DOCUMENT 2: SYSTEM ARCHITECTURE & TECHNICAL DESIGN

1. Architectural Topology: Lightweight Modular Monolith

The Zhailau platform is designed for maximum portability and speed. By removing heavy frontend frameworks and rigid SQL schemas, we achieve a "lean" architecture that is easy to deploy and scales horizontally.

Top-Level Topology:

`[Client (HTML5/ES6)]` -> `[REST API (Go)]` -> `[MongoDB Cluster]`

- **Frontend Strategy: "Vanilla" Architecture**. We utilize native browser standards (ES6 Modules, CSS Variables) to eliminate build-step complexity (No Webpack/React overhead). This ensures faster load times on low-end devices in regional areas.
- **Backend Logic: Golang**. Acts as a high-throughput JSON processor, serving data directly from MongoDB to the client without complex ORM transformations.
- **Persistence Layer: MongoDB (NoSQL)**. Chosen for its **Schemaless Flexibility**, allowing us to store diverse animal data (e.g., a horse has different medical attributes than a cow) without altering database tables.

2. Tech Stack Specification

Layer	Technology	Justification
Frontend	**HTML5, CSS3, Vanilla JS**	Zero-dependency client. Uses the native `Fetch API` for backend communication and DOM manipulation for rendering.
Backend	**Go (Golang)**	Native concurrency for handling WebSocket connections (Telehealth) and strictly typed BSON handling.
Database	**MongoDB**	**Document-Oriented**. Perfect for hierarchical data like "Lineage Trees" and nested "Vaccination Records" which can be embedded directly into the Animal Profile.
Real-Time	**Native WebSockets**	Implemented in pure JS on the client and `gorilla/websocket` on the backend.

3. Data Modeling (NoSQL Document Schema)

Unlike the previous SQL model, we now leverage **Denormalization** and **Embedding** to reduce query latency. Data is stored in BSON (Binary JSON) format.

Collection: `animals` (The Digital Passport)

- **Design Pattern: Polymorphic Pattern**. We store different animal types in the same collection using a `metadata` sub-document.
- **Complex Feature:** Medical records are **embedded** directly into the document for fast retrieval during vet calls.

JSON

```
{
  "_id": ObjectId("..."),
  "rfid_tag": "KAZ-8493-2024",
  "owner_id": ObjectId("..."),
  "type": "HORSE",
  "breed": "Akhal-Teke",
  "lineage": {
    "sire_id": ObjectId("..."),
    "dam_id": ObjectId("...")
  },
  "medical_history": [
    {
      "date": "2023-10-12",
      "vaccine": "Anthrax_V2",
      "vet_signature": "Dr. Aibolit"
    }
  ],
  "attributes": {  // Flexible schema
    "coat_color": "Golden Metallic",
    "wither_height_cm": 160
  }
}
```

Collection: `orders` (E-Commerce Transactions)

- **Design Pattern: Snapshot Pattern**. Product details (price, name) are duplicated into the order document at the time of purchase. This ensures that if the product price changes later, the historical order record remains accurate.

JSON

```
{
  "_id": ObjectId("..."),
  "user_id": ObjectId("..."),
  "status": "CONFIRMED",
  "total_price": 45000,
  "items": [
    {
      "product_id": ObjectId("..."),
      "name": "Premium Oats (50kg)",
      "price_at_purchase": 15000,
      "quantity": 3
    }
  ],
  "created_at": ISODate("2024-02-14T10:00:00Z")
}
```

4. Service Logic & API Design

Module A: The "Vanilla" Frontend Logic

Instead of React components, we use modular JavaScript functions to dynamically render content.

- **Router:** A custom hash-based router (e.g., `/#/market`, `/#/passport`) handles navigation without page reloads.
- **State Management:** A simple reactive store pattern using `Proxy` objects to update the UI when data changes.

Module B: Go Backend (MongoDB Driver)

The backend uses the official `mongo-go-driver`.

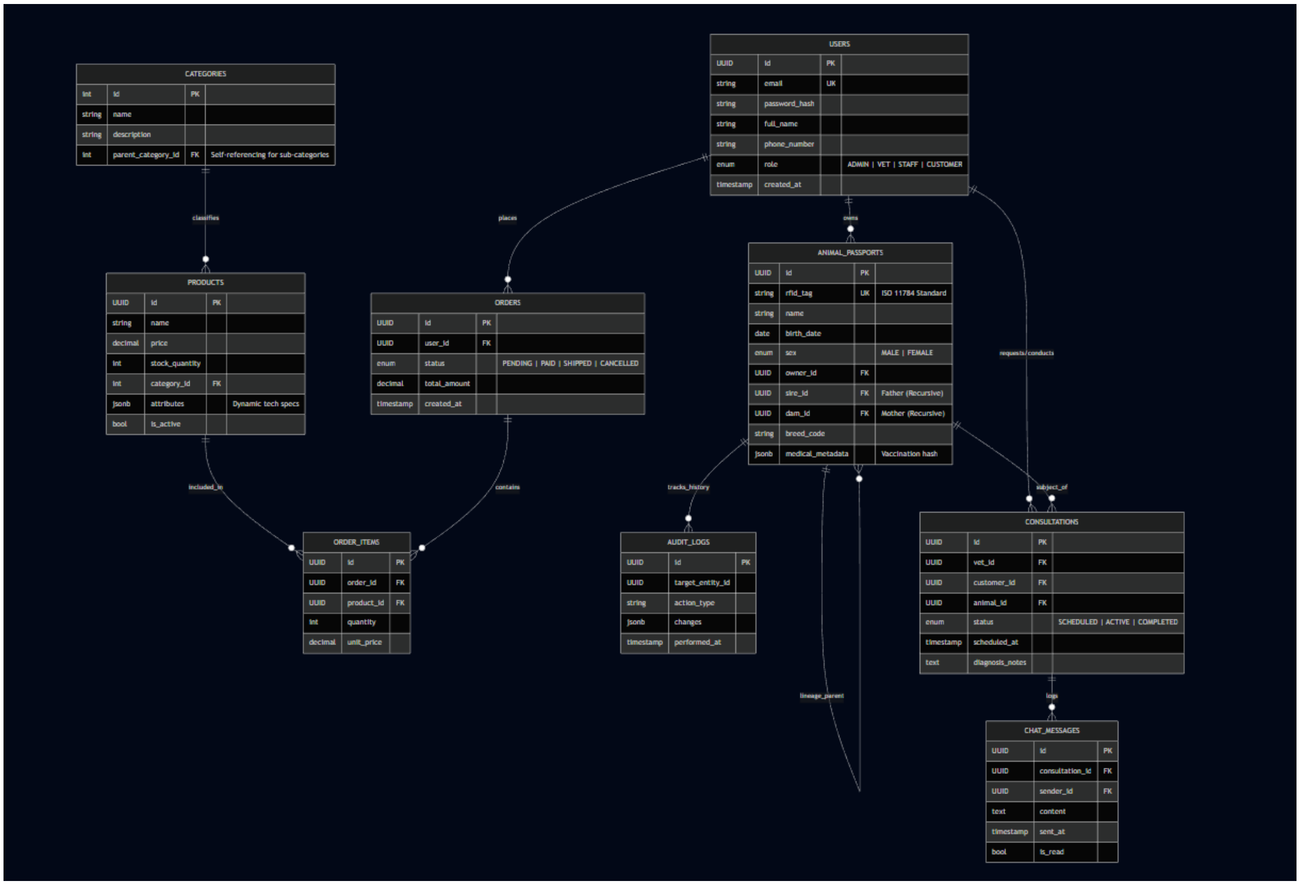
- **Aggregation Pipelines:** We use MongoDB's powerful aggregation framework for complex filtering (e.g., "Find all horses with 'Golden' coat within price range X").
- **Geospatial Indexing:** Using MongoDB's `$near` operator to find the closest veterinarian to the user's GPS coordinates.

5. Security & Scalability

- **NoSQL Injection Protection:** The Go backend validates all inputs against strict Struct types before creating BSON filters, preventing query injection attacks.
- **Sharding:** As the dataset grows (millions of animals), MongoDB allows us to shard the `animals` collection across multiple servers based on the `region_id`

Database Schema (ERD Snippet)

We use a normalized relational schema to ensure ACID compliance.



CATEGORIES

int	id	PK	
string	name		
string	description		
int	parent_category_id	FK	Self-referencing for sub-categories

USERS

UUID	id	PK	
string	email	UK	
string	password_hash		
string	full_name		
string	phone_number		
enum	role		ADMIN | VET | STAFF | CUSTOMER
timestamp	created_at		

PRODUCTS

UUID	id	PK	
string	name		
decimal	price		
int	stock_quantity		
int	category_id	FK	
jsonb	attributes		Dynamic tech specs
bool	is_active		

ORDERS

UUID	id	PK	
UUID	user_id	FK	
enum	status		PENDING | PAID | SHIPPED | CANCELLED
decimal	total_amount		
timestamp	created_at		

ANIMAL_PASSPORTS

UUID	id	PK	
string	rfid_tag	UK	ISO 11784 Standard
string	name		
date	birth_date		
enum	sex		MALE | FEMALE
UUID	owner_id	FK	
UUID	sire_id	FK	Father (Recursive)
UUID	dam_id	FK	Mother (Recursive)
string	breed_code		
jsonb	medical_metadata		Vaccination hash

ORDER_ITEMS

UUID	id	PK
UUID	order_id	FK
UUID	product_id	FK
int	quantity	
decimal	unit_price	

AUDIT_LOGS

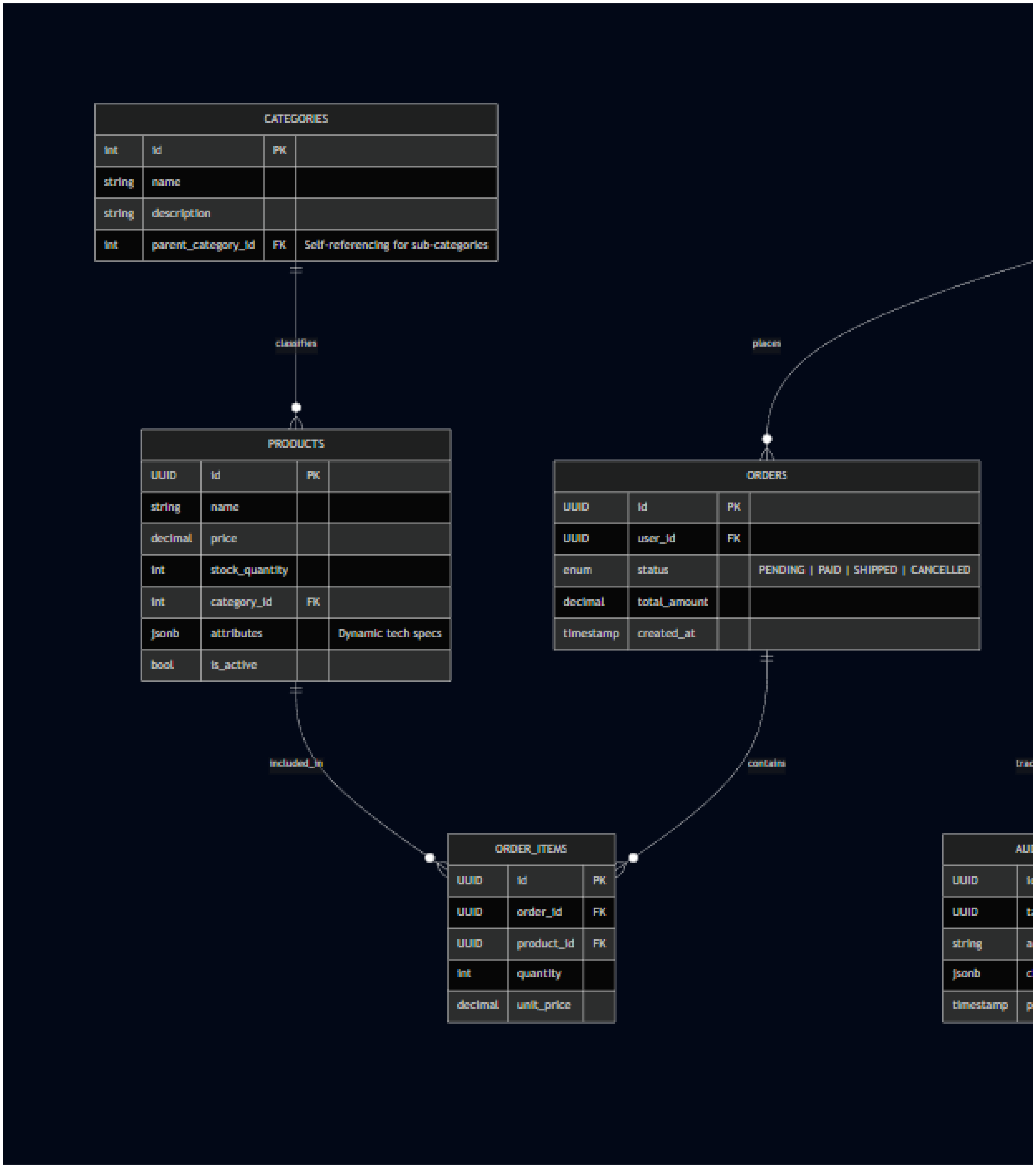
UUID	id	PK
UUID	target_entity_id	
string	action_type	
jsonb	changes	
timestamp	performed_at	

CONSULTATIONS

UUID	id	PK	
UUID	vet_id	FK	
UUID	customer_id	FK	
UUID	animal_id	FK	
enum	status		SCHEDULED | ACTIVE | COMPLETED
timestamp	scheduled_at		
text	diagnosis_notes		

CHAT_MESSAGES

UUID	id	PK
UUID	consultation_id	FK
UUID	sender_id	FK
text	content	
timestamp	sent_at	
bool	is_read	



CATEGORIES

int	id	PK	
string	name		
string	description		
int	parent_category_id	FK	Self-referencing for sub-categories

PRODUCTS

UUID	id	PK	
string	name		
decimal	price		
int	stock_quantity		
int	category_id	FK	
jsonb	attributes		Dynamic tech specs
bool	is_active		

ORDERS

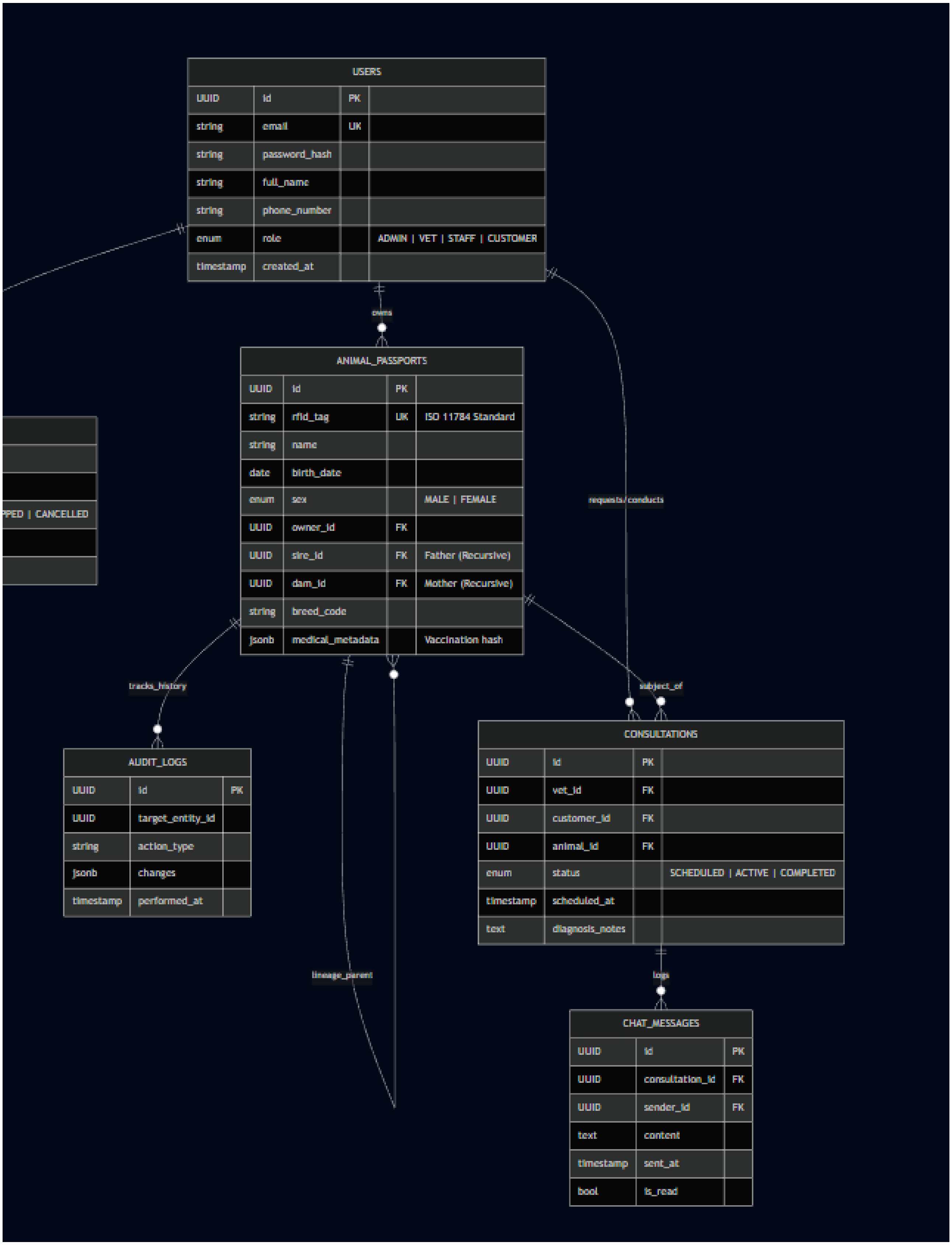
UUID	id	PK	
UUID	user_id	FK	
enum	status		PENDING | PAID | SHIPPED | CANCELLED
decimal	total_amount		
timestamp	created_at		

ORDER_ITEMS

UUID	id	PK
UUID	order_id	FK
UUID	product_id	FK
int	quantity	
decimal	unit_price	

AU

UUID	i
UUID	t
string	a
jsonb	c
timestamp	p



USERS

UUID	id	PK	
string	email	UK	
string	password_hash		
string	full_name		
string	phone_number		
enum	role		ADMIN | VET | STAFF | CUSTOMER
timestamp	created_at		

ANIMAL_PASSPORTS

UUID	id	PK	
string	rfid_tag	UK	ISO 11784 Standard
string	name		
date	birth_date		
enum	sex		MALE | FEMALE
UUID	owner_id	FK	
UUID	sire_id	FK	Father (Recursive)
UUID	dam_id	FK	Mother (Recursive)
string	breed_code		
jsonb	medical_metadata		Vaccination hash

AUDIT_LOGS

UUID	id	PK
UUID	target_entity_id	
string	action_type	
jsonb	changes	
timestamp	performed_at	

CONSULTATIONS

UUID	id	PK	
UUID	vet_id	FK	
UUID	customer_id	FK	
UUID	animal_id	FK	
enum	status		SCHEDULED | ACTIVE | COMPLETED
timestamp	scheduled_at		
text	diagnosis_notes		

CHAT_MESSAGES

UUID	id	PK
UUID	consultation_id	FK
UUID	sender_id	FK
text	content	
timestamp	sent_at	
bool	is_read	