepository extends JpaRepository,
                                          JpaSpecificationExecutor {

    Optional findByUniqueIdentifier(String uniqueIdentifier);

    @Query("SELECT a FROM Animal a " +
           "JOIN a.locations l " +
           "WHERE l.isCurrent = true " +
           "AND l.latitude BETWEEN :minLat AND :maxLat " +
           "AND l.longitude BETWEEN :minLon AND :maxLon " +
           "AND a.approvalStatus = :status")
    List findAnimalsInBoundingBox(@Param("minLat") BigDecimal minLat,
                                          @Param("maxLat") BigDecimal
                                          @Param("minLon") BigDecimal
                                          @Param("maxLon") BigDecimal
                                          @Param("status") ApprovalSta
}
```

Custom Specifications

```
public class AnimalSpecifications {

    public static Specification hasSpecies(AnimalSpecies species) {
        return (root, query, cb) -> cb.equal(root.get("species"), spe
    }

    public static Specification inArea(String area) {
        return (root, query, cb) -> {
            Join locationJoin = root.join("locations", JoinType.LEFT)
            return cb.and(
                cb.equal(locationJoin.get("isCurrent"), true),
                cb.equal(locationJoin.get("area"), area)
            );
        };
    }

    public static Specification isApproved() {
        return (root, query, cb) ->
            cb.equal(root.get("approvalStatus"), ApprovalStatus.APPRO
    }
}
```

Query Optimization Strategies

- **Lazy Loading:** Used for all OneToMany relationships
- **Batch Fetching:** Configured for collections to prevent N+1 queries
- **Projections:** DTOs used for read-only queries
- **Indexes:** Strategic indexes on frequently queried columns

6. Controller Layer & API Design

RESTful API Endpoints

Animal Management APIs

Method	Endpoint	Description	Access
POST	/api/v1/animals	Report new stray animal	Authenticated
GET	/api/v1/animals	Search animals with filters	Public
GET	/api/v1/animals/nearby	Get animals near location	Public
GET	/api/v1/animals/{id}	Get animal details	Public
PUT	/api/v1/animals/{id}/approve	Approve animal report	Admin

Care Management APIs

Method	Endpoint	Description	Access
POST	/api/v1/animals/{id}/care/records	Record care activity	Volunteer/Admin
POST	/api/v1/animals/{id}/care/feeding	Log feeding activity	Authenticated
GET	/api/v1/animals/{id}/care/history	Get care history	Public
GET	/api/v1/animals/{id}/care/feeding/schedule	Get feeding schedule	Public

Request/Response Examples

```
// POST /api/v1/animals - Report Animal Request
{
    "species": "DOG",
    "breed": "Indian Pariah",
    "color": "Brown",
    "approximateAge": "2-3 years",
    "gender": "MALE",
    "temperament": "FRIENDLY",
    "healthStatus": "HEALTHY",
    "latitude": 28.6139,
    "longitude": 77.2090,
    "area": "Connaught Place",
    "city": "New Delhi",
    "landmark": "Near Metro Station Gate 3"
}

// Response
{
    "id": 123,
    "uniqueIdentifier": "D-234567-890",
    "species": "DOG",
    "primaryImageUrl": "https://res.cloudinary.com/...",
    "currentLocation": {
        "latitude": 28.6139,
        "longitude": 77.2090,
        "area": "Connaught Place",
        "city": "New Delhi"
    },
    "approvalStatus": "PENDING"
}
```

7. Security Design

Authentication & Authorization

JWT Token Structure

```
{
    "sub": "username",
    "iat": 1624360000,
    "exp": 1624446400,
    "roles": ["VOLUNTEER"],
    "userId": 123
}
```

Security Configuration

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
        http.csrf().disable()
            .authorizeHttpRequests((authz) -> authz
                .requestMatchers("/api/v1/auth/**").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/v1/animals/**"
                .requestMatchers("/api/v1/donations/campaigns").permi
                .anyRequest().authenticated()
            )
            .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELES

        http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthen
        return http.build();
    }
}
```

Role-Based Access Control

Role	Permissions

PUBLIC_USER	• Report animals • View approved animals • Log feeding activities • Add community logs • Donate to campaigns
VOLUNTEER	All PUBLIC_USER permissions plus: • Record vaccinations • Record sterilizations • Update animal health status • Add medical treatment records
ADMIN	All VOLUNTEER permissions plus: • Approve/reject animal reports • Manage users • View all reports (including pending) • Delete records • Access analytics dashboard

Security Best Practices Implemented

- **Password Security:** BCrypt hashing with salt
- **JWT Security:** Short expiration times, refresh tokens
- **Input Validation:** All inputs validated and sanitized
- **SQL Injection Prevention:** Parameterized queries via JPA
- **XSS Prevention:** Output encoding, Content-Security-Policy headers
- **Rate Limiting:** API rate limiting per user
- **HTTPS Only:** SSL/TLS enforcement

8. Integration Design

Cloudinary Integration

Image upload service for animal photos:

```
@Service
public class ImageUploadService {

    public String uploadImage(MultipartFile file, String folder) {
        Map options = new HashMap<>();
        options.put("folder", "straylove/" + folder);
        options.put("transformation", Arrays.asList(
            Map.of("width", 800, "height", 800, "crop", "limit"),
            Map.of("quality", "auto:good")
        ));

        Map uploadResult = cloudinary.uploader().upload(file.getBytes
        return (String) uploadResult.get("secure_url");
    }
}
```

Google Maps Integration

Geocoding service for address resolution:

```
@Service
public class MapsService {

    public GeocodeResult reverseGeocode(BigDecimal latitude, BigDecim
        String url = String.format(
            "https://maps.googleapis.com/maps/api/geocode/json?latlng
            latitude, longitude, apiKey
        );

        // Parse response and extract address components
        // Return structured address data
    }
}
```

Notification Service

Email and push notification system:

- **New Animal Report:** Notify admins for approval
- **Approval Status:** Notify reporter when approved/rejected
- **Care Updates:** Notify interested users about vaccinations/treatments
- **Urgent Alerts:** Push notifications for high-priority community logs
- **Feeding Reminders:** Daily reminders for regular feeders

Payment Gateway Integration (Mock)

Donation processing workflow:

```
@Service
public class PaymentService {

    public PaymentResponse processDonation(DonationRequest request) {
        // 1. Validate donation amount and campaign
        // 2. Create payment intent with gateway
        // 3. Process payment (mock implementation)
        // 4. Update campaign raised amount
        // 5. Send receipt via email
        // 6. Return transaction details
    }
}
```

9. Error Handling & Validation

Global Exception Handler

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity handleResourceNotFound(ResourceNotFoundExce
        ErrorResponse error = ErrorResponse.builder()
            .timestamp(LocalDateTime.now())
            .status(HttpStatus.NOT_FOUND.value())
            .error("Resource Not Found")
            .message(ex.getMessage())
            .path(getPath())
            .build();

        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(error
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity handleValidationExceptions(
            MethodArgumentNotValidException ex) {

        Map errors = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(error ->
            errors.put(error.getField(), error.getDefaultMessage())
        );

        ErrorResponse error = ErrorResponse.builder()
            .timestamp(LocalDateTime.now())
            .status(HttpStatus.BAD_REQUEST.value())
            .error("Validation Failed")
            .message("Invalid input parameters")
            .validationErrors(errors)
            .path(getPath())
            .build();

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(err
    }
}
```

Custom Validators

Input validation examples:

- **Phone Number Validator:** Validates format and length
- **Image File Validator:** Checks file type and size limits
- **Location Validator:** Ensures valid coordinates
- **Date Range Validator:** Validates date ranges for queries

Error Response Format

```
{
    "timestamp": "2025-06-29T10:30:00",
    "status": 400,
    "error": "Validation Failed",
    "message": "Invalid input parameters",
    "path": "/api/v1/animals",
    "validationErrors": {
        "species": "Species is required",
        "latitude": "Latitude must be between -90 and 90"
    }
}
```

Business Rule Validations

- Animals cannot be marked as sterilized without being vaccinated first
- Feeding logs cannot be future-dated
- Community logs must have minimum 10 characters description
- Users cannot upvote their own community logs
- Images must be under 10MB and in supported formats

10. Caching Strategy

Cache Configuration

```
 @Configuration
@EnableCaching
public class CacheConfig {

    @Bean
    public CacheManager cacheManager() {
        CaffeineCacheManager cacheManager = new CaffeineCacheManager(
        cacheManager.setCaffeine(caffeineCacheBuilder());
        cacheManager.setCacheNames(Arrays.asList(
            "animals",           // 10 min TTL
            "animalDetails",     // 10 min TTL
            "locations",         // 30 min TTL
            "careHistory",       // 5 min TTL
            "feedingSchedule",   // 15 min TTL
            "statistics"        // 1 hour TTL
        ));
        return cacheManager;
    }
}
```

Caching Strategies

Cache Type	Use Case	TTL	Eviction Policy
Animal List	Search results, nearby animals	10 minutes	Time-based + manual on update
Animal Details	Individual animal pages	10 minutes	On update/delete
Location Data	Area-based queries	30 minutes	Time-based
Statistics	Dashboard metrics	1 hour	Scheduled refresh

Cache Implementation Example

```
@Service
public class AnimalService {

    @Cacheable(value = "animalDetails", key = "#id")
    public AnimalDetailDto getAnimalDetails(Long id) {
        // Expensive database query with joins
        // This result will be cached
    }

    @CacheEvict(value = "animalDetails", key = "#animalId")
    public void updateAnimal(Long animalId, AnimalUpdateDto updateDto
        // Update logic
        // Cache will be cleared for this animal
    }

    @CacheEvict(value = "animalDetails", allEntries = true)
    @Scheduled(fixedDelay = 3600000) // 1 hour
    public void evictAllCaches() {
        // Periodic cache cleanup
    }
}
```

11. Application Configuration

application.yml

```
spring:
  application:
    name: straylove-backend

  datasource:
    url: jdbc:postgresql://localhost:5432/straylove
    username: ${DB_USERNAME:straylove}
    password: ${DB_PASSWORD:password}
    hikari:
      maximum-pool-size: 10
      minimum-idle: 5
      connection-timeout: 30000

  jpa:
    hibernate:
      ddl-auto: validate
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
        jdbc:
          batch_size: 25
        order_inserts: true
        order_updates: true

  redis:
    host: ${REDIS_HOST:localhost}
    port: ${REDIS_PORT:6379}
    timeout: 2000ms

  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB

cloudinary:
  cloud-name: ${CLOUDINARY_CLOUD_NAME}
  api-key: ${CLOUDINARY_API_KEY}
  api-secret: ${CLOUDINARY_API_SECRET}

google:
  maps:
    api-key: ${GOOGLE_MAPS_API_KEY}

jwt:
  secret: ${JWT_SECRET}
  expiration: 86400000 # 24 hours

management:
  endpoints:
    web:
```

```
    exposure:
      include: health,info,metrics,prometheus
```

12. Performance Optimizations

Database Optimizations

- **Connection Pooling:** HikariCP with optimized pool size
- **Query Optimization:** Strategic use of indexes and query hints
- **Batch Operations:** Batch inserts/updates for bulk operations
- **Lazy Loading:** Prevent N+1 queries with proper fetch strategies

Application-Level Optimizations

- **Async Processing:** Non-blocking operations for notifications
- **Response Compression:** GZIP compression for API responses
- **Pagination:** Default page size limits to prevent large queries
- **DTO Projections:** Return only required fields in responses

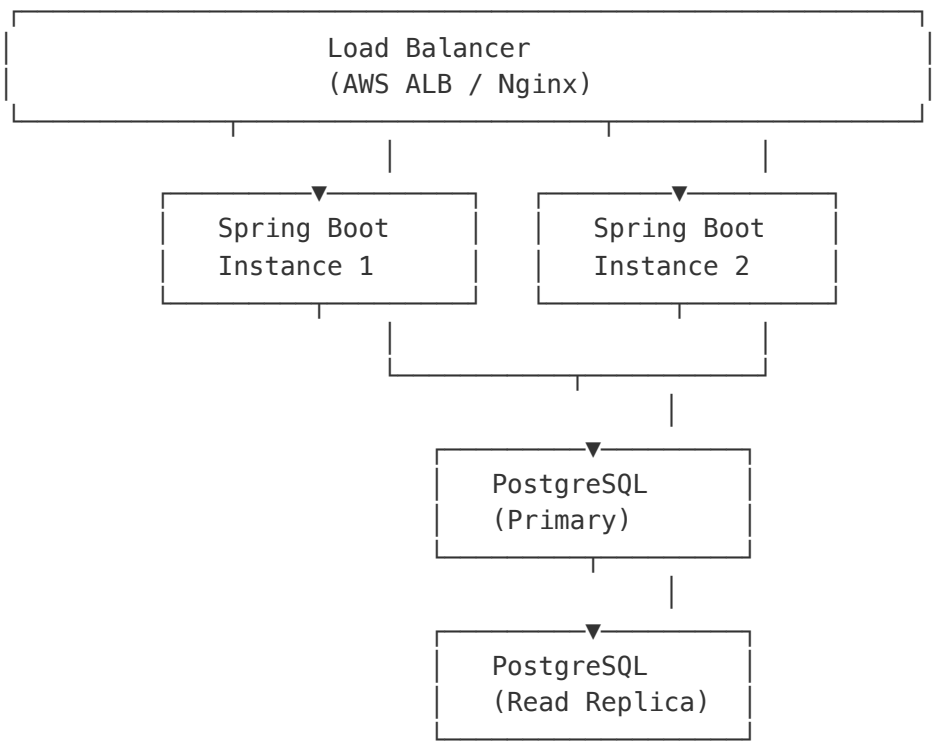
Monitoring & Metrics

Key metrics tracked:

- API response times (p50, p95, p99)
- Database query performance
- Cache hit/miss rates
- Error rates by endpoint
- Active user sessions
- Image upload success rates

13. Deployment Architecture

Production Environment



Infrastructure Components

- **Application Servers:** Multiple Spring Boot instances
- **Database:** PostgreSQL with read replicas
- **Cache:** Redis cluster for distributed caching
- **File Storage:** Cloudinary CDN for images
- **Monitoring:** Prometheus + Grafana
- **Logging:** ELK Stack (Elasticsearch, Logstash, Kibana)

CI/CD Pipeline

1. Code push to Git repository
2. Automated tests execution (unit, integration)

3. Code quality analysis (SonarQube)
4. Docker image build
5. Push to container registry
6. Deploy to staging environment
7. Run E2E tests
8. Deploy to production (blue-green deployment)

14. Future Enhancements

Planned Features

- **AI-Based Animal Recognition:** Identify animals from photos using ML
- **Volunteer Matching:** Match volunteers with nearby animals needing care
- **Adoption Module:** Facilitate adoption process through the platform
- **Real-time Tracking:** GPS collar integration for high-risk animals
- **Multi-language Support:** Localization for regional languages
- **Analytics Dashboard:** Advanced insights for NGOs and authorities

Technical Improvements

- **GraphQL API:** Alternative to REST for mobile clients
- **WebSocket Support:** Real-time updates for urgent alerts
- **Microservices Migration:** Split into smaller, independent services
- **Event Sourcing:** Complete audit trail of all changes
- **Kubernetes Deployment:** Container orchestration for better scaling

15. Conclusion

The Straylove platform's Low-Level Design provides a comprehensive solution for community-driven stray animal welfare. The architecture ensures:

- **Scalability:** Horizontal scaling capabilities with stateless services
- **Security:** Multi-layered security with role-based access control
- **Performance:** Optimized queries, caching, and async processing
- **Maintainability:** Clean architecture with separation of concerns
- **Extensibility:** Modular design allowing easy feature additions

The system successfully addresses the core problem of unorganized stray animal care by providing a centralized platform for tracking, care management, and community collaboration.

> **Document Version:** 1.0
> **Last Updated:** June 2025
> **Next Review:** September 2025