Vet Clinic Platform - Technical Specification

1. Architecture Overview

Multi-Repository Microservices Architecture

The platform follows a distributed architecture where each component is maintained as an independent repository. This approach provides maximum flexibility, independent deployment capabilities, and clear separation of concerns across the entire system.

The architecture consists of six independent repositories that work together to form the complete veterinary clinic platform. Two core repositories serve as shared libraries, while four application repositories consume these libraries to provide specific functionality.

2. Technology Stack

Frontend Technologies

- **Web Application**: React 18+ with TypeScript for the main web interface
- **Mobile Application**: React Native with Expo for cross-platform mobile experience
- **State Management**: Redux Toolkit with RTK Query for efficient API state management
- **UI Framework**: Tailwind CSS with Headless UI components for consistent design
- **Build Tools**: Vite for fast development and optimized production builds

Backend Technologies

- **API Framework**: FastAPI for high-performance REST API development
- **Programming Language**: Python 3.11+ for all backend services
- **Validation**: Pydantic v2 for robust data validation and serialization
- **Authentication**: Clerk integration with Google OAuth for secure user management

Database and Storage

- **Primary Database**: PostgreSQL 15+ hosted on Supabase for reliability and scalability
- **Cache Layer**: Redis 7+ for session management and performance optimization
- **ORM**: SQLAlchemy 2.0 with async support for database operations

- **File Storage**: Supabase Storage for images and documents

Background Processing

- **Task Queue**: Celery 5.3+ for asynchronous job processing
- **Message Broker**: Redis for task distribution and result storage
- **Monitoring**: Flower for task monitoring and management

Infrastructure and DevOps

- **Containerization**: Docker for consistent development and deployment environments
- **Frontend Hosting**: Vercel for optimized React application deployment
- **Backend Hosting**: Railway or DigitalOcean for API and worker services
- **CI/CD**: GitHub Actions for automated testing and deployment
- **Monitoring**: Sentry for error tracking and performance monitoring

3. Multi-Repository Structure

Core Package Repositories

vet-core-package Repository

This repository contains the foundational data models and database utilities that are shared across all services. It includes SQLAlchemy models for users, pets, appointments, clinics, and other core entities. The package also provides Pydantic schemas for data validation, database connection utilities, and common helper functions. This package is published to PyPI and serves as the single source of truth for data structures.

vet-services-package Repository

This repository houses all business logic services that can be reused across different applications. It includes authentication services for Clerk integration, notification services for email and SMS, payment processing utilities, file storage handlers, and third-party API integrations. The package depends on vet-core and is also published to PyPI for easy consumption by other services.

Application Repositories

vet-clinic-api Repository

The main FastAPI application that provides REST API endpoints for the entire platform. This repository imports both core packages and implements all API routes, authentication middleware, request validation, and response formatting. It serves as the primary interface between frontend applications and the database.

vet-clinic-background-jobs Repository

A dedicated Celery worker application that handles all asynchronous tasks. This includes sending appointment reminders, processing payment notifications, generating reports, and performing database maintenance tasks. The service imports both core packages to access models and business logic.

vet-clinic-web Repository

The React-based web application that provides the main user interface for pet owners, veterinarians, and clinic administrators. This repository is completely independent and communicates with the backend through API calls. It includes responsive design, real-time notifications, and comprehensive user workflows.

vet-clinic-mobile Repository

The React Native mobile application that provides native mobile experience for iOS and Android platforms. This repository shares similar functionality with the web app but is optimized for mobile interactions and includes platform-specific features like push notifications and camera integration.

4. Database Design Strategy

Core Entity Relationships

The database schema centers around core entities including users, pets, appointments, clinics, and veterinarians. Users can have multiple roles (pet owner, veterinarian, admin) with appropriate permission levels. Pets belong to users and can have multiple appointments with different veterinarians across various clinics.

Performance Optimization

The database design includes strategic indexing on frequently queried fields such as appointment dates, user emails, and pet owner relationships. Foreign key relationships ensure data integrity while allowing for efficient joins across related tables.

Data Security and Privacy

All sensitive information is properly encrypted and access is controlled through role-based permissions. The schema supports audit trails for critical operations and maintains compliance with veterinary data protection requirements.

5. API Architecture

RESTful Design Principles

The API follows REST conventions with clear resource-based URLs, appropriate HTTP methods, and consistent response formats. All endpoints return standardized JSON responses with proper status codes and error handling.

Authentication and Authorization

Every API endpoint requires authentication through Clerk JWT tokens. Role-based access control ensures that users can only access and modify data they have permission to handle. The system supports different user types with varying levels of access.

Rate Limiting and Security

API endpoints implement rate limiting to prevent abuse and ensure fair usage. All requests are validated for proper format and authorization before processing. The API includes comprehensive error handling and logging for monitoring and debugging.

6. Authentication Architecture

Clerk Integration Strategy

The platform uses Clerk as the primary authentication provider, supporting Google OAuth and other social login methods. Clerk handles user registration, login, password management, and session handling, reducing the complexity of authentication management.

User Synchronization

When users authenticate through Clerk, their information is synchronized with the local database to maintain consistent user profiles and enable efficient querying. The system maintains a mapping between Clerk user IDs and internal user records.

Role Management

The platform implements role-based access control where users can be pet owners, veterinarians, clinic administrators, or system administrators. Each role has specific permissions and access levels throughout the application.

7. Background Processing Architecture

Asynchronous Task Management

Celery workers handle all time-intensive and scheduled operations including sending email notifications, processing payment webhooks, generating monthly reports, and performing database cleanup tasks. This ensures the main API remains responsive for user interactions.

Task Reliability and Monitoring

Background tasks include retry mechanisms for failed operations and comprehensive logging for debugging. The system uses Flower for monitoring task execution and identifying performance bottlenecks.

Scheduled Operations

Regular maintenance tasks such as appointment reminders, vaccination notifications, and system health checks run on predefined schedules. The system ensures these operations don't interfere with peak usage periods.

8. Shared Package Strategy

Package Publishing and Versioning

Both core packages are published to PyPI with semantic versioning to ensure compatibility across different services. Version updates follow a coordinated release process to maintain consistency across all consuming applications.

Development Workflow

During development, packages can be installed in editable mode for rapid iteration. The system supports local development while ensuring production deployments use stable, published versions of the packages.

Dependency Management

Each application repository specifies exact version ranges for core packages to ensure compatibility and enable controlled upgrades. The dependency chain is carefully managed to prevent version conflicts.

9. Development Strategy

Repository Independence

Each repository maintains its own development lifecycle, testing suite, and deployment pipeline. Teams can work independently while coordinating on shared package updates that affect multiple services.

Local Development Environment

Developers can set up complete local environments using Docker Compose for shared infrastructure (PostgreSQL, Redis) while running individual services locally for rapid development and testing.

Cross-Repository Coordination

When features span multiple repositories, the development process follows a coordinated approach: first updating core packages, then publishing new versions, and finally updating consuming applications to use the new functionality.

10. Deployment Strategy

Independent Service Deployment

Each repository has its own deployment pipeline, allowing for independent releases and rollbacks. This approach minimizes the impact of changes and enables rapid iteration on individual components.

Environment Management

The platform supports multiple environments (development, staging, production) with appropriate configuration management for each repository. Environment-specific variables are managed securely through deployment platforms.

Scalability and Monitoring

Each service can be scaled independently based on demand. Comprehensive monitoring across all repositories ensures system health and enables proactive issue resolution.

11. Security and Compliance

Data Protection

The platform implements comprehensive data protection measures including encryption at rest and in transit, secure API endpoints, and proper access controls. All sensitive information is handled according to veterinary industry standards.

Monitoring and Logging

Comprehensive logging across all services enables audit trails and security monitoring. Error tracking and performance monitoring help identify and resolve issues quickly.

Backup and Recovery

Database backups and disaster recovery procedures ensure data safety and business continuity. The distributed architecture provides natural resilience against single points of failure.

12. Performance and Scalability

Horizontal Scaling

The microservices architecture enables horizontal scaling of individual components based on demand. API services, background workers, and frontend applications can be scaled independently.

Caching Strategy

Redis provides multiple levels of caching including session data, frequently accessed database queries, and API response caching. This significantly improves response times and reduces database load.

Database Performance

Strategic indexing, connection pooling, and query optimization ensure efficient database operations. The system is designed to handle growing amounts of veterinary data and user interactions.

This technical specification provides the foundation for building a scalable, maintainable, and secure veterinary clinic platform using modern development practices and proven architectural patterns.