Zoo Management System (ZMS)

Document Date: 29-12-2025

Complete Technical Documentation for Zoo Management System including Frontend, Backend, Database Architecture and Communication Flow.

Table of Contents

1. PROJECT OVERVIEW

Zoo Management System (ZMS) is a comprehensive web application designed to manage all operations of a zoo. It handles animal records, employee management, veterinary care, ticket sales, visitor information, and inventory management. The system serves multiple user types including zoo staff, veterinarians, and public visitors.

Key Features:
• Complete animal record management
• Cage planning and allocation
• Employee and veterinarian management
• Medical checkups and vaccinations
• Ticket sales and visitor tracking
• Inventory management
• Event planning and management

2. TECHNOLOGY STACK

Layer	Technology	Version & Details
Frontend Framework	React	18.3.1 with JSX
Frontend Language	TypeScript	5.5.4
Routing	React Router	6.26.2
Styling	Tailwind CSS	3.4.17
Build Tool	Vite	5.2.0
Backend Framework	Express.js	4.21.1
Backend Runtime	Node.js	CommonJS
Database	SQLite	better-sqlite3 11.5.0
Backend Language	TypeScript	5.5.4
CORS Handling	cors package	2.8.5
Icons Library	lucide-react	0.522.0

3. PROJECT ARCHITECTURE

Directory Structure:

zms/ (Root Directory)
■■■ src/ (Frontend)
■ ■■■ components/ (UI Components)
■ ■ ■■■ ui/ (Basic UI Components)
■ ■■■ pages/ (Pages)
■ ■ ■■■ public/ (Public Pages)
■ ■ ■■■ staff/ (Staff Pages)
■ ■■■ contexts/ (React Context State)
■ ■■■ utils/ (Utility Functions)
■ ■■■ types/ (TypeScript Types)
■■■ backend/src/ (Backend)
■ ■■■ db/ (Database Setup)
■ ■■■ routes/ (API Routes)
■ ■■■ services/ (Business Logic)
■ ■■■ utils/ (Utilities)
■ ■■■ types/ (Type Definitions)
■■■ public/ (Static Assets)
■■■ Configuration Files (package.json, tsconfig.json, etc.)

4. DATABASE DESIGN

Database is SQLite with Foreign Key constraints enabled for data integrity.

Table Name	Purpose	Key Columns
users	User authentication	id, name, email, password, role
animals	Animal records	id, name, species, age, gender, health_status, cage_id
cages	Cage information	id, name, type, capacity, occupancy, location, status
employees	Employee records	id, name, email, role, phone, salary, join_date
doctors	Veterinarian info	id, name, specialization, email, phone, availability
medical_checks	Medical checkups	id, animal_id, doctor_id, date, diagnosis, treatment
vaccinations	Vaccination records	id, animal_id, vaccine_name, date_administered, next_due_date
events	Event management	id, title, description, date, time, location, capacity
tickets	Ticket types	id, type, price, description
ticket_sales	Sales records	id, ticket_id, quantity, total_amount, date, visitor_name
visitors	Visitor info	id, name, email, phone, registration_date

inventory	Inventory items	id, name, category, quantity, unit, min_threshold, expiry_date

5. BACKEND API ENDPOINTS

A. Authentication Endpoints

Method	Endpoint	Purpose
POST	/api/auth/register	Register new user
POST	/api/auth/login	User login

B. Animal Endpoints

GET	/api/animals	Get all animals
POST	/api/animals	Add new animal
PUT	/api/animals/:id	Update animal
DELETE	/api/animals/:id	Delete animal

C. Other Core Endpoints

Cages: GET /api/cages, POST /api/cages, PUT /api/cages/:id, DELETE /api/cages/:id
Employees: GET /api/employees, POST /api/employees, PUT /api/employees/:id, DELETE /api/employees/:id
Doctors: GET /api/doctors, POST /api/doctors, PUT /api/doctors/:id, DELETE /api/doctors/:id
Vaccinations: GET /api/vaccinations, POST /api/vaccinations, PUT /api/vaccinations/:id, DELETE /api/vaccinations/:id
Medical Checks: GET /api/medical-checks, POST /api/medical-checks, PUT /api/medical-checks/:id, DELETE /api/medical-checks/:id
Events: GET /api/events, POST /api/events, PUT /api/events/:id, DELETE /api/events/:id
Tickets: GET /api/tickets, POST /api/tickets, PUT /api/tickets/:id, DELETE /api/tickets/:id
Ticket Sales: GET /api/ticket-sales, POST /api/ticket-sales
Visitors: GET /api/visitors, POST /api/visitors
Inventory: GET /api/inventory, POST /api/inventory, PUT /api/inventory/:id, DELETE /api/inventory/:id
Health Check: GET /api/health

6. FRONTEND STRUCTURE

A. Main Pages

Public Pages:
• LandingPage - Main zoo homepage
• RegisterPage - New user registration
• LoginPage - User login
• EventsPage - Events listing
• TicketPurchasePage - Ticket booking

Staff Pages (Protected):
• DashboardPage - Staff dashboard
• AnimalsPage - Animal management
• CagesPage - Cage management
• EmployeesPage - Employee management
• DoctorsPage - Veterinarian management
• MedicalChecksPage - Medical records
• VaccinationsPage - Vaccination records
• SalariesPage - Salary management
• InventoryPage - Inventory management
• TicketSalesPage - Sales tracking
• EventManagementPage - Event management
• VisitorsPage - Visitor records

B. UI Components

Basic Components: Button, Input, Select, Modal, Table, Card, Badge, StatCard, Navbar, FeatureCard
Special Components: ProtectedRoute (for access control), AnimalShape (custom SVG)

C. State Management

AuthContext: Manages user authentication state globally. Provides user information, login/logout functions to all components. Enables ProtectedRoute to restrict access.

7. FRONTEND AND BACKEND COMMUNICATION

Communication Protocol

Frontend communicates with Backend via HTTP REST API. The api.ts utility file contains all API functions. Communication flow: 1. User interacts with React component 2. Component calls api function (e.g., api.getAnimals()) 3. HTTP request sent to Backend (GET http://localhost:3001/api/animals) 4. Express server receives and processes request 5. Backend queries SQLite database 6. Response sent back as JSON 7. Frontend receives and displays data Base URL: http://localhost:3001/api Default Port: 3001 (configurable via PORT env variable)

Data Flow Example - Getting Animals

```
Frontend Request: fetch('http://localhost:3001/api/animals', { method: 'GET', headers: { 'Content-Type': 'application/json' } }) Backend Handler: app.get('/api/animals', (req, res) => { const animals = db.prepare('SELECT * FROM animals').all(); res.json(keysToCamel(animals)); }); Response: [ { id: 'ani-1', name: 'Leo', species: 'Lion', ... }, { id: 'ani-2', name: 'Ella', species: 'Elephant', ... } ]
```

CORS and Middleware

Backend enables CORS to allow cross-origin requests from frontend. Middleware components: • cors() - Enables cross-origin requests • express.json() - Parses JSON request bodies • Request logger - Logs all incoming requests • Error handlers - Returns appropriate error responses Data Mapping: Frontend uses camelCase (firstName), Database uses snake_case (first_name). mapper.ts utility converts between formats automatically.

8. USER FLOWS

Flow 1: User Registration

1. User opens RegisterPage 2. Enters name, email, password, and role 3. Clicks submit button 4. Frontend calls api.register(userData) 5. POST request sent to /api/auth/register 6. Backend validates and inserts user into database 7. Success response returned 8. User redirected to login page

Flow 2: User Login

1. User opens LoginPage 2. Enters email and password 3. Clicks login button 4. Frontend calls api.login(credentials) 5. POST request sent to /api/auth/login 6. Backend queries database and verifies credentials 7. If valid, returns user data (without password) 8. Frontend stores in AuthContext 9. User redirected to dashboard

Flow 3: Adding Animal

1. Staff opens AnimalsPage 2. Clicks "Add Animal" button 3. Modal opens with form 4. Staff fills animal details 5. Clicks save 6. Frontend calls api.createAnimal(data) 7. POST request sent to /api/animals 8. Backend generates ID and inserts record 9. Cage occupancy updated 10. Response returns new animal 11. Frontend adds to list and refreshes display

Flow 4: Ticket Purchase

1. Visitor opens TicketPurchasePage 2. Selects ticket type and quantity 3. Enters personal information 4. Clicks purchase 5. Frontend calls api.createTicketSale(data) 6. POST request sent to /api/ticket-sales 7. Backend creates sales record 8. Generates receipt 9. Response returns confirmation 10. Frontend displays confirmation with details

9. SECURITY MEASURES

Current Security Features

✓ Role-based Access Control (RBAC) - Different users have different permissions ✓ ProtectedRoute Component - Restricts unauthorized access to staff pages ✓ CORS Configuration - Controls which origins can access API ✓ SQLite Foreign Keys - Maintains data referential integrity ✓ Input validation - Basic validation through frontend ✓ HTTP Methods - Proper use of GET, POST, PUT, DELETE ✓ Error handling - Server returns appropriate error codes

Recommended Security Improvements

■■ CRITICAL IMPROVEMENTS NEEDED: 1. PASSWORD HASHING: Use bcrypt instead of plain-text passwords - Current: password stored as plain text - Recommended: Use bcrypt with salt rounds 2. AUTHENTICATION TOKENS: Implement JWT (JSON Web Tokens) - Current: No session/token system - Recommended: JWT for stateless authentication 3. HTTPS/SSL: Use HTTPS in production - Current: HTTP only - Recommended: Force HTTPS, use certificates 4. INPUT VALIDATION: Validate all inputs on server - Current: Minimal validation - Recommended: Schema validation (joi, zod) 5. RATE LIMITING: Prevent brute force attacks - Current: None - Recommended: Implement rate limiting middleware 6. SQL INJECTION PREVENTION: Already using prepared statements (good!) - Continue using parameterized queries 7. ENVIRONMENT VARIABLES: Protect sensitive data - Current: Some config hardcoded - Recommended: All secrets in .env file 8. AUDIT LOGGING: Track all data modifications - Current: Basic logging only - Recommended: Complete audit trail

10. CONCLUSION AND SUMMARY

Zoo Management System is a modern, scalable web application built with React, Express, and SQLite. It successfully demonstrates: ✓ Full-stack development capabilities ✓ Proper separation of concerns (Frontend/Backend) ✓ RESTful API design principles ✓ Database relationships and constraints ✓ User authentication and role-based access ✓ Responsive UI with Tailwind CSS ✓ TypeScript for type safety The application serves multiple user types with appropriate access levels and provides comprehensive management tools for zoo operations including animal care, staff management, inventory control, and visitor services.

System Performance:
• Frontend: React with Vite for fast development/builds
• Backend: Express on Node.js for high performance
• Database: SQLite for quick queries and local storage
• Deployment: Can be containerized with Docker

Future Enhancements:
→ Advanced reporting and analytics
→ Email/SMS notifications
→ Mobile application
→ Payment gateway integration
→ Real-time notifications with WebSockets
→ Multi-language support
→ API documentation (Swagger/OpenAPI)
→ Automated backups
→ User audit trail
→ Advanced search and filtering

Document Information:
Generated: 29-12-2025
Version: 1.0
Status: Complete and Current
Last Updated: 29-12-2025