# FindMyPet - Technical Requirements Document (TRD)

## Document Information

* **Document Title:** Technical Requirements Document
* **Project Name:** FindMyPet
* **Date:** May 11, 2025
* **Version:** 1.0

## Introduction

This Technical Requirements Document (TRD) outlines the technical specifications, architecture, and implementation details for the FindMyPet platform. The document is intended for the development team, system architects, QA engineers, and technical stakeholders involved in the project.

## System Overview

FindMyPet is a web-based application that helps pet owners find their lost pets by creating listings that can be searched by the community. The application consists of a backend API server built with Flask and a frontend client built with React. The system incorporates geolocation services, user authentication, real-time messaging, and image handling capabilities.

## Technical Architecture

### High-Level Architecture

The FindMyPet application follows a client-server architecture with the following major components:

1. **Frontend Client**
   * React-based single-page application
   * Responsive design using Bootstrap 5
   * Client-side routing with React Router
   * State management with React Context API
2. **Backend API Server**
   * Flask-based RESTful API
   * JWT authentication
   * SQLAlchemy ORM for database operations
   * File handling for image uploads
3. **Database**
   * SQLite (development)
   * PostgreSQL (production, future implementation)
   * Data models for users, pets, messages, and comments
4. **External Services**
   * OpenStreetMap for mapping
   * Leaflet for interactive maps
   * Email notification service (future implementation)

### System Architecture Diagram

+---------------+ HTTP/HTTPS +----------------+  
| |<------------------->| |  
| React Client | | Flask Server |  
| | | |  
+---------------+ +----------------+  
 ^ ^  
 | |  
 v v  
+---------------+ +----------------+  
| | | |  
| External | | Database |  
| Services | | (SQLite/ |  
| (Maps, etc.) | | PostgreSQL) |  
+---------------+ +----------------+

## Frontend Technical Requirements

### Technologies

* **Framework:** React 18+
* **Styling:** Bootstrap 5, CSS3
* **Routing:** React Router 6+
* **State Management:** React Context API
* **HTTP Client:** Axios
* **Mapping:** Leaflet, React-Leaflet
* **Form Handling:** React Hook Form (future implementation)
* **Build Tool:** Create React App, Webpack

### Frontend Components

1. **Authentication Components**
   * Login
   * Registration
   * Password Reset (future implementation)
   * Profile Management
2. **Pet Management Components**
   * Create/Edit Pet Listing
   * Pet List View
   * Pet Detail View
   * Filter and Search Interface
3. **Mapping Components**
   * Interactive Map View
   * Location Picker
   * Map Markers for Pet Locations
4. **Communication Components**
   * Messaging Interface
   * Comment System
   * Notification Center (future implementation)
5. **Layout Components**
   * Navigation Bar
   * Footer
   * Responsive Sidebar (future implementation)
   * Error Boundaries

### Frontend Performance Requirements

* Initial load time under 3 seconds on standard broadband connections
* Time to interactive under 4 seconds
* Responsive design supporting viewports from 320px to 2560px width
* Graceful degradation for older browsers
* Lazy loading for images and heavy components

## Backend Technical Requirements

### Technologies

* **Framework:** Flask 2.0+
* **API Style:** RESTful
* **Authentication:** JWT (JSON Web Tokens)
* **Database ORM:** SQLAlchemy
* **Image Processing:** Pillow
* **File Storage:** Local filesystem (S3 in production, future implementation)
* **Validation:** Marshmallow or Flask-WTF
* **Testing:** Pytest

### API Endpoints

1. **Authentication Endpoints**
   * POST /api/auth/register - User registration
   * POST /api/auth/login - User login
   * GET /api/auth/profile - Get user profile
   * PUT /api/auth/profile - Update user profile
2. **Pet Endpoints**
   * GET /api/pets - List all pets with filters
   * POST /api/pets - Create new pet listing
   * GET /api/pets/<id> - Get pet details
   * PUT /api/pets/<id> - Update pet listing
   * DELETE /api/pets/<id> - Delete pet listing
   * GET /api/pets/user - Get current user’s pet listings
3. **Communication Endpoints**
   * POST /api/messages - Send a message
   * GET /api/messages - Get user’s messages
   * POST /api/comments - Post a comment
   * GET /api/pets/<id>/comments - Get comments for a pet listing
4. **Search Endpoints**
   * GET /api/search - Search pets by criteria
   * GET /api/map - Get geolocation data for map

### Database Schema

#### Users Table

users  
- id (Primary Key)  
- username (Unique)  
- email (Unique)  
- password\_hash  
- phone (Optional)  
- created\_at  
- updated\_at  
- last\_login  
- is\_admin (Boolean)

#### Pets Table

pets  
- id (Primary Key)  
- user\_id (Foreign Key -> users.id)  
- title  
- description  
- pet\_type (cat, dog, bird, other)  
- status (missing, found, reunited)  
- last\_seen\_address  
- last\_seen\_date  
- latitude (Optional)  
- longitude (Optional)  
- image\_url (Optional)  
- created\_at  
- updated\_at

#### Messages Table

messages  
- id (Primary Key)  
- sender\_id (Foreign Key -> users.id)  
- recipient\_id (Foreign Key -> users.id)  
- pet\_id (Foreign Key -> pets.id)  
- content  
- read (Boolean)  
- created\_at

#### Comments Table

comments  
- id (Primary Key)  
- user\_id (Foreign Key -> users.id)  
- pet\_id (Foreign Key -> pets.id)  
- content  
- created\_at  
- updated\_at

### Security Requirements

* HTTPS for all communications
* Password hashing using bcrypt
* JWT with appropriate expiration
* CSRF protection
* Input validation and sanitization
* Rate limiting for API endpoints
* XSS protection
* SQL injection prevention

## Deployment and DevOps

### Development Environment

* Local development using Docker containers
* Version control with Git
* Development database: SQLite
* Environment variables for configuration

### Testing Strategy

* Unit tests for backend services and API endpoints
* Integration tests for critical user flows
* Frontend component tests with React Testing Library
* End-to-end tests with Cypress (future implementation)
* Test coverage target: 70%+

### Deployment Pipeline

* CI/CD using GitHub Actions
* Staging environment for QA testing
* Production deployment with zero downtime
* Database migrations managed with Alembic

### Hosting Requirements

* Backend: Virtual private server or container service
* Frontend: Static file hosting with CDN
* Database: Managed database service (production)
* File storage: Object storage service (production)
* SSL certificate management
* Daily database backups

## Performance Requirements

### Backend Performance

* API response time under 300ms for 95% of requests
* Support for 100+ concurrent users
* Database query optimization
* Efficient file handling and image processing
* Caching strategy for frequently accessed data

### Scalability Considerations

* Horizontal scaling capability for backend services
* Database connection pooling
* Load balancing for production deployment
* Content delivery network for static assets
* Stateless authentication for distributed systems

## Integration Requirements

### Third-party Integrations (Future)

* Email service provider for notifications
* Social media authentication (OAuth)
* Payment gateway for premium features
* SMS notification service
* Pet microchip database integration

### API Documentation

* OpenAPI/Swagger specification
* Endpoint documentation with examples
* Authentication and error handling documentation
* Rate limiting and usage guidelines

## Maintenance and Support

### Monitoring

* Application performance monitoring
* Error tracking and logging
* User behavior analytics
* Server health monitoring
* Database performance monitoring

### Backup and Recovery

* Daily automated backups
* Point-in-time recovery capability
* Backup verification procedures
* Disaster recovery plan

### Update and Patch Management

* Regular security updates
* Dependency management and updating
* Feature release schedule
* Backward compatibility considerations

## Implementation Plan

### Phase 1: MVP Development

* Core user authentication
* Basic pet listing functionality
* Simple search and filtering
* Essential map integration
* Basic messaging system

### Phase 2: Enhanced Features

* Advanced search capabilities
* Comment system
* Image optimization
* User profile enhancements
* Performance optimizations

### Phase 3: Advanced Features

* Mobile app development
* Push notifications
* Social sharing integration
* Analytics implementation
* Premium feature implementation

## Technical Risks and Mitigation

| Risk | Impact | Probability | Mitigation Strategy |
| --- | --- | --- | --- |
| Database performance issues with scaling | High | Medium | Implement proper indexing, query optimization, and consider sharding for large data sets |
| Image storage and delivery bottlenecks | Medium | Medium | Use CDN for image delivery, implement image optimization and caching |
| API rate limiting and abuse | High | Medium | Implement rate limiting, request throttling, and monitoring for unusual patterns |
| Authentication security vulnerabilities | High | Low | Regular security audits, following best practices, keeping dependencies updated |
| Cross-browser compatibility issues | Medium | Medium | Comprehensive testing across browsers, progressive enhancement approach |

## Approval

This document requires review and approval from key technical stakeholders before proceeding with implementation.

| Stakeholder | Role | Signature | Date |
| --- | --- | --- | --- |
| [Name] | Technical Lead |  |  |
| [Name] | Backend Developer |  |  |
| [Name] | Frontend Developer |  |  |
| [Name] | DevOps Engineer |  |  |
| [Name] | QA Engineer |  |  |