GOSEWA: Gaushala Management System Complete Module Specification Document

System Architecture Team

January 14, 2026

Contents

1 Introduction

This document provides complete specifications for all 28 modules of the GOSEWA platform. Each module is designed to be developed independently using generative AI while ensuring seamless integration through standardized interfaces.

2 Development Guidelines

- **Version Control**: All modules use semantic versioning (v1.0.0)
- **API Standards**: RESTful JSON APIs with consistent error handling
- **Database**: PostgreSQL with standardized table naming convention
- **Auth**: JWT tokens for all authenticated requests
- **Error Codes**: Standardized across all modules

3 Module 1: User Registration & Authentication

Module Overview

ID: M01
Priority: 1
Dependencies: None
Database Required: Yes

3.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: users, user_profiles, verification_tokens
- **API Base Path**: /api/v1/auth
- **Authentication Method**: JWT + Refresh Tokens

3.2 Database Schema

```
-- users table
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(15) UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    user_type VARCHAR(20) CHECK (user_type IN ('GAUSHALA', 'ENTREPRENEUR',
    is_verified BOOLEAN DEFAULT FALSE,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- verification_tokens table
CREATE TABLE verification_tokens (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    token VARCHAR(255) NOT NULL,
    token_type VARCHAR(50) CHECK (token_type IN ('EMAIL_VERIFICATION', 'PAS
    expires_at TIMESTAMP NOT NULL,
    used_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3.3 API Endpoints

Method	Endpoint	Description
POST	/register	Register new user
POST	/login	User login
POST	/verify-email	Verify email address
POST	/forgot-password	Request password reset
POST	/reset-password	Reset password
POST	/refresh-token	Refresh access token
GET	/profile	Get user profile
PUT	/profile	Update profile

3.4 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a complete user authentication module for a gaushala management platform with the following specifications:
1. REQUIREMENTS: - Support 4 user types: GAUSHALA, ENTREPRENEUR, TRANSPORTER, ADMIN - Email and phone number verification - JWT-based authentication with refresh tokens - Password strength validation (min 8 chars, mixed case, numbers) - Rate limiting on login attempts - Secure password reset flow
2. DATABASE SCHEMA: Use PostgreSQL with the exact tables provided in the specification
3. API ENDPOINTS: Implement all endpoints listed in the specification with proper error handling
4. VALIDATION: - Email format validation - Phone number validation for Indian numbers - Unique email and phone enforcement - Input sanitization against SQL injection
5. SECURITY: - Password hashing using bcrypt (cost factor 10) - JWT secret rotation mechanism - CORS configuration - HTTPS enforcement in production - SQL injection prevention
6. ERROR HANDLING: Use consistent error format:

```
{
    "success": false,
    "error": {
        "code": "AUTH_001",
        "message": "Invalid credentials",
        "details": {}
    }
}
```

7. RESPONSE FORMAT:

```
{
    "success": true,
    "data": {},
    "message": "Operation successful"
}
```

8. INTEGRATION POINTS: - Exports: authMiddleware, userTypeMiddleware - Imports: None for this module - Shared: Error constants, response utilities
9. TESTING REQUIREMENTS: - Unit tests for all endpoints - Integration tests for complete flows - Load testing for login endpoint - Security testing for injection attacks
10. DEPLOYMENT: - Environment variables for configuration - Docker containerization - Health check endpoint at /health - Logging with correlation IDs
Integration Note: This module must export authentication middleware that will be used by ALL other modules. The user context must include: userId, userType, email, and isVerified flags.

3.5 Shared Contracts

- **Auth Middleware**: Will be imported by all other modules
- **User Context**: Standard user object format
- **Error Codes**: AUTH_001 to AUTH_999 series

4 Module 2: Profile Management

Module Overview

ID: M02
Priority: 1
Dependencies: M01 (User Authentication)
Database Required: Yes

4.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: user_profiles, business_documents, addresses
- **API Base Path**: /api/v1/profile
- **Authentication Required**: Yes (uses M01 middleware)

4.2 Database Schema

```
-- user_profiles table
CREATE TABLE user_profiles (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID UNIQUE REFERENCES users(id) ON DELETE CASCADE,
    full_name VARCHAR(255) NOT NULL,
    business_name VARCHAR(255),
    business_type VARCHAR(100),
    gst_number VARCHAR(15),
    pan_number VARCHAR(10),
    profile_image_url VARCHAR(500),
    description TEXT,
    verification_status VARCHAR(20) DEFAULT 'PENDING',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- addresses table
CREATE TABLE addresses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    address_type VARCHAR(20) CHECK (address_type IN ('BUSINESS', 'WAREHOUSE
    street_address TEXT NOT NULL,
    city VARCHAR(100) NOT NULL,
    state VARCHAR(100) NOT NULL,
    country VARCHAR(100) DEFAULT 'India',
    postal_code VARCHAR(10) NOT NULL,
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
```

```
    is_default BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- business_documents table
CREATE TABLE business_documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    document_type VARCHAR(50) CHECK (document_type IN ('GST_CERTIFICATE',
    document_url VARCHAR(500) NOT NULL,
    verified_by UUID REFERENCES users(id),
    verification_status VARCHAR(20) DEFAULT 'PENDING',
    verified_at TIMESTAMP,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

4.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a comprehensive profile management module that integrates with Module 1's authentication system.
1. REQUIREMENTS: - CRUD operations for user profiles - Multiple address management (business, warehouse, billing) - Document upload and verification workflow - Location coordinates (latitude/longitude) storage - Profile completion percentage calculation
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /profile - Get complete profile with addresses - PUT /profile - Update profile information - POST /addresses - Add new address - PUT /addresses/:id - Update address - DELETE /addresses/:id - Delete address - POST /documents - Upload business document - GET /documents - List all documents - GET /profile/completion - Get profile completion percentage
4. INTEGRATION WITH M01: - Import authMiddleware from Module 1 - Use JWT tokens for authentication - Access user context from req.user (set by M01 middleware)
5. FILE UPLOAD: - Support PDF, JPG, PNG up to 10MB - Store in cloud storage (S3 compatible) - Generate secure URLs with expiration
6. VALIDATION: - GST number format validation - PAN card format validation - Address validation using pincode API - Coordinate validation
7. BUSINESS LOGIC: - Only one default address per type - Document verification workflow (PENDING → REVIEW → VERIFIED/REJECTED) - Profile completion rules: * Basic info: 20* Business details: 30* Address: 20* Documents: 30
8. ERROR HANDLING: Use PROFILE_001 error code series
9. INTEGRATION POINTS: - Imports: authMiddleware from M01 - Exports: getUserProfile function - Shared: File upload utilities
10. TESTING: - Test with different user types - Document upload and retrieval - Address management scenarios - Profile completion calculation
Integration Note: This module must be able to retrieve user profiles for order processing (Module 5), logistics (Module 10), and payment (Module 6). Store user IDs consistently with Module 1.

5 Module 3: Gaushala Inventory Catalog

Module Overview

ID: M03
Priority: 2
Dependencies: M01, M02
Database Required: Yes

5.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: products, categories, inventory_logs, product_images
- **API Base Path**: /api/v1/inventory
- **Authentication Required**: Yes (GAUSHALA users only for write)

5.2 Database Schema

```
-- categories table
CREATE TABLE categories (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(100) NOT NULL,
    slug VARCHAR(100) UNIQUE NOT NULL,
    parent_id UUID REFERENCES categories(id),
    description TEXT,
    icon_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- products table
CREATE TABLE products (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    gaushala_id UUID REFERENCES users(id) ON DELETE CASCADE,
    category_id UUID REFERENCES categories(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    sku VARCHAR(100) UNIQUE,
    unit_type VARCHAR(20) CHECK (unit_type IN ('KG', 'LITER', 'PIECE', 'BAG
    price_per_unit DECIMAL(10, 2) NOT NULL,
    available_quantity DECIMAL(10, 2) DEFAULT 0,
    min_order_quantity DECIMAL(10, 2) DEFAULT 1,
    max_order_quantity DECIMAL(10, 2),
    quality_grade VARCHAR(20) CHECK (quality_grade IN ('A', 'B', 'C', 'PREM
    is_organic BOOLEAN DEFAULT FALSE,
    is_available BOOLEAN DEFAULT TRUE,
    harvest_date DATE,
```

```
    expiry_date DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- product_images table
CREATE TABLE product_images (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    image_url VARCHAR(500) NOT NULL,
    is_primary BOOLEAN DEFAULT FALSE,
    display_order INTEGER DEFAULT 0,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- inventory_logs table
CREATE TABLE inventory_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    quantity_change DECIMAL(10, 2) NOT NULL,
    new_quantity DECIMAL(10, 2) NOT NULL,
    change_type VARCHAR(20) CHECK (change_type IN ('ADDITION', 'SALE', 'ADJ
    reference_id UUID, -- Can reference order_id or other entities
    notes TEXT,
    created_by UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

5.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create an inventory management module for gaushala products with category management and stock tracking.
1. REQUIREMENTS: - CRUD operations for products (only for GAUSHALA users) - Category hierarchy management - Inventory stock tracking with logs - Product image management - Available quantity validation
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /categories - List all categories with hierarchy - POST /categories - Create category (ADMIN only) - GET /products - List products with filters - POST /products - Create product (GAUSHALA only) - PUT /products/:id - Update product - DELETE /products/:id - Delete product - POST /products/:id/inventory - Update inventory quantity - GET /products/:id/logs - Get inventory history - POST /products/:id/images - Upload product images
4. INTEGRATION: - Import authMiddleware and userTypeMiddleware from M01 - Use gaushala_id from user context - Validate user is GAUSHALA type for write operations
5. BUSINESS LOGIC: - Auto-generate SKU: GS-{category}-{timestamp} - Prevent negative inventory - Track all inventory changes with logs - Calculate available quantity from logs - Enforce min/max order quantities
6. SEARCH/FILTERS: - By category and subcategory - By price range - By location (proximity to entrepreneur) - By quality grade - By availability - By organic certification
7. VALIDATION: - Price must be positive - Quantity must be non-negative - Date validations (harvest ¡ expiry) - Image file type and size limits
8. INTEGRATION POINTS: - Imports: authMiddleware from M01, location from M02 - Exports: getProductById, updateInventory functions - Shared: Product availability check for M05
9. ERROR CODES: Use INVENTORY_001 series
10. WEBHOOKS: - Inventory low alert (less than 10%) - Product availability change notifications
Integration Note: This module provides product data to M04 (Marketplace) and validates inventory for M05 (Orders). The product IDs must be consistent across all modules.

6 Module 4: Marketplace Browsing

Module Overview

ID: M04
Priority: 2
Dependencies: M01, M02, M03
Database Required: Yes (read-only from M03 tables)

6.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: search_history, saved_searches, product_views
- **API Base Path**: /api/v1/marketplace
- **Authentication Required**: Optional (for personalized features)

6.2 Database Schema

```
-- search_history table
CREATE TABLE search_history (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    search_query TEXT NOT NULL,
    filters JSONB,
    result_count INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- saved_searches table
CREATE TABLE saved_searches (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    search_query TEXT,
    filters JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    notification_frequency VARCHAR(20),
    last_notified_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- product_views table
CREATE TABLE product_views (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
```

```
    view_duration INTEGER, -- in seconds
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a marketplace browsing module for entrepreneurs to discover gaushala products with advanced search and filtering.
1. REQUIREMENTS: - Advanced product search with multiple filters - Location-based proximity search - Save search criteria with alerts - Track user browsing behavior - Pagination and sorting options
2. DATABASE: Use exact schema for tracking tables. Read from M03 tables.
3. API ENDPOINTS: - GET /search - Search products with filters - GET /categories/:id/products - Get products by category - GET /products/:id/details - Get product details with gaushala info - POST /search/save - Save search criteria - GET /search/history - Get user search history - GET /products/featured - Get featured products - GET /products/nearby - Get products near user location
4. INTEGRATION: - Read from M03 products and categories tables - Import user location from M02 addresses - Calculate distance using Haversine formula
5. SEARCH PARAMETERS: - Keyword search in product name/description - Category and subcategory filters - Price range (min/max) - Distance radius (5km, 10km, 25km, 50km, 100km) - Quality grade filter - Organic certification filter - Availability status - Gaushala rating (from M07)
6. SORTING OPTIONS: - Price: low to high, high to low - Distance: nearest first - Rating: highest first - Recent: newest products - Popular: most viewed
7. BUSINESS LOGIC: - Calculate distance between user and gaushala - Filter out unavailable products - Show only verified gaushalas - Highlight organic products - Show stock availability status
8. PERFORMANCE: - Implement search indexing - Cache frequent searches - Pagination with cursor-based pagination - Rate limiting on search endpoints
9. INTEGRATION POINTS: - Imports: Product data from M03, location from M02 - Exports: searchProducts function - Shared: Distance calculation utilities
10. PERSONALIZATION: - Show recently viewed products - Recommend based on search history - Show popular products in user's area
Integration Note: This module provides the interface for M05 (Order Management). Product selection from here flows directly to the cart. Ensure product IDs match exactly with M03.

7 Module 5: Basic Order Management

Module Overview

ID: M05
Priority: 3
Dependencies: M01, M02, M03, M04
Database Required: Yes

7.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: carts, cart_items, orders, order_items, order_status_history
- **API Base Path**: /api/v1/orders
- **Authentication Required**: Yes

7.2 Database Schema

```
-- carts table
CREATE TABLE carts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID UNIQUE REFERENCES users(id) ON DELETE CASCADE,
    session_id VARCHAR(255),
    total_amount DECIMAL(10, 2) DEFAULT 0,
    item_count INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- cart_items table
CREATE TABLE cart_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    cart_id UUID REFERENCES carts(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    quantity DECIMAL(10, 2) NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL,
    total_price DECIMAL(10, 2) GENERATED ALWAYS AS (quantity * unit_price)
    added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- orders table
CREATE TABLE orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_number VARCHAR(50) UNIQUE NOT NULL,
    entrepreneur_id UUID REFERENCES users(id) ON DELETE CASCADE,
    gaushala_id UUID REFERENCES users(id),
```

```
    total_amount DECIMAL(10, 2) NOT NULL,
    subtotal DECIMAL(10, 2) NOT NULL,
    tax_amount DECIMAL(10, 2) DEFAULT 0,
    shipping_amount DECIMAL(10, 2) DEFAULT 0,
    discount_amount DECIMAL(10, 2) DEFAULT 0,
    final_amount DECIMAL(10, 2) NOT NULL,
    shipping_address_id UUID REFERENCES addresses(id),
    billing_address_id UUID REFERENCES addresses(id),
    order_status VARCHAR(20) DEFAULT 'PENDING',
    payment_status VARCHAR(20) DEFAULT 'PENDING',
    payment_method VARCHAR(50),
    notes TEXT,
    expected_delivery_date DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- order_items table
CREATE TABLE order_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_id UUID REFERENCES orders(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id),
    product_name VARCHAR(255) NOT NULL,
    quantity DECIMAL(10, 2) NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL,
    total_price DECIMAL(10, 2) NOT NULL,
    unit_type VARCHAR(20) NOT NULL,
    quality_grade VARCHAR(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- order_status_history table
CREATE TABLE order_status_history (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_id UUID REFERENCES orders(id) ON DELETE CASCADE,
    old_status VARCHAR(20),
    new_status VARCHAR(20) NOT NULL,
    changed_by UUID REFERENCES users(id),
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

7.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a complete order management system with cart functionality, order processing, and status tracking.
1. REQUIREMENTS: - Shopping cart management (add/remove/update) - Order creation from cart - Order status workflow management - Order history and tracking - Multi-vendor order support (one order per gaushala)
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /cart - Get cart contents - POST /cart/items - Add item to cart - PUT /cart/items/:id - Update cart item - DELETE /cart/items/:id - Remove from cart - POST /cart/checkout - Convert cart to order - GET /orders - List user orders - GET /orders/:id - Get order details - PUT /orders/:id/status - Update order status - POST /orders/:id/cancel - Cancel order - GET /orders/:id/timeline - Get status history
4. INTEGRATION: - Import product data and inventory from M03 - Import user addresses from M02 - Use authMiddleware from M01 - Check product availability before adding to cart
5. ORDER WORKFLOW: 1. PENDING → CONFIRMED (when payment successful) 2. CONFIRMED → PROCESSING (gaushala accepts) 3. PROCESSING → READY$_F$$O$$R_S$$H$$I$$P$$M$$E$$N$$T$4.$R$$E$$A$$D$$Y_F$$O$$R_S$$H$$I$$P$$M$$E$$N$$T$ß$I$$N_T$$R$$A$$N$$S$$I$$T$5.$I$$N_T$$R$$A$$N$$S$$I$$T$ß$D$$E$$L$$I$$V$$E$
6. BUSINESS LOGIC: - Generate order number: GO-{YYMMDD}-{6 digits} - Validate quantity against available stock - Calculate taxes based on product category - Apply bulk discounts (if any) - Split cart by gaushala (one order per gaushala) - Lock inventory when order is confirmed
7. VALIDATION: - Minimum order quantity per product - Maximum order quantity per product - Shipping address validation - Product availability check - Business hour restrictions
8. INTEGRATION POINTS: - Imports: M03 (inventory), M02 (addresses), M01 (auth) - Exports: createOrder, updateOrderStatus functions - Shared: Order status constants - Webhooks: Order status change notifications
9. ERROR CODES: Use ORDER_001 series
10. NOTIFICATIONS: - Email confirmation on order creation - SMS alert to gaushala on new order - Status update notifications - Delivery reminders
Integration Note: This module connects to M06 (Payment), M10 (Logistics), and M07 (Reviews). Order status changes must trigger logistics workflow.

8 Module 6: Payment Processing

Module Overview

ID: M06
Priority: 3
Dependencies: M01, M05
Database Required: Yes

8.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: payments, payment_methods, invoices, refunds
- **API Base Path**: /api/v1/payments
- **Authentication Required**: Yes
- **External APIs**: Razorpay/Stripe, UPI

8.2 Database Schema

```
-- payments table
CREATE TABLE payments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_id UUID REFERENCES orders(id) ON DELETE CASCADE,
    payment_id VARCHAR(255) UNIQUE, -- Gateway payment ID
    payment_method VARCHAR(50) NOT NULL,
    amount DECIMAL(10, 2) NOT NULL,
    currency VARCHAR(3) DEFAULT 'INR',
    status VARCHAR(20) DEFAULT 'PENDING',
    gateway_response JSONB,
    paid_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- payment_methods table
CREATE TABLE payment_methods (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    method_type VARCHAR(20) CHECK (method_type IN ('UPI', 'CARD', 'NETBANK
    method_details JSONB NOT NULL,
    is_default BOOLEAN DEFAULT FALSE,
    is_verified BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```sql
-- invoices table
CREATE TABLE invoices (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    invoice_number VARCHAR(50) UNIQUE NOT NULL,
    order_id UUID REFERENCES orders(id) ON DELETE CASCADE,
    entrepreneur_id UUID REFERENCES users(id),
    gaushala_id UUID REFERENCES users(id),
    invoice_date DATE NOT NULL,
    due_date DATE,
    amount DECIMAL(10, 2) NOT NULL,
    tax_amount DECIMAL(10, 2),
    paid_amount DECIMAL(10, 2) DEFAULT 0,
    status VARCHAR(20) DEFAULT 'PENDING',
    download_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- refunds table
CREATE TABLE refunds (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    payment_id UUID REFERENCES payments(id),
    refund_id VARCHAR(255) UNIQUE,
    amount DECIMAL(10, 2) NOT NULL,
    reason TEXT,
    status VARCHAR(20) DEFAULT 'PENDING',
    processed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

8.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a secure payment processing module integrated with Razorpay and UPI for handling transactions.
1. REQUIREMENTS: - Multiple payment method support (UPI, Cards, Net Banking, Wallets) - Payment gateway integration (Razorpay) - Invoice generation and management - Refund processing - Payment status synchronization
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /payments/create - Create payment for order - POST /payments/verify - Verify payment webhook - GET /payments/:id - Get payment details - POST /payments/:id/refund - Initiate refund - GET /invoices/:order$_id - Get invoice for order - POST /payment - methods - Add payment method - GET /payment - methods - List user payment methods - DELETE /payment - methods/ : id - Remove payment method
4. INTEGRATION: - Import order data from M05 - Update order payment status in M05 - Use authMiddleware from M01
5. PAYMENT METHODS: - UPI (collect and intent) - Credit/Debit Cards - Net Banking (major Indian banks) - Wallets (Paytm, PhonePe, etc.) - Cash on Delivery (limited availability)
6. RAZORPAY INTEGRATION: - Create order in Razorpay - Handle webhook for payment capture - Store payment gateway response - Idempotency key for duplicate requests
7. INVOICE GENERATION: - Generate invoice number: INV-{YYMMDD}-{6 digits} - Include GST details if available - Include itemized breakdown - PDF generation and storage - Email invoice to customer
8. BUSINESS LOGIC: - Validate payment amount matches order amount - Handle partial payments for bulk orders - Auto-cancel unpaid orders after 24 hours - Escrow release on delivery confirmation - Commission calculation (platform fee)
9. SECURITY: - Never store card details - PCI DSS compliance for card payments - Webhook signature verification - Rate limiting on payment endpoints - Idempotent payment processing
10. ERROR HANDLING: - Payment gateway errors - Network timeouts - Duplicate payment prevention - Webhook replay attacks prevention
11. INTEGRATION POINTS: - Imports: M05 (order data), M01 (auth) - Exports: processPayment, verifyPayment functions - Webhooks: Payment success/failure notifications - Shared: Payment status constants
12. TESTING: - Test with Razorpay sandbox - Webhook simulation - Refund flow testing - Error scenario testing
Integration Note: Payment success must trigger order confirmation in M05. Payment failure must trigger order cancellation. Store order_id foreign key exactly as in M05.

9 Module 7: Review & Rating System

Module Overview

ID: M07
Priority: 4
Dependencies: M01, M05
Database Required: Yes

9.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: reviews, ratings, review_reports, helpful_votes
- **API Base Path**: /api/v1/reviews
- **Authentication Required**: Yes (for posting reviews)

9.2 Database Schema

```
-- reviews table
CREATE TABLE reviews (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_id UUID REFERENCES orders(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    gaushala_id UUID REFERENCES users(id) ON DELETE CASCADE,
    entrepreneur_id UUID REFERENCES users(id) ON DELETE CASCADE,
    rating DECIMAL(2,1) CHECK (rating >= 1 AND rating <= 5),
    title VARCHAR(200),
    review_text TEXT,
    is_verified_purchase BOOLEAN DEFAULT TRUE,
    status VARCHAR(20) DEFAULT 'PENDING',
    moderator_notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- ratings table (aggregated)
CREATE TABLE ratings (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    entity_id UUID NOT NULL,
    entity_type VARCHAR(20) CHECK (entity_type IN ('GAUSHALA', 'PRODUCT',
    total_ratings INTEGER DEFAULT 0,
    average_rating DECIMAL(2,1) DEFAULT 0,
    rating_1_count INTEGER DEFAULT 0,
    rating_2_count INTEGER DEFAULT 0,
    rating_3_count INTEGER DEFAULT 0,
    rating_4_count INTEGER DEFAULT 0,
```

```
    rating_5_count INTEGER DEFAULT 0,
    last_calculated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- review_reports table
CREATE TABLE review_reports (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    review_id UUID REFERENCES reviews(id) ON DELETE CASCADE,
    reported_by UUID REFERENCES users(id),
    report_reason VARCHAR(50),
    report_text TEXT,
    status VARCHAR(20) DEFAULT 'PENDING',
    resolved_by UUID REFERENCES users(id),
    resolved_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- helpful_votes table
CREATE TABLE helpful_votes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    review_id UUID REFERENCES reviews(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    is_helpful BOOLEAN NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(review_id, user_id)
);
```

9.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a comprehensive review and rating system with moderation and analytics.
1. REQUIREMENTS: - Product and gaushala ratings (1-5 stars) - Verified purchase reviews only - Review moderation system - Helpful vote tracking - Review reporting mechanism
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /reviews - Submit review for order - GET /reviews - List reviews with filters - GET /reviews/:id - Get specific review - PUT /reviews/:id - Update review - DELETE /reviews/:id - Delete review - POST /reviews/:id/report - Report inappropriate review - POST /reviews/:id/helpful - Mark review as helpful - GET /ratings/:entity$_{i}ype/$: $entity_{i}d - Getaggregatedratings - GET/reviews/stats - Getreviewstatistics$
4. INTEGRATION: - Import order data from M05 (for verified purchase check) - Use authMiddleware from M01 - Update aggregated ratings in real-time
5. BUSINESS RULES: - Only allow reviews for delivered orders - One review per order per entity (product/gaushala) - Allow review editing within 7 days - Auto-approve reviews after 24 hours unless reported - Calculate average rating with weighted algorithm
6. MODERATION WORKFLOW: 1. PENDING → Auto-approved after 24 hours 2. PENDING → APPROVED (manual) 3. PENDING → REJECTED (if violates guidelines) 4. APPROVED → FLAGGED (if reported) 5. FLAGGED → APPROVED or REMOVED
7. RATING CALCULATION: - Weight recent reviews higher (last 30 days) - Consider helpful votes in ranking - Calculate Bayesian average for small sample sizes - Update aggregated counts incrementally
8. VALIDATION: - Order must be in DELIVERED status - User must be the entrepreneur who placed order - Minimum 10 characters for review text - Rating must be between 1 and 5
9. SEARCH/FILTERS: - Filter by rating (1-5 stars) - Sort by date (newest first) - Sort by helpful votes - Show verified purchases only filter - Filter by product or gaushala
10. INTEGRATION POINTS: - Imports: M05 (order validation), M01 (auth) - Exports: getAverageRating, getReviews functions - Shared: Rating calculation utilities - Webhooks: New review notifications
11. ANALYTICS: - Rating distribution charts - Review response rate - Average response time - Most helpful reviews
Integration Note: Average ratings from this module are displayed in M04 (Marketplace). Review counts affect gaushala ranking. Ensure entity IDs match user and product IDs from M01 and M03.

10 Module 8: Notification System

Module Overview

ID: M08
Priority: 4
Dependencies: M01
Database Required: Yes

10.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: notifications, notification_templates, notification_preferences, notification_channels
- **API Base Path**: /api/v1/notifications
- **External Services**: Email (SendGrid/SES), SMS (Twilio), Push (Firebase)

10.2 Database Schema

```
-- notifications table
CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    notification_type VARCHAR(50) NOT NULL,
    title VARCHAR(255) NOT NULL,
    message TEXT NOT NULL,
    data JSONB,
    priority VARCHAR(10) CHECK (priority IN ('LOW', 'MEDIUM', 'HIGH', 'URGE
    status VARCHAR(20) DEFAULT 'PENDING',
    read_at TIMESTAMP,
    sent_at TIMESTAMP,
    failed_at TIMESTAMP,
    failure_reason TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- notification_templates table
CREATE TABLE notification_templates (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    template_code VARCHAR(100) UNIQUE NOT NULL,
    template_type VARCHAR(20) CHECK (template_type IN ('EMAIL', 'SMS', 'PUS
    subject VARCHAR(255),
    body_template TEXT NOT NULL,
    variables JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- notification_preferences table
CREATE TABLE notification_preferences (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    notification_type VARCHAR(50) NOT NULL,
    email_enabled BOOLEAN DEFAULT TRUE,
    sms_enabled BOOLEAN DEFAULT TRUE,
    push_enabled BOOLEAN DEFAULT TRUE,
    in_app_enabled BOOLEAN DEFAULT TRUE,
    frequency VARCHAR(20) DEFAULT 'IMMEDIATE',
    quiet_hours_start TIME,
    quiet_hours_end TIME,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, notification_type)
);

-- notification_channels table (for async processing)
CREATE TABLE notification_channels (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    notification_id UUID REFERENCES notifications(id) ON DELETE CASCADE,
    channel_type VARCHAR(20) CHECK (channel_type IN ('EMAIL', 'SMS', 'PUSH'
    channel_id VARCHAR(255),
    status VARCHAR(20) DEFAULT 'PENDING',
    sent_at TIMESTAMP,
    delivered_at TIMESTAMP,
    failed_at TIMESTAMP,
    failure_reason TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

10.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create an asynchronous notification system with multi-channel support (email, SMS, push, in-app).
1. REQUIREMENTS: - Multi-channel notification delivery - Template-based notification system - User preference management - Async processing with retries - Delivery tracking and analytics
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /notifications/send - Send notification - GET /notifications - List user notifications - PUT /notifications/:id/read - Mark as read - DELETE /notifications/:id - Delete notification - GET /notifications/unread-count - Get unread count - GET /preferences - Get user preferences - PUT /preferences - Update preferences - GET /templates - List templates (ADMIN) - POST /templates - Create template (ADMIN) - PUT /templates/:id - Update template (ADMIN)
4. INTEGRATION: - Import authMiddleware from M01 - Provide notification service to all other modules - Use queue system for async processing
5. NOTIFICATION TYPES: - $\text{ORDER}_{C}REATED, ORDER_{C}ONFIRMED, ORDER_{S}HIPP$
$PAYMENT_{S}UCCESS, PAYMENT_{F}AILED-SHIPMENT_{D}ELAYED, SHIPMENT_{D}ELIV$
$REVIEW_{R}EQUEST, REVIEW_{R}ECEIVED-INVENTORY_{L}OW, PRICE_{D}ROP-$
$NEW_{M}ESSAGE, SYSTEM_{A}LERT$
6. TEMPLATE SYSTEM: - Variable substitution: {{user.name}}, {{order.number}} - Conditional content in templates - Localization support (Hindi/English) - A/B testing for template variants
7. CHANNEL INTEGRATION: - Email: SendGrid or Amazon SES - SMS: Twilio or Indian SMS providers - Push: Firebase Cloud Messaging - In-app: WebSocket or polling
8. BUSINESS LOGIC: - Respect user quiet hours - Honor channel preferences - Retry failed deliveries (max 3 attempts) - Exponential backoff for retries - Bulk notification optimization
9. PERFORMANCE: - Queue-based async processing - Batch notifications where possible - Cache templates in memory - Connection pooling for external services
10. MONITORING: - Delivery success rate tracking - Channel performance analytics - User engagement metrics - Notification volume trends
11. INTEGRATION POINTS: - Imports: M01 (user data) - Exports: sendNotification, sendBulkNotification functions - Shared: Notification service instance - Webhooks: Delivery status callbacks
12. ERROR HANDLING: - Graceful degradation (if SMS fails, try email) - Circuit breaker for external services - Dead letter queue for failed notifications - Alert on high failure rates
Integration Note: All other modules will import the notification service from this module. Use dependency injection pattern. Store user_id exactly as in M01.

11 Module 9: Dashboard Analytics

Module Overview

ID: M09
Priority: 5
Dependencies: M01, M03, M05, M06, M07
Database Required: Yes (read-only aggregations)

11.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: analytics_cache, dashboard_widgets, user_dashboard_prefs
- **API Base Path**: /api/v1/analytics
- **Authentication Required**: Yes
- **External Services**: Charting libraries, Cache (Redis)

11.2 Database Schema

```
-- analytics_cache table
CREATE TABLE analytics_cache (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    cache_key VARCHAR(255) UNIQUE NOT NULL,
    cache_type VARCHAR(50) NOT NULL,
    data JSONB NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- dashboard_widgets table
CREATE TABLE dashboard_widgets (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    widget_type VARCHAR(50) NOT NULL,
    widget_name VARCHAR(100) NOT NULL,
    description TEXT,
    config JSONB,
    user_type VARCHAR(20) CHECK (user_type IN ('GAUSHALA', 'ENTREPRENEUR',
    default_position INTEGER,
    is_active BOOLEAN DEFAULT TRUE,
    refresh_interval INTEGER DEFAULT 300, -- seconds
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- user_dashboard_prefs table
```

```
CREATE TABLE user_dashboard_prefs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    widget_id UUID REFERENCES dashboard_widgets(id) ON DELETE CASCADE,
    position INTEGER NOT NULL,
    is_visible BOOLEAN DEFAULT TRUE,
    config JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, widget_id)
);
```

11.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a real-time analytics dashboard module with customizable widgets and performance metrics.
1. REQUIREMENTS: - Role-based dashboards (Gaushala, Entrepreneur, Transporter, Admin) - Customizable widget layouts - Real-time data updates - Performance metric calculations - Export functionality
2. DATABASE: Use exact schema for caching and preferences
3. API ENDPOINTS: - GET /dashboard - Get user dashboard with widgets - PUT /dashboard/layout - Update widget layout - GET /analytics/sales - Sales analytics - GET /analytics/inventory - Inventory analytics - GET /analytics/orders - Order analytics - GET /analytics/customers - Customer analytics - GET /analytics/export - Export analytics data - GET /widgets - Available widgets for user type - POST /widgets/:id/toggle - Toggle widget visibility
4. INTEGRATION: - Import data from M03 (inventory), M05 (orders), M06 (payments), M07 (reviews) - Use authMiddleware from M01 - Cache frequently accessed data
5. GAUSHALA DASHBOARD WIDGETS: - Sales Overview (today, week, month) - Top Selling Products - Inventory Levels - Order Fulfillment Rate - Customer Reviews - Revenue Trends - Geographical Orders Map
6. ENTREPRENEUR DASHBOARD WIDGETS: - Recent Orders - Spending Analysis - Favorite Gaushalas - Price Comparison - Order Status Tracking - Saved Products - Recommended Products
7. ADMIN DASHBOARD WIDGETS: - Platform Overview - User Growth - Transaction Volume - Revenue Metrics - Top Performing Gaushalas - System Health - Geographical Distribution
8. BUSINESS LOGIC: - Calculate metrics with proper aggregations - Handle date range filters (today, week, month, quarter, year) - Compare with previous period - Calculate growth percentages - Identify trends and anomalies
9. PERFORMANCE: - Implement data caching (Redis) - Use materialized views for complex queries - Paginate large datasets - Lazy load widget data - Background data processing
10. DATA VISUALIZATION: - Chart.js or similar for graphs - Export to PDF/Excel - Real-time updates via WebSocket - Responsive design for mobile
11. INTEGRATION POINTS: - Imports: M01 (user context), M03, M05, M06, M07 - Exports: getMetrics, generateReport functions - Shared: Date range utilities, aggregation functions - Webhooks: Data update notifications
12. SECURITY: - Role-based data access - Data sanitization for exports - Rate limiting on data queries - Audit logging for sensitive data access
Integration Note: This module reads data from multiple modules but doesn't write back. Ensure foreign key relationships match exactly with source modules.

12 Module 10: Logistics Booking

Module Overview

ID: M10
Priority: 5
Dependencies: M01, M02, M05, M08
Database Required: Yes

12.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: shipments, shipment_items, pickup_requests, delivery_slots
- **API Base Path**: /api/v1/logistics
- **Authentication Required**: Yes

12.2 Database Schema

```
-- shipments table
CREATE TABLE shipments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_number VARCHAR(50) UNIQUE NOT NULL,
    order_id UUID REFERENCES orders(id) ON DELETE CASCADE,
    gaushala_id UUID REFERENCES users(id),
    entrepreneur_id UUID REFERENCES users(id),
    pickup_address_id UUID REFERENCES addresses(id),
    delivery_address_id UUID REFERENCES addresses(id),
    transporter_id UUID REFERENCES users(id),
    estimated_weight DECIMAL(10, 2), -- in kg
    actual_weight DECIMAL(10, 2),
    package_dimensions JSONB,
    shipment_type VARCHAR(20) CHECK (shipment_type IN ('STANDARD', 'COLD_CH
    status VARCHAR(20) DEFAULT 'PENDING',
    pickup_scheduled_at TIMESTAMP,
    picked_up_at TIMESTAMP,
    estimated_delivery_at TIMESTAMP,
    delivered_at TIMESTAMP,
    shipping_cost DECIMAL(10, 2),
    distance_km DECIMAL(6, 2),
    tracking_url VARCHAR(500),
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- shipment_items table
```

```
CREATE TABLE shipment_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_id UUID REFERENCES shipments(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id),
    order_item_id UUID REFERENCES order_items(id),
    quantity DECIMAL(10, 2) NOT NULL,
    unit_type VARCHAR(20) NOT NULL,
    handling_instructions TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- pickup_requests table
CREATE TABLE pickup_requests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_id UUID REFERENCES shipments(id) ON DELETE CASCADE,
    requested_by UUID REFERENCES users(id),
    pickup_window_start TIMESTAMP NOT NULL,
    pickup_window_end TIMESTAMP NOT NULL,
    status VARCHAR(20) DEFAULT 'PENDING',
    assigned_to UUID REFERENCES users(id),
    completed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- delivery_slots table
CREATE TABLE delivery_slots (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    transporter_id UUID REFERENCES users(id),
    slot_date DATE NOT NULL,
    slot_start TIME NOT NULL,
    slot_end TIME NOT NULL,
    capacity_kg DECIMAL(10, 2) DEFAULT 1000,
    booked_capacity DECIMAL(10, 2) DEFAULT 0,
    is_available BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(transporter_id, slot_date, slot_start)
);
```

12.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a logistics booking system for managing shipments, pickups, and deliveries.
1. REQUIREMENTS: - Shipment creation from orders - Pickup scheduling and management - Delivery slot booking - Transporter assignment - Shipment tracking integration
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /shipments - Create shipment for order - GET /shipments - List shipments - GET /shipments/:id - Get shipment details - PUT /shipments/:id/status - Update shipment status - POST /shipments/:id/pickup-request - Schedule pickup - GET /delivery-slots/available - Get available slots - POST /delivery-slots/book - Book delivery slot - GET /shipments/:id/tracking - Get tracking info - POST /shipments/:id/cancel - Cancel shipment
4. INTEGRATION: - Import order data from M05 - Import addresses from M02 - Use notification service from M08 - Use authMiddleware from M01
5. SHIPMENT WORKFLOW: 1. PENDING → SCHEDULED (pickup scheduled) 2. SCHEDULED → $\text{IN}_T RANSIT(pickedup) 3. IN_T RANSIT ß OUT_F OR_D ELIV ERY 4. OUT_F OR_D ELIV ERY ß DELIV$
6. BUSINESS LOGIC: - Generate shipment number: SH-{YYMMDD}-{6 digits} - Calculate shipping cost based on weight and distance - Assign optimal transporter based on location and capacity - Validate pickup and delivery addresses - Handle special shipment types (cold chain, livestock)
7. DISTANCE CALCULATION: - Use Google Maps API or OSRM - Calculate road distance, not straight line - Cache distance calculations - Update ETA based on traffic conditions
8. SLOT MANAGEMENT: - Transporter defines available slots - Auto-assign slots based on preference - Prevent double-booking - Handle slot rescheduling
9. INTEGRATION POINTS: - Imports: M05 (orders), M02 (addresses), M08 (notifications) - Exports: createShipment, updateShipmentStatus functions - Shared: Distance calculation utilities - Webhooks: Shipment status updates
10. NOTIFICATIONS: - Pickup scheduled notification - Transporter assigned notification - Shipment picked up notification - Out for delivery notification - Delivery confirmation request
11. ERROR HANDLING: - Address validation errors - Transporter capacity exceeded - Slot not available - Distance calculation failures
Integration Note: Shipment status updates must trigger order status updates in M05. Use the same order_id as in M05.

13 Module 11: Shipment Tracking

Module Overview

ID: M11
Priority: 5
Dependencies: M01, M10
Database Required: Yes

13.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: tracking_events, location_updates, tracking_devices
- **API Base Path**: /api/v1/tracking
- **Authentication Required**: Yes
- **External Services**: GPS tracking, Map APIs

13.2 Database Schema

```
-- tracking_events table
CREATE TABLE tracking_events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_id UUID REFERENCES shipments(id) ON DELETE CASCADE,
    event_type VARCHAR(50) NOT NULL,
    event_code VARCHAR(20),
    description TEXT,
    location JSONB,
    reported_by UUID REFERENCES users(id),
    reported_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- location_updates table
CREATE TABLE location_updates (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_id UUID REFERENCES shipments(id) ON DELETE CASCADE,
    device_id VARCHAR(100),
    latitude DECIMAL(10, 8) NOT NULL,
    longitude DECIMAL(11, 8) NOT NULL,
    accuracy DECIMAL(5, 2),
    speed DECIMAL(5, 2),
    bearing DECIMAL(5, 2),
    altitude DECIMAL(8, 2),
    battery_level INTEGER,
    timestamp TIMESTAMP NOT NULL,
```

```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- tracking_devices table
CREATE TABLE tracking_devices (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    device_id VARCHAR(100) UNIQUE NOT NULL,
    transporter_id UUID REFERENCES users(id),
    vehicle_number VARCHAR(50),
    device_type VARCHAR(20) CHECK (device_type IN ('MOBILE', 'GPS_TRACKER'
    is_active BOOLEAN DEFAULT TRUE,
    last_seen_at TIMESTAMP,
    last_location JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

13.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a real-time shipment tracking system with GPS integration and event logging.
1. REQUIREMENTS: - Real-time location tracking - Tracking event logging - ETA calculation and updates - Delivery proof capture - Tracking device management
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /tracking/events - Log tracking event - GET /tracking/:shipment$_i d/events - Gettrackinghistory - POST/tracking/location - Updateshipmentlocation - GET/tracking/ : shipment$_i d/location - Getcurrentlocation - GET/tracking/ : shipment$_i d/eta - GetupdatedETA - POST/tracking/ : shipment$_i d/proof - Uploaddeliveryproof - GET/tracking/devices - Listtrackingdevices - POST/tracking/devices - Registertrackingdevice - PUT/tracking/devices/ : id - Updatedevicestatus
4. INTEGRATION: - Import shipment data from M10 - Use notification service from M08 - Use authMiddleware from M01
5. TRACKING EVENTS: - PICKUP$_S CHEDULED, PICKUP_C OMPLETED - DEPARTED_F ROM_W AREHOUSE - ARRIVED_A T_H UB, DEPARTED_F ROM_H UB - OUT_F OR_D ELIVERY, DELIVERY_A TTEMPTED - DELIVERED, DELIVERY_F AILED - DELAYED, ROUTE_C HANGED$
6. LOCATION TRACKING: - Support multiple tracking methods (GPS, mobile app, manual) - Store location updates with timestamps - Calculate distance traveled - Detect unusual stops or delays - Generate route visualization
7. ETA CALCULATION: - Calculate initial ETA based on distance - Update ETA based on current speed and traffic - Consider historical delivery times - Factor in time of day and day of week - Provide optimistic/pessimistic ETA ranges
8. DELIVERY PROOF: - Photo capture on delivery - Recipient signature capture - OTP-based delivery verification - GPS location validation at delivery point - Timestamp validation
9. BUSINESS LOGIC: - Auto-detect delivery events based on location - Alert for shipment delays (beyond threshold) - Generate tracking URL for sharing - Calculate actual vs estimated delivery time - Identify frequent delay locations
10. PERFORMANCE: - WebSocket for real-time updates - Geospatial indexing for location queries - Batch location updates processing - Cache frequent tracking queries
11. INTEGRATION POINTS: - Imports: M10 (shipment data), M08 (notifications) - Exports: trackShipment, getTrackingInfo functions - Shared: Location validation utilities - Webhooks: Real-time tracking updates
12. SECURITY: - Validate location updates authenticity - Rate limit location updates - Encrypt sensitive location data - Audit all tracking events
Integration Note: Tracking events must update shipment status in M10. Delivery confirmation must trigger order completion in M05.

14 Module 12: Transporter Management

Module Overview

ID: M12
Priority: 6
Dependencies: M01, M02, M08, M10
Database Required: Yes

14.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: transporters, vehicles, driver_profiles, transporter_performance
- **API Base Path**: /api/v1/transporters
- **Authentication Required**: Yes

14.2 Database Schema

```
-- transporters table
CREATE TABLE transporters (
    id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,
    company_name VARCHAR(255) NOT NULL,
    registration_number VARCHAR(100) UNIQUE NOT NULL,
    service_areas JSONB, -- Array of pincodes or districts
    vehicle_types JSONB, -- Array of vehicle types
    max_capacity_kg DECIMAL(10, 2),
    service_types JSONB, -- STANDARD, COLD_CHAIN, etc.
    insurance_details JSONB,
    license_valid_until DATE,
    rating DECIMAL(2,1) DEFAULT 0,
    total_shipments INTEGER DEFAULT 0,
    on_time_delivery_rate DECIMAL(5,2) DEFAULT 0,
    acceptance_rate DECIMAL(5,2) DEFAULT 0,
    is_verified BOOLEAN DEFAULT FALSE,
    verification_reason TEXT,
    is_active BOOLEAN DEFAULT TRUE,
    onboarding_date DATE DEFAULT CURRENT_DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- vehicles table
CREATE TABLE vehicles (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    transporter_id UUID REFERENCES transporters(id) ON DELETE CASCADE,
    registration_number VARCHAR(50) UNIQUE NOT NULL,
```

```
    vehicle_type VARCHAR(50) NOT NULL,
    make VARCHAR(100),
    model VARCHAR(100),
    year INTEGER,
    capacity_kg DECIMAL(10, 2) NOT NULL,
    dimensions JSONB,
    features JSONB, -- REFRIGERATED, LIFT_GATE, etc.
    insurance_number VARCHAR(100),
    insurance_valid_until DATE,
    fitness_certificate_number VARCHAR(100),
    fitness_valid_until DATE,
    is_active BOOLEAN DEFAULT TRUE,
    current_location JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- driver_profiles table
CREATE TABLE driver_profiles (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    transporter_id UUID REFERENCES transporters(id) ON DELETE CASCADE,
    driver_id UUID REFERENCES users(id) ON DELETE CASCADE,
    license_number VARCHAR(100) UNIQUE NOT NULL,
    license_type VARCHAR(50),
    license_valid_until DATE,
    experience_years INTEGER,
    contact_number VARCHAR(15),
    emergency_contact JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    current_assignment UUID REFERENCES shipments(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- transporter_performance table
CREATE TABLE transporter_performance (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    transporter_id UUID REFERENCES transporters(id) ON DELETE CASCADE,
    period DATE NOT NULL, -- First day of month
    total_shipments INTEGER DEFAULT 0,
    completed_shipments INTEGER DEFAULT 0,
    on_time_shipments INTEGER DEFAULT 0,
    damaged_shipments INTEGER DEFAULT 0,
    lost_shipments INTEGER DEFAULT 0,
    average_rating DECIMAL(2,1) DEFAULT 0,
    acceptance_rate DECIMAL(5,2) DEFAULT 0,
    cancellation_rate DECIMAL(5,2) DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    UNIQUE(transporter_id, period)
);
```

14.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a transporter management system for logistics partners with vehicle and driver management.
1. REQUIREMENTS: - Transporter registration and verification - Vehicle fleet management - Driver profile management - Performance tracking and analytics - Service area management
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /transporters/register - Register as transporter - GET /transporters - List transporters - GET /transporters/:id - Get transporter details - PUT /transporters/:id - Update transporter profile - POST /transporters/:id/verify - Verify transporter (ADMIN) - GET /transporters/:id/vehicles - List vehicles - POST /transporters/:id/vehicles - Add vehicle - PUT /vehicles/:id - Update vehicle - GET /transporters/:id/drivers - List drivers - POST /transporters/:id/drivers - Add driver - GET /transporters/:id/performance - Get performance metrics - GET /transporters/search - Search transporters by criteria
4. INTEGRATION: - Import user data from M01 - Import address data from M02 - Use notification service from M08 - Integrate with M10 for shipment assignments
5. TRANSPORTER VERIFICATION: - Document verification (license, insurance, registration) - Background check process - Vehicle inspection scheduling - Verification levels (BASIC, VERIFIED, PREMIUM) - Verification expiry and renewal
6. VEHICLE MANAGEMENT: - Support multiple vehicle types (truck, tempo, van, refrigerated) - Track vehicle availability - Maintenance scheduling - Insurance and fitness certificate tracking - Real-time location tracking
7. PERFORMANCE METRICS: - On-time delivery rate - Shipment acceptance rate - Customer rating average - Damage/loss rate - Response time to shipment requests
8. BUSINESS LOGIC: - Auto-assign shipments based on: * Service area match * Vehicle type requirements * Current capacity * Performance rating * Proximity to pickup - Calculate performance scores - Tier-based rewards system - Suspension for poor performance
9. SEARCH FILTERS: - By service area (pincode/district) - By vehicle type and capacity - By availability date/time - By performance rating - By service type (cold chain, livestock)
10. INTEGRATION POINTS: - Imports: M01 (users), M02 (addresses), M08 (notifications) - Exports: findTransporter, assignShipment functions - Shared: Performance calculation utilities - Webhooks: New shipment assignment notifications
11. NOTIFICATIONS: - New shipment assignments - Verification status updates - Performance milestone alerts - Document expiry reminders - Maintenance due alerts
Integration Note: Transporter user_id must match user ID from M01. Vehicle assignments must sync with M10 shipments.

15 Module 13: Bulk Order System

Module Overview

ID: M13
Priority: 6
Dependencies: M01, M03, M05, M06, M10
Database Required: Yes

15.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: bulk_orders, bulk_order_items, bulk_quotes, bulk_contracts
- **API Base Path**: /api/v1/bulk
- **Authentication Required**: Yes

15.2 Database Schema

```
-- bulk_orders table
CREATE TABLE bulk_orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_number VARCHAR(50) UNIQUE NOT NULL,
    entrepreneur_id UUID REFERENCES users(id) ON DELETE CASCADE,
    gaushala_id UUID REFERENCES users(id),
    contract_id UUID REFERENCES bulk_contracts(id),
    total_quantity DECIMAL(12, 2) NOT NULL,
    total_amount DECIMAL(12, 2) NOT NULL,
    delivery_schedule JSONB, -- Array of delivery dates and quantities
    status VARCHAR(20) DEFAULT 'DRAFT',
    payment_terms VARCHAR(100),
    delivery_terms TEXT,
    quality_specifications TEXT,
    special_instructions TEXT,
    start_date DATE,
    end_date DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- bulk_order_items table
CREATE TABLE bulk_order_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    bulk_order_id UUID REFERENCES bulk_orders(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id),
    quantity DECIMAL(10, 2) NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL,
```

```
    total_price DECIMAL(10, 2) NOT NULL,
    delivery_schedule JSONB,
    quality_grade VARCHAR(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- bulk_quotes table
CREATE TABLE bulk_quotes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    bulk_order_id UUID REFERENCES bulk_orders(id) ON DELETE CASCADE,
    gaushala_id UUID REFERENCES users(id),
    quoted_amount DECIMAL(12, 2) NOT NULL,
    validity_days INTEGER DEFAULT 7,
    terms_and_conditions TEXT,
    status VARCHAR(20) DEFAULT 'PENDING',
    submitted_at TIMESTAMP,
    accepted_at TIMESTAMP,
    rejected_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- bulk_contracts table
CREATE TABLE bulk_contracts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    contract_number VARCHAR(50) UNIQUE NOT NULL,
    entrepreneur_id UUID REFERENCES users(id),
    gaushala_id UUID REFERENCES users(id),
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    total_value DECIMAL(12, 2),
    payment_schedule JSONB,
    delivery_schedule JSONB,
    quality_standards TEXT,
    penalty_clauses TEXT,
    termination_terms TEXT,
    status VARCHAR(20) DEFAULT 'ACTIVE',
    signed_by_entrepreneur_at TIMESTAMP,
    signed_by_gaushala_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

15.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a bulk ordering system for large-scale purchases with contract management and scheduling.
1. REQUIREMENTS: - Bulk order creation and management - Quote request and response system - Contract management for recurring orders - Delivery scheduling for bulk quantities - Quality specifications management
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /bulk/requests - Create bulk order request - GET /bulk/requests - List bulk orders - GET /bulk/requests/:id - Get bulk order details - POST /bulk/requests/:id/quotes - Submit quote - GET /bulk/requests/:id/quotes - List quotes - POST /bulk/quotes/:id/accept - Accept quote - POST /bulk/contracts - Create contract - GET /bulk/contracts - List contracts - GET /bulk/-contracts/:id - Get contract details - POST /bulk/contracts/:id/renew - Renew contract - POST /bulk/schedule - Schedule bulk delivery - GET /bulk/schedule - View delivery schedule
4. INTEGRATION: - Import product data from M03 - Integrate with M05 for order processing - Use payment system from M06 - Use logistics from M10 - Use authMiddleware from M01
5. BULK ORDER WORKFLOW: 1. DRAFT → REQUESTED (entrepreneur submits) 2. REQUESTED → QUOTED (gaushala provides quote) 3. QUOTED → ACCEPTED (entrepreneur accepts) 4. ACCEPTED → CONTRACTED (contract signed) 5. CONTRACTED → ACTIVE (deliveries begin) 6. ACTIVE → COMPLETED (contract ends)
6. BUSINESS LOGIC: - Generate bulk order number: BO-{YYMMDD}-{6 digits} - Generate contract number: CON-{YYMMDD}-{6 digits} - Calculate bulk pricing discounts - Validate minimum order quantities for bulk - Schedule recurring deliveries - Track contract compliance
7. QUOTE MANAGEMENT: - Multiple gaushalas can quote on same request - Quote validity period (default 7 days) - Quote comparison tools - Auto-reject expired quotes - Quote revision capability
8. CONTRACT FEATURES: - Define delivery schedules (daily, weekly, monthly) - Set quality standards and testing protocols - Include penalty clauses for non-compliance - Automatic renewal options - Performance review triggers
9. DELIVERY SCHEDULING: - Create delivery calendar - Handle schedule changes - Track actual vs scheduled deliveries - Generate delivery notes - Quality check on delivery
10. INTEGRATION POINTS: - Imports: M03 (products), M05 (orders), M06 (payments), M10 (logistics) - Exports: createBulkOrder, scheduleDelivery functions - Shared: Contract template utilities - Webhooks: Quote submission, contract signing
11. NOTIFICATIONS: - New bulk request to gaushalas - Quote submission notifications - Contract signing reminders - Delivery schedule updates - Contract expiry warnings
Integration Note: Bulk orders create regular orders in M05 for each delivery. Contract payments sync with M06 payment schedules.

16 Module 14: Subscription Management

Module Overview

ID: M14
Priority: 7
Dependencies: M01, M03, M05, M06, M08
Database Required: Yes

16.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: subscriptions, subscription_plans, subscription_items, subscription_invoices
- **API Base Path**: /api/v1/subscriptions
- **Authentication Required**: Yes

16.2 Database Schema

```
-- subscription_plans table
CREATE TABLE subscription_plans (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    plan_code VARCHAR(100) UNIQUE NOT NULL,
    plan_name VARCHAR(255) NOT NULL,
    description TEXT,
    plan_type VARCHAR(20) CHECK (plan_type IN ('PRODUCT', 'DELIVERY', 'MEM
    target_user VARCHAR(20) CHECK (target_user IN ('ENTREPRENEUR', 'GAUSHAI
    billing_cycle VARCHAR(20) CHECK (billing_cycle IN ('DAILY', 'WEEKLY',
    price DECIMAL(10, 2) NOT NULL,
    currency VARCHAR(3) DEFAULT 'INR',
    trial_days INTEGER DEFAULT 0,
    is_active BOOLEAN DEFAULT TRUE,
    features JSONB,
    limitations JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- subscriptions table
CREATE TABLE subscriptions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    subscription_number VARCHAR(50) UNIQUE NOT NULL,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    plan_id UUID REFERENCES subscription_plans(id),
    status VARCHAR(20) DEFAULT 'ACTIVE',
    current_period_start DATE NOT NULL,
```

```
    current_period_end DATE NOT NULL,
    cancel_at_period_end BOOLEAN DEFAULT FALSE,
    canceled_at TIMESTAMP,
    trial_start DATE,
    trial_end DATE,
    quantity INTEGER DEFAULT 1,
    metadata JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- subscription_items table
CREATE TABLE subscription_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    subscription_id UUID REFERENCES subscriptions(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id),
    quantity DECIMAL(10, 2) NOT NULL,
    delivery_schedule JSONB,
    next_delivery_date DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- subscription_invoices table
CREATE TABLE subscription_invoices (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    subscription_id UUID REFERENCES subscriptions(id) ON DELETE CASCADE,
    invoice_number VARCHAR(50) UNIQUE NOT NULL,
    invoice_date DATE NOT NULL,
    due_date DATE,
    amount DECIMAL(10, 2) NOT NULL,
    tax_amount DECIMAL(10, 2) DEFAULT 0,
    status VARCHAR(20) DEFAULT 'PENDING',
    paid_at TIMESTAMP,
    payment_method VARCHAR(50),
    download_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

16.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a subscription management system for recurring orders and memberships.
1. REQUIREMENTS: - Subscription plan management - Recurring billing - Subscription lifecycle management - Delivery scheduling for subscriptions - Pause/resume functionality
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /plans - List subscription plans - POST /plans - Create plan (ADMIN) - PUT /plans/:id - Update plan (ADMIN) - POST /subscriptions - Create subscription - GET /subscriptions - List user subscriptions - GET /subscriptions/:id - Get subscription details - POST /subscriptions/:id/cancel - Cancel subscription - POST /subscriptions/:id/pause - Pause subscription - POST /subscriptions/:id/resume - Resume subscription - PUT /subscriptions/:id - Update subscription - GET /subscriptions/:id/invoices - List subscription invoices - GET /subscriptions/:id/items - List subscription items - PUT /subscriptions/items/:id - Update subscription item
4. INTEGRATION: - Import product data from M03 - Integrate with M05 for order generation - Use payment system from M06 - Use notification service from M08 - Use authMiddleware from M01
5. SUBSCRIPTION WORKFLOW: 1. ACTIVE → PAUSED (user pauses) 2. PAUSED → ACTIVE (user resumes) 3. ACTIVE → CANCELLED (user cancels) 4. CANCELLED → EXPIRED (period ends) 5. Any status → $\text{PAST}_DUE(payment failed)$
6. PLAN TYPES: - Product Subscription: Regular delivery of products - Delivery Subscription: Free/discounted delivery - Membership: Premium features access - Bundle: Multiple products/services
7. BUSINESS LOGIC: - Generate subscription number: SUB-{YYMMDD}-{6 digits} - Auto-generate orders at delivery dates - Handle prorated charges for mid-cycle changes - Process automatic payments on due dates - Send payment failure notifications
8. DELIVERY SCHEDULING: - Define delivery frequency (daily, weekly, etc.) - Specify delivery days of week - Set delivery time windows - Handle holiday/skip delivery - Reschedule missed deliveries
9. BILLING: - Recurring invoice generation - Automatic payment retry logic - Proration for plan changes - Trial period management - Coupon/discount application
10. INTEGRATION POINTS: - Imports: M03 (products), M05 (orders), M06 (payments), M08 (notifications) - Exports: createSubscriptionOrder, processRenewal functions - Shared: Subscription scheduling utilities - Webhooks: Subscription lifecycle events
11. CRON JOBS: - Daily: Generate delivery orders - Daily: Process renewals - Daily: Send upcoming delivery reminders - Hourly: Retry failed payments - Monthly: Generate usage reports
Integration Note: Subscriptions auto-create orders in M05. Failed payments trigger notifications via M08 and may pause subscriptions.

17 Module 15: Quality Certification

Module Overview

ID: M15
Priority: 7
Dependencies: M01, M03
Database Required: Yes

17.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: quality_certificates, lab_tests, quality_standards, certification_bodies
- **API Base Path**: /api/v1/quality
- **Authentication Required**: Yes

17.2 Database Schema

```
-- quality_certificates table
CREATE TABLE quality_certificates (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    certificate_number VARCHAR(100) UNIQUE NOT NULL,
    gaushala_id UUID REFERENCES users(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    certificate_type VARCHAR(50) CHECK (certificate_type IN ('ORGANIC', 'FS
    issuing_body UUID REFERENCES certification_bodies(id),
    issue_date DATE NOT NULL,
    expiry_date DATE,
    certificate_url VARCHAR(500) NOT NULL,
    verification_status VARCHAR(20) DEFAULT 'PENDING',
    verified_by UUID REFERENCES users(id),
    verified_at TIMESTAMP,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- lab_tests table
CREATE TABLE lab_tests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    test_number VARCHAR(100) UNIQUE NOT NULL,
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    batch_number VARCHAR(100),
    test_type VARCHAR(100) NOT NULL,
    lab_name VARCHAR(255),
    test_date DATE NOT NULL,
```

```
    parameters JSONB NOT NULL, -- {"fat_content": "4.5%", "protein": "3.2%
    result VARCHAR(20) CHECK (result IN ('PASS', 'FAIL', 'INCONCLUSIVE')),
    report_url VARCHAR(500),
    tested_by VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- quality_standards table
CREATE TABLE quality_standards (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    standard_code VARCHAR(50) UNIQUE NOT NULL,
    standard_name VARCHAR(255) NOT NULL,
    category VARCHAR(100),
    description TEXT,
    parameters JSONB NOT NULL, -- Minimum/maximum values
    applicable_products JSONB, -- Product categories
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- certification_bodies table
CREATE TABLE certification_bodies (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    body_name VARCHAR(255) UNIQUE NOT NULL,
    accreditation_number VARCHAR(100),
    accreditation_valid_until DATE,
    contact_info JSONB,
    is_verified BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

17.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a quality certification and testing module for gaushala products.
1. REQUIREMENTS: - Quality certificate management - Laboratory test tracking - Quality standards definition - Certification body management - Quality badge display system
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /certificates - Upload quality certificate - GET /certificates - List certificates - GET /certificates/:id - Get certificate details - POST /certificates/:id/verify - Verify certificate (ADMIN) - GET /products/:id/certificates - Get product certificates - POST /lab-tests - Record lab test results - GET /lab-tests - List lab tests - GET /products/:id/lab-tests - Get product test history - GET /standards - List quality standards - POST /standards - Create standard (ADMIN) - GET /certification-bodies - List certification bodies - POST /certification-bodies - Add certification body (ADMIN) - GET /quality-badges/:entity$_id - Get quality badges for display$
4. INTEGRATION: - Import product data from M03 - Use authMiddleware from M01 - Display quality badges in M04 marketplace
5. CERTIFICATE TYPES: - Organic Certification - FSSAI License - ISO Certifications - HACCP (Food Safety) - GMP (Good Manufacturing Practices) - Custom Quality Certificates
6. LAB TEST PARAMETERS: - For Milk: Fat content, SNF, protein, acidity, bacteria count - For Ghee: Moisture, FFA, peroxide value - For Dung Products: Moisture, nutrient content - For Compost: NPK values, organic matter
7. BUSINESS LOGIC: - Generate certificate number: QC-{YYMMDD}-{6 digits} - Generate test number: LT-{YYMMDD}-{6 digits} - Auto-check certificate expiry and send alerts - Validate test results against quality standards - Calculate quality score based on certificates and tests
8. QUALITY BADGES: - Organic Certified (green badge) - Lab Tested (blue badge) - Premium Quality (gold badge) - FSSAI Approved (red badge) - ISO Certified (purple badge) - Quality scores (1-5 stars)
9. VERIFICATION WORKFLOW: 1. UPLOADED $\rightarrow$ PENDING$_REVIEW2.PENDING_REVIEWßUNDER_VERIFICATION3.UNDER_VERIFIC$
$Manualverificationforsensitivecertificates - Auto - verificationforknowncertificationbodies$
10. INTEGRATION POINTS: - Imports: M03 (products), M01 (auth) - Exports: getQualityBadges, checkProductQuality functions - Shared: Certificate validation utilities - Webhooks: Certificate expiry alerts
11. NOTIFICATIONS: - Certificate verification status updates - Expiry warnings (30 days, 7 days, expired) - New lab test results - Quality standard updates
Integration Note: Quality badges from this module are displayed in M04 marketplace. Product quality affects search ranking.

18 Module 16: Government Scheme Integration

Module Overview

ID: M16
Priority: 8
Dependencies: M01, M02, M08
Database Required: Yes

18.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: government_schemes, scheme_applications, scheme_benefits, subsidy_transactions
- **API Base Path**: /api/v1/government
- **Authentication Required**: Yes
- **External APIs**: Government portals, DBT (Direct Benefit Transfer)

18.2 Database Schema

```
-- government_schemes table
CREATE TABLE government_schemes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    scheme_code VARCHAR(100) UNIQUE NOT NULL,
    scheme_name VARCHAR(255) NOT NULL,
    department VARCHAR(255),
    description TEXT,
    eligibility_criteria JSONB,
    benefits JSONB,
    application_process TEXT,
    required_documents JSONB,
    start_date DATE,
    end_date DATE,
    is_active BOOLEAN DEFAULT TRUE,
    application_url VARCHAR(500),
    contact_info JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- scheme_applications table
CREATE TABLE scheme_applications (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    application_number VARCHAR(100) UNIQUE NOT NULL,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
```

```
    scheme_id UUID REFERENCES government_schemes(id),
    application_data JSONB NOT NULL,
    documents JSONB,
    status VARCHAR(20) DEFAULT 'DRAFT',
    submitted_at TIMESTAMP,
    verified_by UUID REFERENCES users(id),
    verification_status VARCHAR(20) DEFAULT 'PENDING',
    verification_notes TEXT,
    approved_at TIMESTAMP,
    rejection_reason TEXT,
    government_reference_id VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- scheme_benefits table
CREATE TABLE scheme_benefits (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    application_id UUID REFERENCES scheme_applications(id) ON DELETE CASCAD
    benefit_type VARCHAR(50) CHECK (benefit_type IN ('SUBSIDY', 'GRANT', 'L
    amount DECIMAL(12, 2),
    description TEXT,
    disbursement_date DATE,
    disbursement_status VARCHAR(20) DEFAULT 'PENDING',
    transaction_reference VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- subsidy_transactions table
CREATE TABLE subsidy_transactions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    order_id UUID REFERENCES orders(id),
    scheme_id UUID REFERENCES government_schemes(id),
    subsidy_amount DECIMAL(10, 2) NOT NULL,
    subsidy_percentage DECIMAL(5, 2),
    transaction_type VARCHAR(20) CHECK (transaction_type IN ('DIRECT', 'RED
    status VARCHAR(20) DEFAULT 'PENDING',
    utr_number VARCHAR(100),
    bank_details JSONB,
    processed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

18.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a government scheme integration module for subsidies and benefits.
1. REQUIREMENTS: - Government scheme catalog - Application management for schemes - Benefit tracking and disbursement - Subsidy application to orders - Integration with government portals
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /schemes - List available schemes - GET /schemes/:id - Get scheme details - POST /schemes/:id/apply - Apply for scheme - GET /applications - List user applications - GET /applications/:id - Get application details - PUT /applications/:id - Update application - POST /applications/:id/submit - Submit application - GET /applications/:id/status - Check application status - POST /applications/:id/verify - Verify application (ADMIN) - GET /benefits - List user benefits - POST /subsidy/apply - Apply subsidy to order - GET /subsidy/transactions - List subsidy transactions - GET /schemes/eligibility/:user$_id - Check eligibility for schemes$
4. INTEGRATION: - Import user data from M01 and M02 - Integrate with M05 for order subsidies - Use notification service from M08 - Connect to government APIs where available
5. SCHEME TYPES: - Subsidy schemes (percentage/amount off) - Grant schemes (direct funding) - Loan schemes (low interest loans) - Training programs (skill development) - Equipment schemes (subsidized equipment)
6. APPLICATION WORKFLOW: 1. DRAFT → SUBMITTED (user submits) 2. SUBMITTED → UNDER$_REVIEW$ (admin reviews) 3. $UNDER_REVIEW$ ß $APPROVED or REJECTED 4. APPRO$
7. BUSINESS LOGIC: - Generate application number: APP-{YYMMDD}-{6 digits} - Auto-check eligibility based on user profile - Document verification checklist - Subsidy calculation based on scheme rules - Integration with DBT (Direct Benefit Transfer)
8. SUBSIDY APPLICATION: - Apply subsidy at checkout - Validate subsidy eligibility per order - Calculate final amount after subsidy - Track subsidy utilization - Generate subsidy certificates
9. GOVERNMENT API INTEGRATION: - Scheme data synchronization - Application status updates - Benefit disbursement tracking - Document verification APIs - Real-time subsidy validation
10. INTEGRATION POINTS: - Imports: M01 (users), M02 (profile), M05 (orders), M08 (notifications) - Exports: applySubsidy, checkEligibility functions - Shared: Government API client utilities - Webhooks: Application status updates
11. NOTIFICATIONS: - New scheme announcements - Application submission confirmation - Application status updates - Benefit disbursement alerts - Subsidy application to orders
Integration Note: Subsidies affect final order amount in M05. Benefit disbursements may integrate with M06 payments.

19 Module 17: Demand Forecasting

Module Overview

ID: M17
Priority: 8
Dependencies: M03, M05, M09
Database Required: Yes

19.1 Technical Specifications

- **Backend**: Python/FastAPI (for ML) + Node.js for API
- **Database Tables**: demand_forecasts, historical_demand, forecast_models, seasonality_patterns
- **API Base Path**: /api/v1/forecasting
- **Authentication Required**: Yes

19.2 Database Schema

```
-- demand_forecasts table
CREATE TABLE demand_forecasts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    gaushala_id UUID REFERENCES users(id),
    forecast_date DATE NOT NULL,
    period_type VARCHAR(20) CHECK (period_type IN ('DAILY', 'WEEKLY', 'MON
    forecast_quantity DECIMAL(10, 2) NOT NULL,
    confidence_interval_lower DECIMAL(10, 2),
    confidence_interval_upper DECIMAL(10, 2),
    model_version VARCHAR(50),
    generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(product_id, forecast_date, period_type)
);

-- historical_demand table
CREATE TABLE historical_demand (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    date DATE NOT NULL,
    actual_quantity DECIMAL(10, 2) NOT NULL,
    order_count INTEGER DEFAULT 0,
    average_price DECIMAL(10, 2),
    weather_conditions JSONB,
    holiday_flag BOOLEAN DEFAULT FALSE,
    special_event VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    UNIQUE(product_id, date)
);

-- forecast_models table
CREATE TABLE forecast_models (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    model_name VARCHAR(100) NOT NULL,
    model_type VARCHAR(50) CHECK (model_type IN ('ARIMA', 'PROPHET', 'LSTM
    product_category VARCHAR(100),
    hyperparameters JSONB,
    accuracy_metrics JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    trained_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- seasonality_patterns table
CREATE TABLE seasonality_patterns (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    product_category VARCHAR(100) NOT NULL,
    pattern_type VARCHAR(50) CHECK (pattern_type IN ('WEEKLY', 'MONTHLY',
    pattern_data JSONB NOT NULL, -- { "monday": 1.2, "tuesday": 0.9, ... }
    confidence_score DECIMAL(3, 2),
    detected_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

19.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a demand forecasting module using machine learning to predict product demand.
1. REQUIREMENTS: - Historical demand data collection - Multiple forecasting models - Seasonality pattern detection - Forecast accuracy tracking - Inventory recommendation engine
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /forecasts/product/:id - Get product forecasts - GET /forecasts/gaushala/:id - Get gaushala forecasts - POST /forecasts/generate - Generate forecasts (cron/trigger) - GET /historical/:product$_i d$ $-$ $Gethistoricaldemand$ $-$ $POST/historical/$: $product_i d$ $-$ $Recordhistoricaldemand$ $-$ $GET/models$ $-$ $Listforecastingmodels$ $-$ $POST/models/train$ $-$ $Trainnewmodel$ $-$ $GET/patterns - Listseasonalitypatterns - GET/recommendations/inventory -$ $Getinventoryrecommendations - GET/accuracy - Getforecastaccuracymetrics$
4. INTEGRATION: - Import product data from M03 - Import order data from M05 - Use analytics from M09 for insights
5. FORECASTING MODELS: - ARIMA (AutoRegressive Integrated Moving Average) - Prophet (Facebook's time series) - LSTM (Long Short-Term Memory neural networks) - XGBoost (gradient boosting) - Ensemble (combination of models)
6. BUSINESS LOGIC: - Collect historical data from orders - Include external factors: weather, holidays, events - Calculate seasonality patterns - Generate confidence intervals - Update forecasts regularly (daily/weekly)
7. FEATURE ENGINEERING: - Day of week patterns - Monthly trends - Holiday effects - Price elasticity - Competitor activity - Weather impact (for dairy products)
8. FORECAST USAGE: - Inventory planning for gaushalas - Production scheduling - Price optimization suggestions - Marketing campaign timing - Resource allocation
9. ACCURACY MONITORING: - Track forecast vs actual - Calculate MAPE (Mean Absolute Percentage Error) - Identify model drift - Auto-retrain models when accuracy drops - A/B test different models
10. INTEGRATION POINTS: - Imports: M03 (products), M05 (orders), M09 (analytics) - Exports: getDemandForecast, getInventoryRecommendation functions - Shared: Forecasting utilities, ML model serving - Webhooks: Forecast generation completion
11. CRON JOBS: - Daily: Collect historical data - Weekly: Generate forecasts - Monthly: Retrain models - Daily: Calculate accuracy metrics - Hourly: Update real-time predictions
12. VISUALIZATION: - Forecast charts with confidence bands - Historical vs forecast comparison - Seasonality pattern visualization - Accuracy trend charts
Integration Note: Forecasts help gaushalas plan inventory in M03. Recommendations appear in M09 dashboard.

20 Module 18: Route Optimization

Module Overview

ID: M18
Priority: 8
Dependencies: M01, M10, M11
Database Required: Yes

20.1 Technical Specifications

- **Backend**: Python/FastAPI (for optimization) + Node.js for API
- **Database Tables**: optimized_routes, route_segments, optimization_requests, traffic_patterns
- **API Base Path**: /api/v1/routing
- **Authentication Required**: Yes
- **External Services**: Map APIs, Traffic data

20.2 Database Schema

```
-- optimized_routes table
CREATE TABLE optimized_routes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    route_id VARCHAR(100) UNIQUE NOT NULL,
    transporter_id UUID REFERENCES users(id),
    vehicle_id UUID REFERENCES vehicles(id),
    start_location JSONB NOT NULL,
    end_location JSONB,
    waypoints JSONB, -- Array of locations
    optimized_waypoints JSONB,
    total_distance DECIMAL(8, 2) NOT NULL, -- in km
    estimated_duration INTEGER NOT NULL, -- in seconds
    actual_duration INTEGER,
    fuel_estimate DECIMAL(6, 2), -- in liters
    cost_estimate DECIMAL(8, 2),
    status VARCHAR(20) DEFAULT 'PLANNED',
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    optimization_parameters JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- route_segments table
CREATE TABLE route_segments (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    route_id UUID REFERENCES optimized_routes(id) ON DELETE CASCADE,
    segment_order INTEGER NOT NULL,
    shipment_id UUID REFERENCES shipments(id),
    pickup_location JSONB,
    delivery_location JSONB,
    estimated_distance DECIMAL(6, 2),
    estimated_duration INTEGER,
    actual_distance DECIMAL(6, 2),
    actual_duration INTEGER,
    status VARCHAR(20) DEFAULT 'PENDING',
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- optimization_requests table
CREATE TABLE optimization_requests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    request_id VARCHAR(100) UNIQUE NOT NULL,
    shipments JSONB NOT NULL, -- Array of shipment IDs
    constraints JSONB, -- Time windows, capacity, etc.
    optimization_type VARCHAR(50) CHECK (optimization_type IN ('TIME', 'DIS
    status VARCHAR(20) DEFAULT 'PENDING',
    result JSONB,
    processing_time INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- traffic_patterns table
CREATE TABLE traffic_patterns (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    location_hash VARCHAR(100) NOT NULL,
    day_of_week INTEGER CHECK (day_of_week >= 0 AND day_of_week <= 6),
    hour_of_day INTEGER CHECK (hour_of_day >= 0 AND hour_of_day <= 23),
    traffic_level VARCHAR(20) CHECK (traffic_level IN ('LOW', 'MEDIUM', 'HI
    average_speed DECIMAL(5, 2), -- km/h
    confidence_score DECIMAL(3, 2),
    sample_count INTEGER DEFAULT 0,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(location_hash, day_of_week, hour_of_day)
);
```

20.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a route optimization module for logistics with traffic-aware routing and multi-stop optimization.
1. REQUIREMENTS: - Multi-stop route optimization - Traffic pattern analysis - Real-time route adjustment - Fuel and cost optimization - Route planning and scheduling
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /routes/optimize - Optimize route for shipments - GET /routes/:id - Get optimized route details - PUT /routes/:id - Update route (real-time adjustments) - POST /routes/:id/start - Start route execution - POST /routes/:id/complete - Complete route - GET /routes/transporter/:id - List transporter routes - GET /routes/active - Get active routes - POST /traffic/update - Update traffic data - GET /traffic/predict - Predict traffic for route - GET /optimization/history - List optimization requests - POST /routes/:id/reroute - Re-route due to traffic/obstacles
4. INTEGRATION: - Import shipment data from M10 - Import location data from M11 - Use authMiddleware from M01
5. OPTIMIZATION ALGORITHMS: - Traveling Salesman Problem (TSP) solvers - Vehicle Routing Problem (VRP) with time windows - Clarke-Wright savings algorithm - Genetic algorithms for complex routes - Real-time traffic adjustment algorithms
6. OPTIMIZATION OBJECTIVES: - Minimize total distance - Minimize total time - Minimize fuel consumption - Balance workload among drivers - Meet delivery time windows - Consider vehicle capacity constraints
7. TRAFFIC INTEGRATION: - Historical traffic pattern database - Real-time traffic API integration - Traffic prediction based on time/day - Incident detection and avoidance - Alternative route suggestions
8. BUSINESS LOGIC: - Generate route ID: RT-{YYMMDD}-{6 digits} - Consider delivery time windows - Account for loading/unloading time - Handle priority shipments - Optimize for fuel efficiency - Consider road restrictions (weight, height)
9. REAL-TIME FEATURES: - Live route tracking - ETA updates based on traffic - Re-routing for accidents/roadblocks - Driver communication for changes - Customer notification for delays
10. INTEGRATION POINTS: - Imports: M10 (shipments), M11 (tracking), M01 (auth) - Exports: optimizeRoute, getOptimalRoute functions - Shared: Map integration utilities, optimization algorithms - Webhooks: Route optimization completion, traffic alerts
11. VISUALIZATION: - Interactive route maps - Stop sequence visualization - Time window displays - Traffic overlay on routes - Performance metrics dashboard
12. PERFORMANCE: - Cache optimized routes - Pre-compute common routes - Batch optimization for similar shipments - Parallel processing for large optimizations
Integration Note: Optimized routes are used in M10 for shipment assignments. Real-time updates sync with M11 tracking.

21 Module 19: Smart Matching Engine

Module Overview

ID: M19
Priority: 9
Dependencies: M01, M02, M03, M04, M07, M15
Database Required: Yes

21.1 Technical Specifications

- **Backend**: Python/FastAPI (for ML) + Node.js for API
- **Database Tables**: matching_preferences, match_scores, recommendation_logs, matching_rules
- **API Base Path**: /api/v1/matching
- **Authentication Required**: Yes

21.2 Database Schema

```
-- matching_preferences table
CREATE TABLE matching_preferences (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    preference_type VARCHAR(50) CHECK (preference_type IN ('SEARCH', 'RECOM
    entity_type VARCHAR(20) CHECK (entity_type IN ('PRODUCT', 'GAUSHALA')),
    preferences JSONB NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, preference_type, entity_type)
);

-- match_scores table
CREATE TABLE match_scores (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    entrepreneur_id UUID REFERENCES users(id) ON DELETE CASCADE,
    gaushala_id UUID REFERENCES users(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id) ON DELETE CASCADE,
    score DECIMAL(4, 3) CHECK (score >= 0 AND score <= 1),
    score_components JSONB, -- {"proximity": 0.8, "price": 0.6, "quality":
    match_reasons JSONB,
    last_calculated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(entrepreneur_id, gaushala_id, product_id)
);
```

```
-- recommendation_logs table
CREATE TABLE recommendation_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    recommendation_type VARCHAR(50),
    recommended_entity_id UUID NOT NULL,
    entity_type VARCHAR(20),
    position INTEGER,
    was_clicked BOOLEAN DEFAULT FALSE,
    click_timestamp TIMESTAMP,
    converted_to_order BOOLEAN DEFAULT FALSE,
    order_id UUID REFERENCES orders(id),
    session_id VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- matching_rules table
CREATE TABLE matching_rules (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    rule_name VARCHAR(100) NOT NULL,
    rule_type VARCHAR(50) CHECK (rule_type IN ('PROXIMITY', 'PRICE', 'QUAL
    weight DECIMAL(3, 2) CHECK (weight >= 0 AND weight <= 1),
    conditions JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    priority INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

21.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a smart matching engine using ML to connect entrepreneurs with gaushalas based on multiple criteria.
1. REQUIREMENTS: - Multi-factor matching algorithm - Personalized recommendations - Preference learning over time - Match scoring and ranking - Recommendation tracking and optimization
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /matches/entrepreneur/:id - Get matches for entrepreneur - GET /matches/gaushala/:id - Get matches for gaushala - POST /matches/calculate - Calculate match scores (batch) - GET /recommendations - Get personalized recommendations - POST /recommendations/feedback - Provide feedback on recommendations - GET /preferences - Get user matching preferences - PUT /preferences - Update matching preferences - GET /rules - List matching rules (ADMIN) - POST /rules - Create matching rule (ADMIN) - PUT /rules/:id - Update matching rule (ADMIN) - GET /analytics/recommendations - Get recommendation analytics
4. INTEGRATION: - Import product data from M03 - Import location data from M02 - Import ratings from M07 - Import quality data from M15 - Use marketplace data from M04
5. MATCHING FACTORS: - Proximity (distance between locations) - Price competitiveness - Quality ratings and certifications - Historical order success rate - Availability and delivery time - Business size compatibility - Past interaction history - Entrepreneur preferences
6. MATCHING ALGORITHMS: - Collaborative filtering - Content-based filtering - Hybrid recommendation systems - Reinforcement learning for feedback - A/B testing for algorithm optimization
7. BUSINESS LOGIC: - Calculate match scores (0-1 scale) - Weight different factors based on user preferences - Learn from user interactions (clicks, orders) - Adjust recommendations based on feedback - Handle cold start problem for new users/entities
8. RECOMMENDATION TYPES: - Product recommendations (based on browsing history) - Gaushala recommendations (based on preferences) - Cross-sell recommendations (complementary products) - Time-based recommendations (seasonal products) - Trending recommendations (popular in area)
9. PERSONALIZATION: - User preference learning - Session-based recommendations - Context-aware recommendations (location, time) - Ensemble of multiple recommendation strategies - Explainable recommendations (why recommended)
10. INTEGRATION POINTS: - Imports: M03, M02, M07, M15, M04 - Exports: getRecommendations, calculateMatches functions - Shared: Matching algorithms, preference learning - Webhooks: New match opportunities, recommendation updates
11. ANALYTICS: - Click-through rate (CTR) tracking - Conversion rate analysis - Recommendation effectiveness - User engagement metrics - A/B test results
12. PERFORMANCE: - Cache calculated matches - Incremental score updates - Batch processing for large datasets - Real-time scoring for new entities
Integration Note: Recommendations appear in M04 marketplace. Match scores influence search rankings.

22 Module 20: Cold Chain Management

Module Overview

ID: M20
Priority: 9
Dependencies: M01, M03, M10, M11, M12
Database Required: Yes

22.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: cold_chain_shipments, temperature_logs, cold_storage_facilities, temperature_alerts
- **API Base Path**: /api/v1/cold-chain
- **Authentication Required**: Yes
- **IoT Integration**: Temperature sensors, GPS trackers

22.2 Database Schema

```
-- cold_chain_shipments table
CREATE TABLE cold_chain_shipments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_id UUID REFERENCES shipments(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id),
    required_temperature_min DECIMAL(5, 2) NOT NULL, -- in Celsius
    required_temperature_max DECIMAL(5, 2) NOT NULL,
    temperature_unit VARCHAR(10) DEFAULT 'CELSIUS',
    max_temperature_exposure_time INTEGER, -- in minutes
    packaging_type VARCHAR(50),
    insulation_quality VARCHAR(20),
    cold_chain_compliance_score DECIMAL(3, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- temperature_logs table
CREATE TABLE temperature_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_id UUID REFERENCES shipments(id) ON DELETE CASCADE,
    device_id VARCHAR(100),
    temperature DECIMAL(5, 2) NOT NULL,
    humidity DECIMAL(5, 2),
    timestamp TIMESTAMP NOT NULL,
    location JSONB,
```

```
    battery_level INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- cold_storage_facilities table
CREATE TABLE cold_storage_facilities (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    facility_name VARCHAR(255) NOT NULL,
    owner_id UUID REFERENCES users(id),
    location JSONB NOT NULL,
    facility_type VARCHAR(50) CHECK (facility_type IN ('WAREHOUSE', 'TRUCK
    temperature_zones JSONB, -- {"freezer": -18, "chiller": 4, "ambient":
    capacity_cubic_meters DECIMAL(8, 2),
    available_capacity DECIMAL(8, 2),
    features JSONB, -- {"power_backup": true, "monitoring": true, ...}
    is_available BOOLEAN DEFAULT TRUE,
    hourly_rate DECIMAL(8, 2),
    certification JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- temperature_alerts table
CREATE TABLE temperature_alerts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_id UUID REFERENCES shipments(id) ON DELETE CASCADE,
    alert_type VARCHAR(50) CHECK (alert_type IN ('HIGH_TEMP', 'LOW_TEMP',
    threshold_value DECIMAL(5, 2),
    actual_value DECIMAL(5, 2),
    duration_exceeded INTEGER, -- seconds
    severity VARCHAR(20) DEFAULT 'WARNING',
    acknowledged_by UUID REFERENCES users(id),
    acknowledged_at TIMESTAMP,
    resolved_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

22.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a cold chain management system for temperature-sensitive products with IoT integration.
1. REQUIREMENTS: - Temperature monitoring for shipments - Cold storage facility management - Temperature compliance tracking - Real-time alert system - Cold chain quality assurance
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /cold-chain/shipments - Create cold chain shipment - GET /cold-chain/shipments/:id - Get cold chain shipment details - POST /temperature/logs - Record temperature data (IoT/webhook) - GET /temperature/shipment/:id/logs - Get temperature history - GET /temperature/-shipment/:id/compliance - Get compliance report - GET /cold-storage/facilities - List cold storage facilities - POST /cold-storage/facilities - Add cold storage facility - POST /cold-storage/book - Book cold storage space - GET /alerts - List temperature alerts - POST /alerts/:id/acknowledge - Acknowledge alert - POST /alerts/:id/resolve - Resolve alert - GET /cold-chain/quality/:shipment$_id$ − $Getqualityassurancereport$
4. INTEGRATION: - Import shipment data from M10 - Import product data from M03 - Import tracking data from M11 - Import transporter data from M12 - Use notification service from M08
5. TEMPERATURE MONITORING: - Real-time temperature tracking - Humidity monitoring - Device battery level tracking - GPS location with temperature - Data logging interval configuration
6. ALERT SYSTEM: - High temperature alerts - Low temperature alerts - Humidity out of range alerts - Device offline alerts - Prolonged exposure alerts - Escalation rules for critical alerts
7. BUSINESS LOGIC: - Calculate cold chain compliance score - Validate temperature requirements per product type - Monitor maximum exposure time - Generate quality assurance certificates - Calculate shelf life impact due to temperature deviations
8. COLD STORAGE MANAGEMENT: - Facility availability booking - Temperature zone management - Capacity planning and optimization - Rate calculation and billing - Certification and compliance tracking
9. COMPLIANCE REPORTING: - Temperature compliance percentage - Time outside temperature range - Impact on product quality - Regulatory compliance documentation - Customer-facing quality reports
10. INTEGRATION POINTS: - Imports: M10, M03, M11, M12, M08 - Exports: monitorTemperature, checkCompliance functions - Shared: Temperature validation utilities, IoT device management - Webhooks: Temperature alerts, compliance status updates
11. IoT INTEGRATION: - Device registration and authentication - Data encryption for transmission - Device health monitoring - Firmware update management - Multi-vendor device support
12. NOTIFICATIONS: - Temperature threshold breaches - Device battery low warnings - Cold storage booking confirmations - Compliance report generation - Quality assurance alerts
Integration Note: Cold chain compliance affects product quality ratings in M15. Alerts trigger notifications via M08.

23 Module 21: Live Stock Management

Module Overview

ID: M21
Priority: 9
Dependencies: M01, M02, M10, M12, M15
Database Required: Yes

23.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: livestock, animal_health, livestock_shipments, veterinary_certificates
- **API Base Path**: /api/v1/livestock
- **Authentication Required**: Yes

23.2 Database Schema

```
-- livestock table
CREATE TABLE livestock (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    gaushala_id UUID REFERENCES users(id) ON DELETE CASCADE,
    animal_id VARCHAR(100) UNIQUE NOT NULL,
    animal_type VARCHAR(50) CHECK (animal_type IN ('COW', 'BULL', 'CALF',
    breed VARCHAR(100),
    age_months INTEGER,
    gender VARCHAR(10) CHECK (gender IN ('MALE', 'FEMALE')),
    color_markings TEXT,
    weight_kg DECIMAL(6, 2),
    health_status VARCHAR(20) DEFAULT 'HEALTHY',
    vaccination_status VARCHAR(20) DEFAULT 'UP_TO_DATE',
    is_available_for_sale BOOLEAN DEFAULT FALSE,
    price DECIMAL(10, 2),
    special_requirements TEXT,
    photos JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- animal_health table
CREATE TABLE animal_health (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    livestock_id UUID REFERENCES livestock(id) ON DELETE CASCADE,
    checkup_date DATE NOT NULL,
    veterinarian_name VARCHAR(255),
    weight_kg DECIMAL(6, 2),
```

```
    temperature DECIMAL(5, 2),
    heart_rate INTEGER,
    respiration_rate INTEGER,
    general_condition TEXT,
    diagnosis TEXT,
    treatment TEXT,
    medication_administered TEXT,
    next_checkup_date DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- livestock_shipments table
CREATE TABLE livestock_shipments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    shipment_id UUID REFERENCES shipments(id) ON DELETE CASCADE,
    livestock_id UUID REFERENCES livestock(id),
    transport_requirements JSONB,
    feeding_schedule JSONB,
    rest_stops JSONB,
    water_requirements TEXT,
    special_handling TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- veterinary_certificates table
CREATE TABLE veterinary_certificates (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    livestock_id UUID REFERENCES livestock(id) ON DELETE CASCADE,
    certificate_type VARCHAR(50) CHECK (certificate_type IN ('HEALTH', 'VA
    certificate_number VARCHAR(100),
    issuing_veterinarian VARCHAR(255),
    issue_date DATE NOT NULL,
    expiry_date DATE,
    certificate_url VARCHAR(500),
    verified BOOLEAN DEFAULT FALSE,
    verified_by UUID REFERENCES users(id),
    verified_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

23.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a livestock management system for live animal sales and transportation.
1. REQUIREMENTS: - Live animal inventory management - Animal health tracking - Veterinary certificate management - Specialized livestock transportation - Animal welfare compliance
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /livestock - Add animal to inventory - GET /livestock - List animals with filters - GET /livestock/:id - Get animal details - PUT /livestock/:id - Update animal information - POST /livestock/:id/health - Record health checkup - GET /livestock/:id/health - Get health history - POST /livestock/:id/certificates - Upload veterinary certificate - GET /livestock/:id/certificates - List certificates - POST /livestock/shipment - Create livestock shipment - GET /livestock/shipment/:id - Get shipment details - PUT /livestock/shipment/:id/requirements - Update transport requirements - GET /livestock/available - List animals available for sale - POST /livestock/:id/sell - Mark animal as sold
4. INTEGRATION: - Import gaushala data from M01 - Import location data from M02 - Integrate with M10 for shipments - Integrate with M12 for specialized transporters - Use quality certification from M15
5. ANIMAL TYPES: - Cows (milk producing) - Bulls (breeding) - Calves (young stock) - Oxen (draft animals) - Special breeds (indigenous varieties)
6. HEALTH MONITORING: - Regular health checkups - Vaccination tracking - Weight monitoring - Temperature tracking - Medical history
7. TRANSPORT REQUIREMENTS: - Special vehicle requirements - Feeding and watering schedule - Rest stop requirements - Temperature control - Handling instructions - Emergency procedures
8. BUSINESS LOGIC: - Generate animal ID: AN-{gaushala code}-{sequence} - Validate animal health before sale - Calculate appropriate pricing based on breed, age, health - Track animal lineage and breeding history - Ensure welfare compliance during transport
9. CERTIFICATES REQUIRED: - Health certificate (mandatory for transport) - Vaccination certificate - Breed certification - Transport fitness certificate - Ownership transfer documents
10. INTEGRATION POINTS: - Imports: M01, M02, M10, M12, M15 - Exports: createLivestockShipment, validateAnimalHealth functions - Shared: Animal welfare compliance utilities - Webhooks: Animal health alerts, certificate expiry warnings
11. COMPLIANCE: - Animal welfare regulations - Transport safety standards - Health certification requirements - Quarantine regulations - Interstate movement permits
12. NOTIFICATIONS: - Health checkup reminders - Certificate expiry warnings - Sale inquiries and offers - Transport booking confirmations - Welfare compliance alerts
Integration Note: Livestock shipments are special cases in M10. Health certificates integrate with M15 quality system.

24 Module 22: Waste-to-Value Tracking

Module Overview

ID: M22
Priority: 10
Dependencies: M01, M03, M05, M09
Database Required: Yes

24.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: waste_collection, waste_conversion, sustainability_metrics, circular_economy_projects
- **API Base Path**: /api/v1/sustainability
- **Authentication Required**: Yes

24.2 Database Schema

```
-- waste_collection table
CREATE TABLE waste_collection (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    gaushala_id UUID REFERENCES users(id) ON DELETE CASCADE,
    collection_date DATE NOT NULL,
    waste_type VARCHAR(50) CHECK (waste_type IN ('COW_DUNG', 'URINE', 'OTH
    quantity_kg DECIMAL(8, 2) NOT NULL,
    collection_method VARCHAR(100),
    storage_location VARCHAR(255),
    quality_notes TEXT,
    collected_by UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- waste_conversion table
CREATE TABLE waste_conversion (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    waste_collection_id UUID REFERENCES waste_collection(id) ON DELETE CASC
    conversion_type VARCHAR(50) CHECK (conversion_type IN ('COMPOST', 'BIOG
    input_quantity_kg DECIMAL(8, 2) NOT NULL,
    output_quantity DECIMAL(8, 2) NOT NULL,
    output_unit VARCHAR(20),
    product_id UUID REFERENCES products(id),
    conversion_efficiency DECIMAL(5, 2),
    conversion_date DATE NOT NULL,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);

-- sustainability_metrics table
CREATE TABLE sustainability_metrics (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    gaushala_id UUID REFERENCES users(id) ON DELETE CASCADE,
    period DATE NOT NULL, -- First day of month
    waste_collected_kg DECIMAL(10, 2) DEFAULT 0,
    waste_converted_kg DECIMAL(10, 2) DEFAULT 0,
    biogas_produced_cubic_meters DECIMAL(10, 2) DEFAULT 0,
    compost_produced_kg DECIMAL(10, 2) DEFAULT 0,
    co2_emissions_saved_kg DECIMAL(10, 2) DEFAULT 0,
    chemical_fertilizer_offset_kg DECIMAL(10, 2) DEFAULT 0,
    water_saved_liters DECIMAL(10, 2) DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(gaushala_id, period)
);

-- circular_economy_projects table
CREATE TABLE circular_economy_projects (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_name VARCHAR(255) NOT NULL,
    gaushala_id UUID REFERENCES users(id) ON DELETE CASCADE,
    project_type VARCHAR(50) CHECK (project_type IN ('BIOGAS_PLANT', 'COMP
    start_date DATE NOT NULL,
    end_date DATE,
    status VARCHAR(20) DEFAULT 'ACTIVE',
    description TEXT,
    investment_amount DECIMAL(12, 2),
    roi_percentage DECIMAL(5, 2),
    environmental_impact JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

24.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a waste-to-value tracking system for circular economy and sustainability metrics.
1. REQUIREMENTS: - Waste collection tracking - Waste conversion monitoring - Sustainability metrics calculation - Circular economy project management - Environmental impact reporting
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /waste/collection - Record waste collection - GET /waste/collection - List waste collections - POST /waste/conversion - Record waste conversion - GET /waste/conversion - List conversions - GET /sustainability/metrics - Get sustainability metrics - POST /sustainability/calculate - Calculate metrics (batch) - GET /sustainability/report - Generate sustainability report - POST /projects - Create circular economy project - GET /projects - List projects - GET /projects/:id - Get project details - PUT /projects/:id - Update project - GET /environmental-impact - Get environmental impact summary - POST /impact/calculate - Calculate environmental impact
4. INTEGRATION: - Import gaushala data from M01 - Import product data from M03 - Import order data from M05 - Use analytics from M09
5. WASTE TYPES: - Cow dung (primary) - Cow urine (for medicines) - Other organic waste - Agricultural residues - Food waste
6. CONVERSION PROCESSES: - Composting (dung to compost) - Biogas production (dung to biogas) - Medicine production (urine based) - Fertilizer production - Other value-added products
7. SUSTAINABILITY METRICS: - Waste diversion rate (from landfill) - Carbon emissions saved - Chemical fertilizer offset - Water saved - Energy generated (from biogas) - Economic value created from waste
8. BUSINESS LOGIC: - Calculate conversion efficiency - Track waste-to-value chain - Monitor circular economy indicators - Calculate environmental impact using standard formulas - Generate sustainability certificates
9. ENVIRONMENTAL IMPACT CALCULATIONS: - CO2 emissions saved from composting vs landfill - Methane capture from biogas - Chemical fertilizer replacement value - Water conservation from organic farming - Soil health improvement metrics
10. CIRCULAR ECONOMY PROJECTS: - Biogas plant installation - Composting facility setup - Urine collection and processing - Organic fertilizer production - Sustainable farming initiatives
11. INTEGRATION POINTS: - Imports: M01, M03, M05, M09 - Exports: calculateSustainability, generateImpactReport functions - Shared: Environmental calculation utilities - Webhooks: Sustainability milestone achievements
12. REPORTING: - Monthly sustainability reports - Environmental impact statements - Circular economy progress tracking - Waste diversion certificates - Carbon credit calculations
Integration Note: Waste conversion creates products in M03. Sustainability metrics appear in M09 dashboard.

25 Module 23: API Gateway

Module Overview

ID: M23
Priority: 10
Dependencies: All modules (central coordination)
Database Required: No (or minimal for routing)

25.1 Technical Specifications

- **Backend**: Node.js/Express + API Gateway middleware
- **Database Tables**: api_routes, rate_limits, api_keys, request_logs
- **API Base Path**: /api (root gateway)
- **Authentication Required**: For protected routes

25.2 Database Schema

```
-- api_routes table
CREATE TABLE api_routes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    service_name VARCHAR(100) NOT NULL,
    route_path VARCHAR(500) NOT NULL,
    target_url VARCHAR(500) NOT NULL,
    http_method VARCHAR(10) CHECK (http_method IN ('GET', 'POST', 'PUT', 'D
    is_protected BOOLEAN DEFAULT TRUE,
    required_permissions JSONB,
    rate_limit_per_minute INTEGER DEFAULT 60,
    timeout_ms INTEGER DEFAULT 30000,
    circuit_breaker_config JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- rate_limits table
CREATE TABLE rate_limits (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    api_key VARCHAR(100),
    user_id UUID REFERENCES users(id),
    endpoint VARCHAR(500) NOT NULL,
    request_count INTEGER DEFAULT 0,
    window_start TIMESTAMP NOT NULL,
    window_end TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
-- api_keys table
CREATE TABLE api_keys (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    key_hash VARCHAR(255) UNIQUE NOT NULL,
    user_id UUID REFERENCES users(id),
    name VARCHAR(255) NOT NULL,
    permissions JSONB,
    rate_limit_per_day INTEGER DEFAULT 1000,
    expires_at TIMESTAMP,
    last_used_at TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- request_logs table
CREATE TABLE request_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    request_id VARCHAR(100) NOT NULL,
    user_id UUID REFERENCES users(id),
    api_key_id UUID REFERENCES api_keys(id),
    endpoint VARCHAR(500) NOT NULL,
    method VARCHAR(10) NOT NULL,
    status_code INTEGER,
    request_body TEXT,
    response_body TEXT,
    response_time_ms INTEGER,
    ip_address INET,
    user_agent TEXT,
    errors JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

25.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create an API Gateway that routes requests to all other modules with centralized authentication, rate limiting, and monitoring.
1. REQUIREMENTS: - Request routing to microservices - Centralized authentication and authorization - Rate limiting and throttling - Request/response logging - Circuit breaker pattern implementation
2. DATABASE: Use exact schema for routing and monitoring
3. API ENDPOINTS: - All routes dynamically loaded from api_routes table - POST /auth/login - Centralized login (proxies to M01) - POST /auth/register - Centralized registration (proxies to M01) - GET /health - Health check endpoint - GET /metrics - API metrics endpoint - POST /apikeys - Generate API key - GET /apikeys - List API keys - DELETE /apikeys/:id - Revoke API key
4. INTEGRATION: - Routes to all other modules (M01-M22) - Import authMiddleware from M01 - Centralized error handling - Request transformation if needed
5. ROUTING CONFIGURATION: - Dynamic route loading from database - Service discovery integration - Load balancing between service instances - Path prefix rewriting - Header manipulation
6. AUTHENTICATION: - JWT validation for protected routes - API key authentication - Permission-based access control - Session management - Token refresh handling
7. RATE LIMITING: - User-based rate limiting - API key-based rate limiting - IP-based rate limiting - Burst allowance - Different limits for different endpoints
8. CIRCUIT BREAKER: - Failure threshold detection - Automatic service fallback - Half-open state testing - Service recovery monitoring - Cascading failure prevention
9. REQUEST/RESPONSE: - Request validation - Response transformation - Error formatting standardization - CORS handling - Compression (gzip)
10. MONITORING: - Request/response logging - Performance metrics collection - Error tracking and alerting - Usage analytics - Service health monitoring
11. SECURITY: - Input sanitization - SQL injection prevention - XSS protection - DDoS mitigation - SSL/TLS enforcement
12. INTEGRATION POINTS: - Routes to: All modules (M01-M22) - Imports: authMiddleware from M01 - Exports: API Gateway instance - Shared: Request validation utilities, error handlers - Webhooks: Service health alerts, rate limit breaches
13. PERFORMANCE: - Response caching - Connection pooling - Request queuing during high load - Efficient routing algorithms - Minimal overhead
14. DEPLOYMENT: - Docker containerization - Kubernetes readiness/liveness probes - Horizontal scaling capability - Blue-green deployment support - Canary release capability
Integration Note: This is the central entry point for all API requests. All other modules must register their routes here.

26 Module 24: Third-party Integration

Module Overview

ID: M24
Priority: 11
Dependencies: M01, M08
Database Required: Yes

26.1 Technical Specifications

- **Backend**: Node.js/Express or Python/FastAPI
- **Database Tables**: third_party_services, integration_configs, webhook_logs, api_credentials
- **API Base Path**: /api/v1/integrations
- **Authentication Required**: Yes (ADMIN for configuration)

26.2 Database Schema

```
-- third_party_services table
CREATE TABLE third_party_services (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    service_name VARCHAR(100) UNIQUE NOT NULL,
    service_type VARCHAR(50) CHECK (service_type IN ('PAYMENT', 'SMS', 'EM
    provider VARCHAR(100) NOT NULL,
    base_url VARCHAR(500),
    documentation_url VARCHAR(500),
    is_active BOOLEAN DEFAULT TRUE,
    supported_features JSONB,
    rate_limits JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- integration_configs table
CREATE TABLE integration_configs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    service_id UUID REFERENCES third_party_services(id) ON DELETE CASCADE,
    environment VARCHAR(20) CHECK (environment IN ('SANDBOX', 'PRODUCTION')
    config_key VARCHAR(100) NOT NULL,
    config_value TEXT,
    is_encrypted BOOLEAN DEFAULT TRUE,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(service_id, environment, config_key)
);
```

```sql
-- webhook_logs table
CREATE TABLE webhook_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    service_id UUID REFERENCES third_party_services(id) ON DELETE CASCADE,
    webhook_type VARCHAR(100),
    payload JSONB NOT NULL,
    headers JSONB,
    status_code INTEGER,
    response_body TEXT,
    processing_time_ms INTEGER,
    error_message TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- api_credentials table
CREATE TABLE api_credentials (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    service_id UUID REFERENCES third_party_services(id) ON DELETE CASCADE,
    credential_type VARCHAR(50) CHECK (credential_type IN ('API_KEY', 'OAUT
    credential_data JSONB NOT NULL,
    expires_at TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    last_used_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

26.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a third-party integration module for managing external service connections and webhooks.
1. REQUIREMENTS: - Third-party service configuration management - API credential management with encryption - Webhook handling and processing - Integration status monitoring - Service fallback and redundancy
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /services - List third-party services - POST /services - Add new service (ADMIN) - PUT /services/:id - Update service (ADMIN) - GET /services/:id/config - Get service configuration - POST /services/:id/config - Update configuration (ADMIN) - GET /credentials - List API credentials - POST /credentials - Add credential (ADMIN) - PUT /credentials/:id - Update credential (ADMIN) - POST /webhooks/:service$_type - Handleincomingwebhooks - GET/webhooks/logs - Viewwebhooklogs - GET/integrations/status - Checkintegrationstatus - POST/services/ : id/test - Testserviceconnection$
4. INTEGRATION: - Centralized configuration for all external services - Provide integration clients to other modules - Use notification service from M08 for alerts - Use authMiddleware from M01
5. SUPPORTED SERVICES: - Payment gateways (Razorpay, Stripe, etc.) - SMS providers (Twilio, Indian providers) - Email services (SendGrid, Amazon SES) - Map services (Google Maps, OSRM) - Logistics APIs (Delhivery, etc.) - Government APIs (DBT, FSSAI, etc.) - Analytics (Google Analytics, Mixpanel)
6. CREDENTIAL MANAGEMENT: - Secure encryption of API keys - OAuth token refresh automation - Credential rotation policies - Access audit logging - Environment-specific credentials (sandbox/production)
7. WEBHOOK HANDLING: - Signature verification - Payload validation - Retry logic for failed processing - Idempotency key handling - Webhook replay protection
8. BUSINESS LOGIC: - Service health monitoring - Automatic failover to backup services - Rate limit tracking and management - Connection pooling for external APIs - Request/response transformation
9. ERROR HANDLING: - Graceful degradation when services fail - Circuit breaker pattern implementation - Alerting for service outages - Automatic retry with exponential backoff - Fallback to alternative services
10. INTEGRATION POINTS: - Imports: M01 (auth), M08 (notifications) - Exports: getServiceClient, handleWebhook functions - Shared: Encryption utilities, API client factories - Webhooks: Service status changes, credential expiry
11. SECURITY: - Encrypt all sensitive credentials - IP whitelisting for webhooks - Request signing for outgoing calls - Regular credential rotation - Audit logging for credential usage
12. MONITORING: - Service response time tracking - Error rate monitoring - Rate limit usage tracking - Webhook processing metrics - Integration health dashboard
Integration Note: All other modules use this module to access external services. Provides centralized logging and monitoring.

27 Module 25: Mobile App Backend

Module Overview

ID: M25
Priority: 11
Dependencies: M01-M24 (mobile-optimized interfaces)
Database Required: Yes (sync and cache tables)

27.1 Technical Specifications

- **Backend**: Node.js/Express with mobile optimizations
- **Database Tables**: mobile_sessions, offline_sync, push_tokens, app_versions
- **API Base Path**: /api/mobile/v1
- **Authentication Required**: Yes (with mobile-specific tokens)

27.2 Database Schema

```
-- mobile_sessions table
CREATE TABLE mobile_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    device_id VARCHAR(255) NOT NULL,
    device_type VARCHAR(50) CHECK (device_type IN ('IOS', 'ANDROID', 'OTHER
    device_model VARCHAR(100),
    os_version VARCHAR(50),
    app_version VARCHAR(50),
    fcm_token VARCHAR(255),
    last_active_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, device_id)
);

-- offline_sync table
CREATE TABLE offline_sync (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    entity_type VARCHAR(50) NOT NULL,
    entity_id UUID NOT NULL,
    operation VARCHAR(10) CHECK (operation IN ('CREATE', 'UPDATE', 'DELETE'
    local_data JSONB NOT NULL,
    sync_status VARCHAR(20) DEFAULT 'PENDING',
    retry_count INTEGER DEFAULT 0,
    last_attempt_at TIMESTAMP,
    error_message TEXT,
```

```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    synced_at TIMESTAMP
);

-- push_tokens table
CREATE TABLE push_tokens (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    token VARCHAR(255) UNIQUE NOT NULL,
    platform VARCHAR(20) CHECK (platform IN ('FCM', 'APNS')),
    is_active BOOLEAN DEFAULT TRUE,
    last_used_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- app_versions table
CREATE TABLE app_versions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    platform VARCHAR(20) CHECK (platform IN ('IOS', 'ANDROID')) NOT NULL,
    version_code VARCHAR(50) NOT NULL,
    version_name VARCHAR(50),
    is_mandatory BOOLEAN DEFAULT FALSE,
    release_notes TEXT,
    download_url VARCHAR(500),
    min_os_version VARCHAR(50),
    released_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(platform, version_code)
);
```

27.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a mobile-optimized backend with offline sync, push notifications, and mobile-specific features.
1. REQUIREMENTS: - Mobile-optimized API endpoints - Offline data sync capability - Push notification handling - Device registration and management - App version control and updates
2. DATABASE: Use exact schema for mobile-specific data
3. API ENDPOINTS: - POST /mobile/register-device - Register mobile device - POST /mobile/push-token - Register push token - GET /mobile/sync - Get sync data for offline - POST /mobile/sync - Submit offline changes - GET /mobile/notifications - Get mobile notifications - POST /mobile/notifications/read - Mark notifications read - GET /mobile/app/version - Check app version - GET /mobile/config - Get mobile configuration - POST /mobile/feedback - Submit app feedback - GET /mobile/status - Get mobile service status
4. INTEGRATION: - Mobile-optimized versions of core endpoints - Integrate with M08 for push notifications - Use M01 for authentication with mobile tokens - Provide mobile-specific data formats
5. OFFLINE SYNC: - Conflict resolution strategies - Last-write-wins or manual resolution - Sync queue management - Retry logic for failed syncs - Data compression for sync payloads
6. PUSH NOTIFICATIONS: - FCM (Firebase Cloud Messaging) for Android - APNS (Apple Push Notification Service) for iOS - Silent push notifications for data sync - Rich notifications with actions - Notification grouping and categorization
7. MOBILE OPTIMIZATIONS: - Reduced payload sizes - Efficient pagination - Image compression and caching - Battery-efficient polling - Background sync scheduling
8. BUSINESS LOGIC: - Handle intermittent connectivity - Manage local storage constraints - Optimize for mobile data usage - Support different screen sizes - Handle app background/foreground transitions
9. SECURITY: - Mobile-specific authentication tokens - Device fingerprinting for security - Secure local storage encryption - Certificate pinning for API calls - Jailbreak/root detection
10. INTEGRATION POINTS: - Imports: M01 (auth), M08 (notifications), all other modules via mobile wrappers - Exports: mobileAuthMiddleware, syncHandler functions - Shared: Mobile optimization utilities, push notification handlers - Webhooks: App version updates, device registration events
11. PERFORMANCE: - Response compression - Connection keep-alive optimization - CDN for static assets - Database query optimization for mobile - Cache headers for API responses
12. ANALYTICS: - App usage statistics - Crash reporting integration - Performance monitoring - User engagement metrics - Feature usage tracking
Integration Note: This module wraps core functionality from other modules with mobile optimizations. Sync conflicts must be resolved with source modules.

28 Module 26: Admin Dashboard

Module Overview

ID: M26
Priority: 12
Dependencies: M01-M25 (read access to all)
Database Required: Yes (admin-specific tables)

28.1 Technical Specifications

- **Backend**: Node.js/Express with admin features
- **Database Tables**: admin_users, admin_logs, admin_tasks, system_config
- **API Base Path**: /api/admin/v1
- **Authentication Required**: Yes (ADMIN role only)

28.2 Database Schema

```
-- admin_users table
CREATE TABLE admin_users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID UNIQUE REFERENCES users(id) ON DELETE CASCADE,
    admin_role VARCHAR(50) CHECK (admin_role IN ('SUPER_ADMIN', 'SUPPORT',
    permissions JSONB NOT NULL,
    last_login_at TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    created_by UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- admin_logs table
CREATE TABLE admin_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    admin_id UUID REFERENCES admin_users(id),
    action VARCHAR(100) NOT NULL,
    entity_type VARCHAR(50),
    entity_id UUID,
    changes JSONB,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- admin_tasks table
CREATE TABLE admin_tasks (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    task_type VARCHAR(100) NOT NULL,
    assigned_to UUID REFERENCES admin_users(id),
    assigned_by UUID REFERENCES admin_users(id),
    priority VARCHAR(20) DEFAULT 'MEDIUM',
    status VARCHAR(20) DEFAULT 'PENDING',
    due_date DATE,
    completed_at TIMESTAMP,
    notes TEXT,
    metadata JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- system_config table
CREATE TABLE system_config (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    config_key VARCHAR(100) UNIQUE NOT NULL,
    config_value JSONB NOT NULL,
    data_type VARCHAR(20) CHECK (data_type IN ('STRING', 'NUMBER', 'BOOLEAN
    category VARCHAR(50),
    description TEXT,
    is_public BOOLEAN DEFAULT FALSE,
    updated_by UUID REFERENCES admin_users(id),
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

28.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a comprehensive admin dashboard backend for platform management and monitoring.
1. REQUIREMENTS: - Admin user management with role-based permissions - System-wide configuration management - Audit logging for all admin actions - Admin task assignment and tracking - Platform analytics and monitoring
2. DATABASE: Use exact schema for admin-specific data
3. API ENDPOINTS: - GET /admin/users - List admin users - POST /admin/users - Create admin user (SUPER$_ADMIN only$) – $PUT/admin/users/ : id – Updateadminuser – GET/admin/logs – Viewadminactionlogs – GET/admin/tasks – Listadmintasks – POST/admin/tasks – Createadmintask – PUT/admin/tasks/ : id – Updatetaskstatus – GET/admin/config – Getsystemconfiguration – PUT/admin/config – Updateconfiguration – GET/admin/analytics/platform – Platformanalytics – GET/admin/analytics/users – Useranalytics – GET/admin/analytics/transactions – Transactionanalytics – GET/admin/health – Systemhealthcheck – POST/admin/maintenance – Togglemaintenancemode$
4. INTEGRATION: - Read access to all other modules - Write access for moderation and management - Import authMiddleware from M01 with ADMIN check - Use notification service from M08 for admin alerts
5. ADMIN ROLES: - SUPER$_ADMIN : Fullsystemaccess – SUPPORT : Usersupportandissueresolution – MODERATOR : Contentmoderationandreviews – ANALYST : Analyticsandreportingaccess – FINANCE : Financialdataaccessonly$
6. PERMISSION SYSTEM: - Fine-grained permission control - Module-level permissions - Action-level permissions (read, write, delete) - Permission inheritance from roles - Temporary permission grants
7. AUDIT LOGGING: - Log all admin actions - Track data changes (before/after) - IP and user agent tracking - Searchable audit logs - Exportable log data
8. BUSINESS LOGIC: - Admin approval workflows - Escalation procedures - Scheduled admin reports - Automated alerting for issues - Performance monitoring thresholds
9. SYSTEM CONFIGURATION: - Platform-wide settings - Feature flags management - Email template configuration - Rate limit configuration - Maintenance mode control
10. INTEGRATION POINTS: - Imports: M01 (auth with ADMIN check), M08 (notifications), all other modules (read access) - Exports: adminAuthMiddleware, auditLogger functions - Shared: Permission validation utilities, admin task scheduler - Webhooks: Admin alert notifications, system health warnings
11. SECURITY: - Two-factor authentication for admin access - IP whitelisting for admin panel - Session timeout for admin sessions - Login attempt limiting - Regular security audit logging
12. MONITORING: - Real-time platform metrics - Error rate monitoring - Performance bottleneck detection - User complaint tracking - System resource monitoring
Integration Note: Admin actions that modify data in other modules must use those modules' APIs, not direct database access.

29 Module 27: Reporting Engine

Module Overview

ID: M27
Priority: 12
Dependencies: M01-M26 (data sources)
Database Required: Yes (report definitions and cache)

29.1 Technical Specifications

- **Backend**: Node.js/Express with reporting features
- **Database Tables**: report_definitions, scheduled_reports, report_exports, data_sources
- **API Base Path**: /api/v1/reports
- **Authentication Required**: Yes (with report permissions)

29.2 Database Schema

```
-- report_definitions table
CREATE TABLE report_definitions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    report_code VARCHAR(100) UNIQUE NOT NULL,
    report_name VARCHAR(255) NOT NULL,
    description TEXT,
    report_type VARCHAR(50) CHECK (report_type IN ('ANALYTICAL', 'OPERATION
    data_source_id UUID REFERENCES data_sources(id),
    query_template TEXT NOT NULL,
    parameters JSONB,
    columns JSONB NOT NULL,
    filters JSONB,
    sorting JSONB,
    grouping JSONB,
    chart_config JSONB,
    access_roles JSONB,
    is_public BOOLEAN DEFAULT FALSE,
    refresh_interval INTEGER, -- in minutes
    cache_duration INTEGER DEFAULT 300, -- in seconds
    created_by UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- scheduled_reports table
CREATE TABLE scheduled_reports (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    report_id UUID REFERENCES report_definitions(id) ON DELETE CASCADE,
```

```
    schedule_type VARCHAR(20) CHECK (schedule_type IN ('DAILY', 'WEEKLY',
    schedule_config JSONB,
    recipients JSONB,
    format VARCHAR(20) DEFAULT 'PDF',
    last_run_at TIMESTAMP,
    next_run_at TIMESTAMP,
    status VARCHAR(20) DEFAULT 'ACTIVE',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- report_exports table
CREATE TABLE report_exports (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    report_id UUID REFERENCES report_definitions(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id),
    parameters JSONB,
    format VARCHAR(20) NOT NULL,
    file_url VARCHAR(500),
    file_size INTEGER,
    status VARCHAR(20) DEFAULT 'PROCESSING',
    generated_at TIMESTAMP,
    expires_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- data_sources table
CREATE TABLE data_sources (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    source_name VARCHAR(100) UNIQUE NOT NULL,
    source_type VARCHAR(50) CHECK (source_type IN ('DATABASE', 'API', 'FILE
    connection_config JSONB,
    is_encrypted BOOLEAN DEFAULT TRUE,
    refresh_schedule VARCHAR(50),
    last_refreshed_at TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

29.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a powerful reporting engine with scheduled reports, custom report builder, and multiple export formats.
1. REQUIREMENTS: - Custom report definition and building - Scheduled report generation and distribution - Multiple export formats (PDF, Excel, CSV, JSON) - Report caching and performance optimization - Data source integration for reports
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - GET /reports - List available reports - GET /reports/:id - Get report definition - POST /reports - Create report (ADMIN/authorized) - PUT /reports/:id - Update report - POST /reports/:id/run - Run report with parameters - GET /reports/:id/data - Get report data (cached) - POST /reports/:id/export - Export report - GET /exports/:id - Get export status - POST /schedules - Schedule report - GET /schedules - List scheduled reports - PUT /schedules/:id - Update schedule - GET /data-sources - List data sources - POST /data-sources - Add data source (ADMIN) - GET /reports/builder/fields - Get available fields for builder
4. INTEGRATION: - Access data from all other modules - Use authMiddleware from M01 with report permissions - Integrate with M08 for report delivery notifications - Use M09 analytics data for reports
5. REPORT TYPES: - Sales reports (by product, gaushala, region) - Inventory reports (stock levels, turnover) - Financial reports (revenue, payments, commissions) - User activity reports (engagement, growth) - Logistics reports (shipments, delivery performance) - Quality reports (certifications, test results)
6. EXPORT FORMATS: - PDF (with charts and formatting) - Excel (with multiple sheets) - CSV (for data processing) - JSON (for API consumption) - HTML (for web display)
7. SCHEDULING: - Time-based scheduling (daily, weekly, monthly) - Event-based triggering (on data update) - Conditional execution (only if data changed) - Multi-recipient distribution - Failure retry and notification
8. BUSINESS LOGIC: - Parameter validation for reports - Data aggregation and calculation - Chart generation from report data - Report template management - Caching strategy for frequently accessed reports
9. PERFORMANCE: - Query optimization for large datasets - Incremental data loading - Report result caching - Parallel report generation - Memory management for large exports
10. INTEGRATION POINTS: - Imports: M01 (auth), M08 (notifications), all other modules (data sources) - Exports: runReport, scheduleReport functions - Shared: Report template utilities, export format handlers - Webhooks: Report generation completion, schedule triggers
11. SECURITY: - Row-level security for report data - Data masking for sensitive information - Report access audit logging - Export file encryption - Secure file storage and sharing
12. CUSTOM REPORT BUILDER: - Drag-and-drop field selection - Filter and sort configuration - Chart type selection - Parameter definition - Preview before saving
Integration Note: Reports can query any module's data. Ensure data consistency and handle module dependencies in queries.

30 Module 28: System Monitoring

Module Overview

ID: M28
Priority: 12
Dependencies: All modules (monitoring integration)
Database Required: Yes (monitoring data)

30.1 Technical Specifications

- **Backend**: Node.js/Express with monitoring features
- **Database Tables**: system_metrics, error_logs, performance_logs, alert_rules, incident_reports
- **API Base Path**: /api/v1/monitoring
- **Authentication Required**: Yes (ADMIN for most endpoints)

30.2 Database Schema

```
-- system_metrics table
CREATE TABLE system_metrics (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    metric_name VARCHAR(100) NOT NULL,
    metric_value DECIMAL(12, 4) NOT NULL,
    metric_unit VARCHAR(50),
    tags JSONB,
    timestamp TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- error_logs table
CREATE TABLE error_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    error_code VARCHAR(100),
    error_message TEXT NOT NULL,
    stack_trace TEXT,
    module_name VARCHAR(100),
    endpoint VARCHAR(500),
    user_id UUID REFERENCES users(id),
    request_id VARCHAR(100),
    severity VARCHAR(20) CHECK (severity IN ('LOW', 'MEDIUM', 'HIGH', 'CRIT
    environment VARCHAR(20) DEFAULT 'PRODUCTION',
    resolved BOOLEAN DEFAULT FALSE,
    resolved_by UUID REFERENCES users(id),
    resolved_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);

-- performance_logs table
CREATE TABLE performance_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    endpoint VARCHAR(500) NOT NULL,
    method VARCHAR(10) NOT NULL,
    response_time_ms INTEGER NOT NULL,
    status_code INTEGER,
    request_size_bytes INTEGER,
    response_size_bytes INTEGER,
    user_id UUID REFERENCES users(id),
    timestamp TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- alert_rules table
CREATE TABLE alert_rules (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    rule_name VARCHAR(100) NOT NULL,
    metric_name VARCHAR(100) NOT NULL,
    condition VARCHAR(20) CHECK (condition IN ('GREATER_THAN', 'LESS_THAN'
    threshold_value DECIMAL(12, 4),
    time_window INTEGER, -- in seconds
    severity VARCHAR(20) DEFAULT 'MEDIUM',
    notification_channels JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    cooldown_period INTEGER DEFAULT 300, -- in seconds
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- incident_reports table
CREATE TABLE incident_reports (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    incident_id VARCHAR(100) UNIQUE NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    severity VARCHAR(20) DEFAULT 'MEDIUM',
    status VARCHAR(20) DEFAULT 'OPEN',
    affected_services JSONB,
    started_at TIMESTAMP NOT NULL,
    resolved_at TIMESTAMP,
    root_cause TEXT,
    resolution_notes TEXT,
    created_by UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);
```

30.3 AI Development Prompt

ChatGPT/Gemini/Claude Prompt:
Create a comprehensive system monitoring module with metrics collection, alerting, and incident management.
1. REQUIREMENTS: - System metrics collection and visualization - Error logging and tracking - Performance monitoring - Alert rules and notification - Incident management and reporting
2. DATABASE: Use exact schema provided
3. API ENDPOINTS: - POST /metrics - Submit metric data - GET /metrics - Query metrics with filters - POST /errors - Log error - GET /errors - List errors with filters - PUT /errors/:id/resolve - Mark error as resolved - GET /performance - Get performance data - GET /alerts - List alert rules - POST /alerts - Create alert rule (ADMIN) - PUT /alerts/:id - Update alert rule (ADMIN) - GET /alerts/active - Get active alerts - POST /incidents - Create incident report - GET /incidents - List incidents - PUT /incidents/:id - Update incident - GET /incidents/:id/timeline - Get incident timeline - GET /monitoring/health - Overall system health - GET /monitoring/summary - Monitoring summary dashboard
4. INTEGRATION: - Collect metrics from all modules - Integrate with M08 for alert notifications - Use M26 admin module for incident management - Provide monitoring APIs for all modules to log data
5. METRICS COLLECTION: - System resources (CPU, memory, disk) - Application performance (response times, error rates) - Business metrics (orders, users, transactions) - Database performance (query times, connections) - External service health (APIs, third-party services)
6. ALERTING SYSTEM: - Threshold-based alerts - Anomaly detection alerts - Composite alerts (multiple conditions) - Escalation policies - Alert grouping and deduplication
7. PERFORMANCE MONITORING: - Endpoint response time tracking - Database query performance - Cache hit rates - External API latency - Queue processing times
8. BUSINESS LOGIC: - Calculate system health scores - Identify performance bottlenecks - Correlate errors with system events - Generate incident timelines - Calculate service level indicators (SLIs)
9. INCIDENT MANAGEMENT: - Incident creation and tracking - Communication during incidents - Post-incident analysis (postmortems) - Action item tracking - Incident trend analysis
10. INTEGRATION POINTS: - Imports: M08 (notifications), M26 (admin), all modules (metric sources) - Exports: logError, recordMetric, createIncident functions - Shared: Monitoring utilities, alert engine - Webhooks: Alert triggers, incident updates
11. VISUALIZATION: - Real-time metrics dashboards - Historical trend charts - Error frequency graphs - Performance heatmaps - Service dependency maps
12. RETENTION AND ARCHIVING: - Configurable data retention periods - Automated data archiving - Summary data aggregation - Compliance logging - Audit trail maintenance
Integration Note: All modules should use this module for error logging and metric collection. Incident management coordinates with admin team via M26.

31 Integration Guidelines

31.1 Module Integration Sequence

1. **Phase 1 (Core)**: M01 → M02 → M03 → M04 → M05 → M06
2. **Phase 2 (Essential)**: M07 → M08 → M09 → M10 → M11 → M12
3. **Phase 3 (Advanced)**: M13 → M14 → M15 → M16 → M17 → M18
4. **Phase 4 (Specialized)**: M19 → M20 → M21 → M22
5. **Phase 5 (Infrastructure)**: M23 → M24 → M25 → M26 → M27 → M28

31.2 Integration Testing Checklist

- **Data Consistency**: Foreign keys match across modules
- **API Contracts**: Request/response formats standardized
- **Error Handling**: Consistent error codes and messages
- **Authentication**: JWT tokens work across all modules
- **Database Transactions**: Cross-module transactions handled properly
- **Eventual Consistency**: Async operations sync correctly
- **Performance**: No circular dependencies causing slowdowns
- **Security**: Data access controls enforced consistently

31.3 Shared Contracts

- **User ID**: UUID format, consistent across all modules
- **Error Format**: {s̈uccess̈: boolean, ërror̈: {c̈odë: string, m̈essagë: string, d̈etails̈: object}}
- **Date Format**: ISO 8601 (YYYY-MM-DDTHH:mm:ss.sssZ)
- **Currency**: INR, 2 decimal places
- **Measurement Units**: kg, liters, meters standardized
- **Pagination**: limit/offset or cursor-based consistently

32 Development Timeline

Phase	Weeks	Modules
Foundation	1-4	M01, M02, M03, M04, M05, M06
Core Platform	5-8	M07, M08, M09, M10, M11, M12
Advanced Features	9-12	M13, M14, M15, M16, M17, M18
Specialized Modules	13-16	M19, M20, M21, M22
Infrastructure	17-20	M23, M24, M25, M26, M27, M28
Integration & Testing	21-24	All modules integration

33 Conclusion

This modular architecture allows parallel development using AI tools while ensuring seamless integration through standardized contracts and interfaces. Each module is self-contained with clear dependencies, making it ideal for distributed development teams.