Restaurant CRM Platform - Server-Side System Design

1. System Architecture Overview

1.1 Architecture Pattern

- **Microservices Architecture** with event-driven communication
- **API Gateway** for request routing and cross-cutting concerns
- **Event-driven messaging** for asynchronous communication
- **Database per service** pattern for data isolation
- **CQRS (Command Query Responsibility Segregation)** for complex read/write operations

1.2 High-Level Architecture Diagram

```
┌─────────────────┐  ┌─────────────────┐
┌─────────────────┐
│  Mobile Apps  │  │ Web Dashboard │  │  Admin Panel  │
└─────────────────┘  └─────────────────┘
└─────────────────┘
    │         │         │
    └─────────────────────┼─────────────────────┘
              │
        ┌─────────────────┐
        │  API Gateway  │
        │  (Kong/Nginx) │
        └─────────────────┘
              │
    ┌─────────────────────┼─────────────────────┐
    │         │         │
┌─────────────┐   ┌─────────────┐   ┌─────────────┐
│  Auth     │   │ User     │   │  Org     │
│  Service   │   │ Management │   │ Management │
└─────────────┘   └─────────────┘   └─────────────┘
    │         │         │
    └─────────────────────┼─────────────────────┘
              │
        ┌─────────────────┐
        │  Event Bus   │
        │ (RabbitMQ)   │
        └─────────────────┘
              │
```

```
┌──────────────────────────┼──────────────────────────
─┐
  │          │          │
┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐
┌────────┐
│Procure │ │ Stock  │ │Process │ │Kitchen │ │Customer│
│ment    │ │Control │ │ing     │ │        │ │Service │
│Service │ │Service │ │Service │ │Service │ │Service │
└────────┘ └────────┘ └────────┘ └────────┘
└────────┘
  │     │     │     │     │
┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐
┌────────┐
│Account │ │Analytics│ │Reporting│ │Notifica│ │Payment │
│ing     │ │& AI    │ │Engine  │ │tion    │ │Service │
│Service │ │Service │ │        │ │Service │ │        │
└────────┘ └────────┘ └────────┘ └────────┘
└────────┘
```

2. Core Services Architecture

2.1 API Gateway Layer

Technology: Kong or Nginx with custom modules **Responsibilities**:

- Request routing to appropriate microservices
- Authentication and authorization enforcement
- Rate limiting and throttling
- Request/response transformation
- Load balancing
- SSL termination
- API versioning
- Monitoring and logging

Configuration:

```
services:
  - name: auth-service
    url: http://auth-service:3001
    routes:
      - paths: ["/api/v1/auth"]
  - name: procurement-service
    url: http://procurement-service:3002
    routes:
      - paths: ["/api/v1/procurement"]
    plugins:
      - name: jwt
      - name: rate-limiting
```

2.2 Authentication & Authorization Service

Technology: NestJS + Passport.js + JWT **Database**: PostgreSQL **Responsibilities**:

- User authentication (login/logout)
- JWT token generation and validation
- Multi-factor authentication
- Role-based access control (RBAC)
- Permission management
- Session management
- OAuth 2.0 integration

API Endpoints:

```
POST   /api/v1/auth/login
POST   /api/v1/auth/logout
POST   /api/v1/auth/refresh
POST   /api/v1/auth/forgot-password
POST   /api/v1/auth/reset-password
GET    /api/v1/auth/profile
PUT    /api/v1/auth/profile
POST   /api/v1/auth/verify-mfa
```

Database Schema:

```
-- Users table
CREATE TABLE users (
    id UUID PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    phone VARCHAR(20),
    is_active BOOLEAN DEFAULT true,
    is_verified BOOLEAN DEFAULT false,
    mfa_enabled BOOLEAN DEFAULT false,
    mfa_secret VARCHAR(255),
    last_login TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Roles table
```

```
CREATE TABLE roles (
   id UUID PRIMARY KEY,
   name VARCHAR(50) UNIQUE NOT NULL,
   description TEXT,
   permissions JSONB,
   created_at TIMESTAMP DEFAULT NOW()
);

-- User roles junction table
CREATE TABLE user_roles (
   user_id UUID REFERENCES users(id),
   role_id UUID REFERENCES roles(id),
   organization_id UUID,
   PRIMARY KEY (user_id, role_id, organization_id)
);
```

2.3 Organization Management Service

Technology: NestJS + TypeScript **Database**: PostgreSQL **Responsibilities**:

- Restaurant organization management
- Multi-tenant data isolation
- Subscription management
- Organization settings and configuration
- Location management

API Endpoints:

```
POST   /api/v1/organizations
GET    /api/v1/organizations/:id
PUT    /api/v1/organizations/:id
DELETE /api/v1/organizations/:id
GET    /api/v1/organizations/:id/users
POST   /api/v1/organizations/:id/users
GET    /api/v1/organizations/:id/settings
PUT    /api/v1/organizations/:id/settings
```

Database Schema:

```
-- Organizations table
CREATE TABLE organizations (
   id UUID PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   slug VARCHAR(100) UNIQUE NOT NULL,
   description TEXT,
   address JSONB,
   phone VARCHAR(20),
   email VARCHAR(255),
   website VARCHAR(255),
   timezone VARCHAR(50) DEFAULT 'UTC',
   currency VARCHAR(3) DEFAULT 'USD',
   subscription_plan VARCHAR(50),
   subscription_status VARCHAR(20),
   settings JSONB DEFAULT '{}',
   is_active BOOLEAN DEFAULT true,
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Organization locations
CREATE TABLE organization_locations (
   id UUID PRIMARY KEY,
   organization_id UUID REFERENCES organizations(id),
   name VARCHAR(255) NOT NULL,
   address JSONB,
   phone VARCHAR(20),
   is_primary BOOLEAN DEFAULT false,
   settings JSONB DEFAULT '{}',
   created_at TIMESTAMP DEFAULT NOW()
);
```

2.4 Procurement Service

Technology: NestJS + TypeScript **Database**: PostgreSQL **Cache**: Redis **Responsibilities**:

- Purchase order management
- Supplier management
- Procurement workflow
- Budget tracking
- Cost analysis

API Endpoints:

```
// Purchase Orders
POST   /api/v1/procurement/purchase-orders
GET    /api/v1/procurement/purchase-orders
GET    /api/v1/procurement/purchase-orders/:id
PUT    /api/v1/procurement/purchase-orders/:id
DELETE /api/v1/procurement/purchase-orders/:id
POST   /api/v1/procurement/purchase-orders/:id/submit
POST   /api/v1/procurement/purchase-orders/:id/approve
POST   /api/v1/procurement/purchase-orders/:id/execute

// Suppliers
```

```
POST   /api/v1/procurement/suppliers
GET    /api/v1/procurement/suppliers
GET    /api/v1/procurement/suppliers/:id
PUT    /api/v1/procurement/suppliers/:id
DELETE /api/v1/procurement/suppliers/:id

// Transfer to Stock Control
POST   /api/v1/procurement/transfers
GET    /api/v1/procurement/transfers
PUT    /api/v1/procurement/transfers/:id/status
```

Database Schema:

```
-- Purchase orders table
CREATE TABLE purchase_orders (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   po_number VARCHAR(50) UNIQUE NOT NULL,
   supplier_id UUID REFERENCES suppliers(id),
   status VARCHAR(20) DEFAULT 'draft',
   total_amount DECIMAL(12,2),
   currency VARCHAR(3) DEFAULT 'USD',
   notes TEXT,
   requested_by UUID REFERENCES users(id),
   approved_by UUID REFERENCES users(id),
   approved_at TIMESTAMP,
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Purchase order items
CREATE TABLE purchase_order_items (
   id UUID PRIMARY KEY,
   purchase_order_id UUID REFERENCES purchase_orders(id),
   item_name VARCHAR(255) NOT NULL,
   item_sku VARCHAR(100),
   quantity DECIMAL(10,3) NOT NULL,
   unit_price DECIMAL(10,2) NOT NULL,
   total_price DECIMAL(12,2) NOT NULL,
   specifications JSONB,
   created_at TIMESTAMP DEFAULT NOW()
);

-- Suppliers table
CREATE TABLE suppliers (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   name VARCHAR(255) NOT NULL,
   contact_person VARCHAR(255),
   email VARCHAR(255),
   phone VARCHAR(20),
   address JSONB,
   payment_terms VARCHAR(100),
   rating DECIMAL(3,2),
   is_active BOOLEAN DEFAULT true,
   created_at TIMESTAMP DEFAULT NOW()
);
```

2.5 Stock Control Service

Technology: NestJS + TypeScript **Database**: PostgreSQL **Cache**: Redis **Responsibilities**:

- Inventory management
- Stock validation
- Transfer management
- Stock audits
- FIFO tracking

API Endpoints:

```
// Inventory Management
GET    /api/v1/stock/inventory
POST   /api/v1/stock/inventory
GET    /api/v1/stock/inventory/:id
PUT    /api/v1/stock/inventory/:id
DELETE /api/v1/stock/inventory/:id

// Stock Transfers
GET    /api/v1/stock/transfers/incoming
POST   /api/v1/stock/transfers/:id/validate
POST   /api/v1/stock/transfers/outgoing
GET    /api/v1/stock/transfers
PUT    /api/v1/stock/transfers/:id/status

// Stock Requests
POST   /api/v1/stock/requests
GET    /api/v1/stock/requests
PUT    /api/v1/stock/requests/:id/status

// Stock Audits
POST   /api/v1/stock/audits
GET    /api/v1/stock/audits
GET    /api/v1/stock/reports/valuation
GET    /api/v1/stock/reports/usage
```

Database Schema:

```
-- Inventory items table
CREATE TABLE inventory_items (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   item_name VARCHAR(255) NOT NULL,
   sku VARCHAR(100) UNIQUE NOT NULL,
   category VARCHAR(100),
   unit_of_measure VARCHAR(20),
   current_quantity DECIMAL(10,3) DEFAULT 0,
   reserved_quantity DECIMAL(10,3) DEFAULT 0,
   minimum_stock DECIMAL(10,3) DEFAULT 0,
   maximum_stock DECIMAL(10,3),
   unit_cost DECIMAL(10,2),
   storage_location VARCHAR(255),
   expiry_date DATE,
   batch_number VARCHAR(100),
   is_active BOOLEAN DEFAULT true,
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Stock movements table
CREATE TABLE stock_movements (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   inventory_item_id UUID REFERENCES inventory_items(id),
   movement_type VARCHAR(20) NOT NULL, -- 'in', 'out', 'adjustment'
   quantity DECIMAL(10,3) NOT NULL,
   unit_cost DECIMAL(10,2),
   reference_type VARCHAR(50), -- 'purchase_order', 'transfer', 'adjustment'
   reference_id UUID,
   notes TEXT,
   created_by UUID REFERENCES users(id),
   created_at TIMESTAMP DEFAULT NOW()
);

-- Stock transfers table
CREATE TABLE stock_transfers (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   from_department VARCHAR(50),
   to_department VARCHAR(50),
   status VARCHAR(20) DEFAULT 'pending',
   transfer_type VARCHAR(30), -- 'procurement_to_stock', 'stock_to_processing'
```

```
    notes TEXT,
    requested_by UUID REFERENCES users(id),
    validated_by UUID REFERENCES users(id),
    validated_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Stock transfer items
CREATE TABLE stock_transfer_items (
    id UUID PRIMARY KEY,
    transfer_id UUID REFERENCES stock_transfers(id),
    inventory_item_id UUID REFERENCES inventory_items(id),
    requested_quantity DECIMAL(10,3) NOT NULL,
    validated_quantity DECIMAL(10,3),
    unit_cost DECIMAL(10,2),
    notes TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);
```

2.6 Processing Service

Technology: NestJS + TypeScript **Database**: PostgreSQL **AI Integration**: TensorFlow/OpenAI API **Responsibilities**:

- Recipe management
- AI-powered yield calculation
- Production planning
- Kitchen coordination
- Waste optimization

API Endpoints:

```
// Recipe Management
POST   /api/v1/processing/recipes
GET    /api/v1/processing/recipes
GET    /api/v1/processing/recipes/:id
PUT    /api/v1/processing/recipes/:id
DELETE /api/v1/processing/recipes/:id

// AI Yield Calculation
POST   /api/v1/processing/calculate-yield
POST   /api/v1/processing/optimize-recipe

// Production Planning
POST   /api/v1/processing/production-plans
GET    /api/v1/processing/production-plans
```

```
PUT    /api/v1/processing/production-plans/:id/status

// Stock Requests
POST   /api/v1/processing/stock-requests
GET    /api/v1/processing/stock-requests

// Kitchen Transfers
POST   /api/v1/processing/kitchen-transfers
GET    /api/v1/processing/kitchen-transfers
PUT    /api/v1/processing/kitchen-transfers/:id/status
```

Database Schema:

```
-- Recipes table
CREATE TABLE recipes (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   name VARCHAR(255) NOT NULL,
   description TEXT,
   category VARCHAR(100),
   preparation_time INTEGER, -- in minutes
   cooking_time INTEGER,
   serving_size DECIMAL(10,2),
   difficulty_level VARCHAR(20),
   instructions TEXT,
   nutritional_info JSONB,
   cost_per_serving DECIMAL(10,2),
   is_active BOOLEAN DEFAULT true,
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Recipe ingredients
CREATE TABLE recipe_ingredients (
   id UUID PRIMARY KEY,
   recipe_id UUID REFERENCES recipes(id),
   inventory_item_id UUID REFERENCES inventory_items(id),
   quantity DECIMAL(10,3) NOT NULL,
   unit_of_measure VARCHAR(20),
   preparation_notes TEXT,
   created_at TIMESTAMP DEFAULT NOW()
);

-- Production plans
CREATE TABLE production_plans (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   plan_date DATE NOT NULL,
   status VARCHAR(20) DEFAULT 'planned',
   total_recipes INTEGER DEFAULT 0,
   estimated_cost DECIMAL(12,2),
   actual_cost DECIMAL(12,2),
   notes TEXT,
   created_by UUID REFERENCES users(id),
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Production plan items
CREATE TABLE production_plan_items (
   id UUID PRIMARY KEY,
   production_plan_id UUID REFERENCES production_plans(id),
   recipe_id UUID REFERENCES recipes(id),
   planned_quantity INTEGER NOT NULL,
   actual_quantity INTEGER,
   estimated_cost DECIMAL(10,2),
   actual_cost DECIMAL(10,2),
   status VARCHAR(20) DEFAULT 'pending',
   started_at TIMESTAMP,
   completed_at TIMESTAMP
);
```

2.7 Kitchen Service

Technology: NestJS + TypeScript **Database**: PostgreSQL **WebSocket**: Socket.io
Responsibilities:

- Order management
- Real-time kitchen tracking
- 3D visualization integration
- Equipment monitoring
- Customer service coordination

API Endpoints:

```
// Order Management
GET    /api/v1/kitchen/orders
GET    /api/v1/kitchen/orders/:id
PUT    /api/v1/kitchen/orders/:id/status
POST   /api/v1/kitchen/orders/:id/start
POST   /api/v1/kitchen/orders/:id/complete

// Real-time Status
GET    /api/v1/kitchen/status
```

```
PUT    /api/v1/kitchen/stations/:id/status
GET    /api/v1/kitchen/equipment
PUT    /api/v1/kitchen/equipment/:id/status

// Processing Integration
GET    /api/v1/kitchen/incoming-transfers
POST   /api/v1/kitchen/transfers/:id/validate

// 3D Visualization
GET    /api/v1/kitchen/visualization/data
POST   /api/v1/kitchen/visualization/update
```

Database Schema:

```
-- Kitchen orders table
CREATE TABLE kitchen_orders (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   order_number VARCHAR(50) NOT NULL,
   customer_service_order_id UUID,
   status VARCHAR(20) DEFAULT 'received',
   priority INTEGER DEFAULT 1,
   estimated_prep_time INTEGER,
   actual_prep_time INTEGER,
   assigned_station VARCHAR(100),
   assigned_chef UUID REFERENCES users(id),
   special_instructions TEXT,
   started_at TIMESTAMP,
   completed_at TIMESTAMP,
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Kitchen order items
CREATE TABLE kitchen_order_items (
   id UUID PRIMARY KEY,
   kitchen_order_id UUID REFERENCES kitchen_orders(id),
   recipe_id UUID REFERENCES recipes(id),
   quantity INTEGER NOT NULL,
   status VARCHAR(20) DEFAULT 'pending',
   special_instructions TEXT,
   started_at TIMESTAMP,
   completed_at TIMESTAMP
);

-- Kitchen stations
CREATE TABLE kitchen_stations (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   name VARCHAR(100) NOT NULL,
   station_type VARCHAR(50),
   capacity INTEGER DEFAULT 1,
   current_load INTEGER DEFAULT 0,
   status VARCHAR(20) DEFAULT 'available',
   equipment_list JSONB,
   location_coordinates JSONB, -- for 3D visualization
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Kitchen equipment
CREATE TABLE kitchen_equipment (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   station_id UUID REFERENCES kitchen_stations(id),
   name VARCHAR(100) NOT NULL,
   equipment_type VARCHAR(50),
   status VARCHAR(20) DEFAULT 'operational',
   last_maintenance DATE,
   next_maintenance DATE,
   specifications JSONB,
   created_at TIMESTAMP DEFAULT NOW()
);
```

2.8 Customer Service/POS Service

Technology: NestJS + TypeScript **Database**: PostgreSQL **Payment Integration**: Stripe/PayPal **Responsibilities**:

- Order management
- Payment processing
- Customer management
- Kitchen coordination
- Receipt generation

API Endpoints:

```
// Order Management
POST   /api/v1/pos/orders
GET    /api/v1/pos/orders
GET    /api/v1/pos/orders/:id
PUT    /api/v1/pos/orders/:id
DELETE /api/v1/pos/orders/:id

// Payment Processing
```

```
POST   /api/v1/pos/payments/process
POST   /api/v1/pos/payments/refund
GET    /api/v1/pos/payments/methods

// Customer Management
POST   /api/v1/pos/customers
GET    /api/v1/pos/customers
GET    /api/v1/pos/customers/:id
PUT    /api/v1/pos/customers/:id

// Kitchen Integration
POST   /api/v1/pos/orders/:id/send-to-kitchen
GET    /api/v1/pos/orders/:id/kitchen-status

// Receipts
GET    /api/v1/pos/orders/:id/receipt
POST   /api/v1/pos/orders/:id/receipt/email
```

Database Schema:

```
-- Customers table
CREATE TABLE customers (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   first_name VARCHAR(100),
   last_name VARCHAR(100),
   email VARCHAR(255),
   phone VARCHAR(20),
   date_of_birth DATE,
   address JSONB,
   loyalty_points INTEGER DEFAULT 0,
   total_orders INTEGER DEFAULT 0,
   total_spent DECIMAL(12,2) DEFAULT 0,
   last_order_date TIMESTAMP,
   preferences JSONB,
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Orders table
CREATE TABLE orders (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   order_number VARCHAR(50) UNIQUE NOT NULL,
   customer_id UUID REFERENCES customers(id),
   order_type VARCHAR(20) NOT NULL, -- 'dine_in', 'takeout', 'delivery'
   status VARCHAR(20) DEFAULT 'pending',
   table_number VARCHAR(20),
   subtotal DECIMAL(10,2) NOT NULL,
   tax_amount DECIMAL(10,2) DEFAULT 0,
   discount_amount DECIMAL(10,2) DEFAULT 0,
   total_amount DECIMAL(10,2) NOT NULL,
   payment_status VARCHAR(20) DEFAULT 'pending',
   special_instructions TEXT,
   estimated_ready_time TIMESTAMP,
   actual_ready_time TIMESTAMP,
   served_by UUID REFERENCES users(id),
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Order items table
CREATE TABLE order_items (
   id UUID PRIMARY KEY,
   order_id UUID REFERENCES orders(id),
   menu_item_id UUID,
   recipe_id UUID REFERENCES recipes(id),
   item_name VARCHAR(255) NOT NULL,
   quantity INTEGER NOT NULL,
   unit_price DECIMAL(10,2) NOT NULL,
   total_price DECIMAL(10,2) NOT NULL,
   modifications JSONB,
   special_instructions TEXT,
   created_at TIMESTAMP DEFAULT NOW()
);

-- Menu items table
CREATE TABLE menu_items (
   id UUID PRIMARY KEY,
   organization_id UUID NOT NULL,
   name VARCHAR(255) NOT NULL,
   description TEXT,
   category VARCHAR(100),
   price DECIMAL(10,2) NOT NULL,
   recipe_id UUID REFERENCES recipes(id),
   is_available BOOLEAN DEFAULT true,
   image_url VARCHAR(500),
   dietary_info JSONB,
   preparation_time INTEGER,
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP DEFAULT NOW()
);

-- Payments table
CREATE TABLE payments (
   id UUID PRIMARY KEY,
   order_id UUID REFERENCES orders(id),
   payment_method VARCHAR(50) NOT NULL,
   amount DECIMAL(10,2) NOT NULL,
   status VARCHAR(20) DEFAULT 'pending',
   transaction_id VARCHAR(255),
   gateway_response JSONB,
   processed_at TIMESTAMP,
   created_at TIMESTAMP DEFAULT NOW()
);
```

2.9 Accounting Service

Technology: NestJS + TypeScript **Database**: PostgreSQL + Analytics DB **Responsibilities**:

- Financial calculations
- P&L generation
- Cost analysis
- BCG matrix categorization
- Financial reporting

API Endpoints:

```
// Financial Calculations
GET    /api/v1/accounting/food-cost-ratio
GET    /api/v1/accounting/cost-per-unit
GET    /api/v1/accounting/profit-margins
GET    /api/v1/accounting/menu-analysis

// Reports
GET    /api/v1/accounting/reports/pl-statement
GET    /api/v1/accounting/reports/stock-valuation
GET    /api/v1/accounting/reports/usage
GET    /api/v1/accounting/reports/wastage

// Product Categorization
GET    /api/v1/accounting/product-categories/bcg-matrix
PUT    /api/v1/accounting/product-categories/update

// Real-time Monitoring
GET    /api/v1/accounting/dashboard/real-time
GET    /api/v1/accounting/alerts/financial
```

2.10 Analytics & AI Service

Technology: NestJS + Python (FastAPI) + TensorFlow **Database**: MongoDB + Elasticsearch **Responsibilities**:

- Demand forecasting
- Price optimization
- Waste reduction insights
- Performance analytics
- ML model serving

API Endpoints:

```
// Demand Forecasting
POST   /api/v1/analytics/forecast/demand
GET    /api/v1/analytics/forecast/inventory-needs

// Price Optimization
POST   /api/v1/analytics/pricing/optimize
GET    /api/v1/analytics/pricing/recommendations

// Waste Analysis
GET    /api/v1/analytics/waste/analysis
GET    /api/v1/analytics/waste/predictions

// Performance Analytics
GET    /api/v1/analytics/performance/kitchen
GET    /api/v1/analytics/performance/overall
```

3. Event-Driven Architecture

3.1 Event Bus Configuration

Technology: RabbitMQ with topic exchanges **Pattern**: Publish-Subscribe with routing keys

```
exchanges:
  restaurant.events:
    type: topic
    durable: true

queues:
  procurement.events:
    routing_keys: ["procurement.*", "stock.request.*"]
  stock.events:
    routing_keys: ["stock.*", "procurement.transfer.*"]
  processing.events:
    routing_keys: ["processing.*", "stock.transfer.*"]
  kitchen.events:
    routing_keys: ["kitchen.*", "processing.transfer.*", "pos.order.*"]
  pos.events:
    routing_keys: ["pos.*", "kitchen.order.*"]
  accounting.events:
    routing_keys: ["*.cost.*", "*.payment.*", "*.inventory.*"]
  analytics.events:
    routing_keys: ["*"]
```

3.2 Event Types and Schemas

```
// Purchase Order Events
interface PurchaseOrderCreated {
  eventType: 'procurement.purchase_order.created';
  organizationId: string;
  purchaseOrderId: string;
  totalAmount: number;
  supplierId: string;
  items: Array<{
    itemName: string;
    quantity: number;
    unitPrice: number;
  }>;
  timestamp: Date;
}

// Stock Events
interface StockTransferRequested {
  eventType: 'stock.transfer.requested';
  organizationId: string;
  transferId: string;
  fromDepartment: string;
  toDepartment: string;
  items: Array<{
    itemId: string;
    quantity: number;
  }>;
  timestamp: Date;
}

// Order Events
interface OrderCreated {
  eventType: 'pos.order.created';
  organizationId: string;
  orderId: string;
  customerId?: string;
  items: Array<{
    menuItemId: string;
    quantity: number;
    price: number;
  }>;
  totalAmount: number;
  timestamp: Date;
}

// Kitchen Events
interface KitchenOrderStatusUpdated {
  eventType: 'kitchen.order.status_updated';
  organizationId: string;
  orderId: string;
  status: string;
  estimatedReadyTime?: Date;
  timestamp: Date;
}
```

4. Data Architecture

4.1 Database Strategy

- **PostgreSQL**: Primary database for transactional data
- **MongoDB**: Analytics and logging data
- **Redis**: Caching and session storage
- **Elasticsearch**: Search and analytics

4.2 Data Synchronization Strategy

```
// Event-driven data synchronization
class DataSyncHandler {
  async handleInventoryUpdate(event: InventoryUpdated) {
    // Update analytics database
    await this.analyticsDB.updateInventoryMetrics(event);

    // Update search index
    await this.searchService.updateInventoryIndex(event);

    // Update cache
    await this.cacheService.invalidateInventoryCache(event.itemId);
  }
}
```

4.3 Caching Strategy

```
// Multi-level caching strategy
interface CacheStrategy {
  // L1: Application-level cache (in-memory)
  applicationCache: NodeCache;

  // L2: Distributed cache (Redis)
  distributedCache: Redis;

  // Cache keys patterns
  patterns: {
    user: "user:{userId}";
    inventory: "org:{orgId}:inventory:{itemId}";
    menu: "org:{orgId}:menu";
    orders: "org:{orgId}:orders:active";
  };

  // TTL configurations
  ttl: {
    user: 3600; // 1 hour
    inventory: 300; // 5 minutes
    menu: 1800; // 30 minutes
    orders: 60; // 1 minute
  };
}
```

5. Security Architecture

5.1 Authentication Flow

```
// JWT Authentication with refresh tokens
interface AuthenticationFlow {
  login: {
    endpoint: "/api/v1/auth/login";
    method: "POST";
    response: {
      accessToken: string; // 15 minutes expiry
      refreshToken: string; // 30 days expiry
      user: UserProfile;
    };
  };

  tokenValidation: {
    middleware: JwtAuthGuard;
    blacklistCheck: boolean;
    organizationCheck: boolean;
  };

  refreshToken: {
    endpoint: "/api/v1/auth/refresh";
    validation: RefreshTokenGuard;
  };
}
```

5.2 Authorization Matrix

```
interface RolePermissions {
  SUPER_ADMIN: {
    modules: ['*'];
    actions: ['*'];
  };

  PROCUREMENT_MANAGER: {
    modules: ['procurement', 'suppliers', 'stock_requests'];
    actions: ['create', 'read', 'update', 'delete'];
  };

  STOCK_CONTROLLER: {
    modules: ['inventory', 'stock_transfers', 'stock_audits'];
    actions: ['create', 'read', 'update', 'validate'];
  };

  PROCESSING_STAFF: {
    modules: ['recipes', 'production_plans', 'kitchen_transfers'];
    actions: ['create', 'read', 'update'];
  };

  KITCHEN_STAFF: {
    modules: ['kitchen_orders', 'kitchen_status', 'equipment'];
    actions: ['read', 'update_status'];
  };

  POS_OPERATOR: {
    modules: ['orders', 'customers', 'payments', 'menu'];
    actions: ['create', 'read', 'update', 'process_payment'];
  };
}
```

5.3 API Security Middleware Stack

```
// Security middleware chain
const securityMiddleware = [
  helmet(), // Security headers
  cors(corsOptions), // CORS configuration
  rateLimit(rateLimitOptions), // Rate limiting
  validateApiKey(), // API key validation
  authenticateJWT(), // JWT authentication
  authorizePermissions(), // Role-based authorization
  validateOrganization(), // Multi-tenant validation
  auditLog(), // Request logging
];

// Rate limiting configuration
const rateLimitOptions = {
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 1000, // Requests per window
  standardHeaders: true,
  legacyHeaders: false,
  keyGenerator: (req) => `${req.user?.id || req.ip}:${req.user?.organizationId}`,
};
```

6. Monitoring and Observability

6.1 Logging Architecture

```
// Centralized logging with ELK Stack
interface LoggingConfig {
  levels: ['error', 'warn', 'info', 'debug'];

  transports: {
    console: ConsoleTransport;
    elasticsearch: ElasticsearchTransport;
    file: FileTransport;
  };

  format: {
    timestamp: true;
    correlationId: true;
    userId: true;
    organizationId: true;
    service: string;
    module: string;
  };
}

// Structured logging example
class Logger {
  info(message: string, meta: LogMeta) {
    this.log('info', message, {
      ...meta,
      correlationId: AsyncContext.getCorrelationId(),
      userId: AsyncContext.getUserId(),
      organizationId: AsyncContext.getOrganizationId(),
      service: 'procurement-service',
      timestamp: new Date().toISOString(),
    });
  }
}
```

6.2 Metrics and Monitoring

```
// Prometheus metrics configuration
interface MetricsConfig {
  businessMetrics: {
    orderProcessingTime: Histogram;
    inventoryAccuracy: Gauge;
    revenuePerHour: Counter;
    customerSatisfaction: Gauge;
  };

  technicalMetrics: {
    apiResponseTime: Histogram;
    databaseConnections: Gauge;
    cacheHitRatio: Gauge;
    errorRate: Counter;
    throughputPerSecond: Counter;
  };

  customMetrics: {
    stockDiscrepancyRate: Gauge;
    kitchenEfficiency: Histogram;
    paymentSuccessRate: Gauge;
    aiRecommendationAccuracy: Gauge;
  };
}

// Health check endpoints
const healthChecks = {
  '/health': basicHealthCheck,
  '/health/ready': readinessProbe,
  '/health/live': livenessProbe,
  '/metrics': prometheusMetrics,
};
```

6.3 Distributed Tracing

```
// OpenTelemetry configuration
interface TracingConfig {
  serviceName: string;
  version: string;
  exporters: {
    jaeger: JaegerExporter;
    console: ConsoleExporter;
  };

  instrumentations: [
    HttpInstrumentation;
    ExpressInstrumentation;
    PostgreSQLInstrumentation;
    RedisInstrumentation;
  ];

  samplingRatio: 0.1; // 10% sampling in production
}
```

7. Performance and Scalability

7.1 Horizontal Scaling Strategy

```
# Kubernetes deployment configuration
apiVersion: apps/v1
kind: Deployment
metadata:
  name: procurement-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: procurement-service
  template:
    metadata:
      labels:
        app: procurement-service
    spec:
      containers:
      - name: procurement-service
        image: restaurant-crm/procurement-service:latest
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        env:
```

```
        - name: DB_HOST
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: host
---
apiVersion: v1
kind: Service
metadata:
  name: procurement-service-svc
spec:
  selector:
    app: procurement-service
  ports:
  - port: 3002
    targetPort: 3002
  type: ClusterIP
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: procurement-service-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: procurement-service
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

7.2 Database Optimization

```
-- Performance optimization indexes
-- Inventory queries
```

```
CREATE INDEX CONCURRENTLY idx_inventory_items_org_active
ON inventory_items(organization_id, is_active)
WHERE is_active = true;

CREATE INDEX CONCURRENTLY idx_inventory_items_sku_org
ON inventory_items(sku, organization_id);

-- Order queries
CREATE INDEX CONCURRENTLY idx_orders_org_date_status
ON orders(organization_id, created_at DESC, status);

CREATE INDEX CONCURRENTLY idx_orders_customer_date
ON orders(customer_id, created_at DESC);

-- Stock movements for analytics
CREATE INDEX CONCURRENTLY idx_stock_movements_item_date
ON stock_movements(inventory_item_id, created_at DESC);

-- Purchase orders
CREATE INDEX CONCURRENTLY idx_purchase_orders_org_status_date
ON purchase_orders(organization_id, status, created_at DESC);

-- Partitioning for large tables
CREATE TABLE stock_movements_y2025m01 PARTITION OF stock_movements
FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');

CREATE TABLE stock_movements_y2025m02 PARTITION OF stock_movements
FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');
```

7.3 Caching Strategy Implementation

```
// Multi-layer caching implementation
class CacheService {
  private l1Cache: NodeCache; // Application-level
  private l2Cache: Redis; // Distributed

  async get<T>(key: string): Promise<T | null> {
    // Check L1 cache first
    let value = this.l1Cache.get<T>(key);
    if (value) {
      return value;
    }

    // Check L2 cache
    const cached = await this.l2Cache.get(key);
    if (cached) {
      value = JSON.parse(cached);
      // Populate L1 cache
      this.l1Cache.set(key, value, 300); // 5 minutes
      return value;
    }

    return null;
  }

  async set<T>(key: string, value: T, ttl: number = 3600): Promise<void> {
    // Set in both caches
    this.l1Cache.set(key, value, Math.min(ttl, 300)); // L1 max 5 minutes
    await this.l2Cache.setex(key, ttl, JSON.stringify(value));
  }

  async invalidate(pattern: string): Promise<void> {
    // Invalidate L1 cache
    this.l1Cache.flushAll();

    // Invalidate L2 cache by pattern
    const keys = await this.l2Cache.keys(pattern);
    if (keys.length > 0) {
      await this.l2Cache.del(...keys);
    }
  }
}

// Cache warming strategies
class CacheWarmer {
  async warmInventoryCache(organizationId: string): Promise<void> {
    const inventory = await this.inventoryService.getActiveInventory(organizationId);
    const cacheKey = `org:${organizationId}:inventory:active`;
    await this.cacheService.set(cacheKey, inventory, 300); // 5 minutes
  }

  async warmMenuCache(organizationId: string): Promise<void> {
    const menu = await this.menuService.getActiveMenu(organizationId);
    const cacheKey = `org:${organizationId}:menu:active`;
    await this.cacheService.set(cacheKey, menu, 1800); // 30 minutes
  }
}
```

8. Real-time Features Implementation

8.1 WebSocket Architecture

```
// Socket.IO server configuration
```

```
interface WebSocketConfig {
  transports: ['websocket', 'polling'];
  cors: {
    origin: process.env.ALLOWED_ORIGINS;
    methods: ['GET', 'POST'];
  };

  namespaces: {
    '/kitchen': KitchenNamespace;
    '/admin': AdminNamespace;
    '/pos': POSNamespace;
    '/notifications': NotificationNamespace;
  };
}

// Kitchen real-time updates
class KitchenNamespace {
  async handleConnection(socket: Socket): Promise<void> {
    const user = await this.authService.validateSocketToken(socket.handshake.auth.token);
    const organizationId = user.organizationId;

    // Join organization room
    socket.join(`org:${organizationId}`);

    // Join department-specific room
    if (user.department === 'kitchen') {
      socket.join(`org:${organizationId}:kitchen`);
    }

    // Send current kitchen status
    const kitchenStatus = await this.kitchenService.getCurrentStatus(organizationId);
    socket.emit('kitchen:status', kitchenStatus);

    // Handle order status updates
    socket.on('kitchen:order:update', async (data) => {
      await this.handleOrderStatusUpdate(data, organizationId);
    });
  }

  async broadcastOrderUpdate(organizationId: string, orderUpdate: OrderUpdate):
Promise<void> {
    this.io.to(`org:${organizationId}`).emit('kitchen:order:updated', orderUpdate);

    // Also notify POS
    this.io.to(`org:${organizationId}:pos`).emit('pos:order:kitchen_update', orderUpdate);
  }
}
```

8.2 3D Visualization Data Feed

```
// 3D Kitchen Visualization Service
interface KitchenVisualizationData {
  stations: Array<{
    id: string;
    position: { x: number; y: number; z: number };
    status: 'idle' | 'busy' | 'maintenance';
    currentOrders: string[];
    efficiency: number;
  }>;

  staff: Array<{
    id: string;
    position: { x: number; y: number; z: number };
    status: 'working' | 'break' | 'idle';
    currentTask: string;
  }>;

  orders: Array<{
    id: string;
    status: 'preparing' | 'cooking' | 'ready';
    station: string;
    estimatedCompletion: Date;
    progress: number; // 0-100
  }>;

  equipment: Array<{
    id: string;
    type: 'oven' | 'grill' | 'fryer' | 'prep_station';
    status: 'operational' | 'in_use' | 'maintenance';
    temperature?: number;
    utilization: number;
  }>;
}

class VisualizationService {
  async getKitchenVisualizationData(organizationId: string):
Promise<KitchenVisualizationData> {
    const [stations, staff, orders, equipment] = await Promise.all([
      this.getKitchenStations(organizationId),
      this.getActiveStaff(organizationId),
      this.getActiveOrders(organizationId),
      this.getEquipmentStatus(organizationId),
    ]);

    return {
      stations: stations.map(this.mapStationToVisualization),
      staff: staff.map(this.mapStaffToVisualization),
      orders: orders.map(this.mapOrderToVisualization),
      equipment: equipment.map(this.mapEquipmentToVisualization),
    };
  }

  async updateVisualization(organizationId: string, update:
Partial<KitchenVisualizationData>): Promise<void> {
    // Emit real-time updates to connected clients
    this.socketService.emitToOrganization(organizationId, '3d:kitchen:update', update);

    // Store in cache for quick retrieval
    const cacheKey = `org:${organizationId}:3d:kitchen`;
    await this.cacheService.set(cacheKey, update, 30); // 30 seconds TTL
  }
}
```

9. AI/ML Integration Architecture

9.1 ML Pipeline Architecture

```
// AI Service Architecture
interface AIServiceConfig {
  models: {
    demandForecasting: {
      type: 'tensorflow';
      modelPath: '/models/demand_forecast.pb';
      inputFeatures: ['historical_sales', 'seasonality', 'weather', 'events'];
    };

    priceOptimization: {
      type: 'openai';
      model: 'gpt-4';
      endpoint: '/v1/chat/completions';
    };

    yieldCalculation: {
      type: 'tensorflow';
      modelPath: '/models/yield_calculator.pb';
      inputFeatures: ['recipe_id', 'ingredient_quantities', 'cooking_method'];
    };

    wasteReduction: {
      type: 'scikit-learn';
      modelPath: '/models/waste_predictor.pkl';
      inputFeatures: ['inventory_age', 'demand_forecast', 'historical_waste'];
    };
  };
}

// ML Model Service
class MLModelService {
  private models: Map<string, any> = new Map();

  async initializeModels(): Promise<void> {
    // Load TensorFlow models
    const demandModel = await tf.loadLayersModel('file:///models/demand_forecast.json');
    this.models.set('demand_forecasting', demandModel);

    const yieldModel = await tf.loadLayersModel('file:///models/yield_calculator.json');
    this.models.set('yield_calculation', yieldModel);
  }

  async predictDemand(organizationId: string, itemId: string, forecastDays: number = 7):
Promise<DemandForecast> {
    const historicalData = await this.getHistoricalSalesData(organizationId, itemId);
    const features = this.prepareFeatures(historicalData);

    const model = this.models.get('demand_forecasting');
    const prediction = model.predict(features) as tf.Tensor;
    const result = await prediction.data();

    return {
      itemId,
      forecastPeriod: forecastDays,
      predictions: Array.from(result),
      confidence: this.calculateConfidence(result),
      generatedAt: new Date(),
    };
  }

  async calculateOptimalYield(recipeId: string, targetQuantity: number):
Promise<YieldCalculation> {
    const recipe = await this.recipeService.getRecipe(recipeId);
    const features = this.prepareYieldFeatures(recipe, targetQuantity);

    const model = this.models.get('yield_calculation');
    const prediction = model.predict(features) as tf.Tensor;
    const result = await prediction.data();

    return {
      recipeId,
      targetQuantity,
      optimizedIngredients: this.mapResultToIngredients(recipe, result),
      estimatedYield: result[0],
      confidenceScore: this.calculateConfidence(result),
      costOptimization: await this.calculateCostOptimization(recipe, result),
    };
  }
}
```

9.2 Recommendation Engine

```
// AI Recommendation Service
class RecommendationEngine {
  async generateMenuRecommendations(organizationId: string):
Promise<MenuRecommendations> {
    const [salesData, inventoryData, profitabilityData] = await Promise.all([
      this.getSalesAnalytics(organizationId),
      this.getInventoryAnalytics(organizationId),
      this.getProfitabilityAnalysis(organizationId),
    ]);

    const prompt = this.buildRecommendationPrompt(salesData, inventoryData,
profitabilityData);

    const response = await this.openaiClient.chat.completions.create({
      model: 'gpt-4',
      messages: [
        { role: 'system', content: 'You are a restaurant business optimization expert.' },
        { role: 'user', content: prompt },
      ],
      temperature: 0.7,
      max_tokens: 1500,
    });

    return this.parseRecommendations(response.choices[0].message.content);
  }

  async generatePricingOptimization(organizationId: string):
Promise<PricingRecommendations> {
    const marketData = await this.getMarketAnalysis(organizationId);
    const competitorData = await this.getCompetitorPricing(organizationId);
    const demandElasticity = await this.calculateDemandElasticity(organizationId);

    return {
      recommendations: await this.calculateOptimalPricing(marketData, competitorData,
demandElasticity),
      projectedImpact: await this.simulatePricingImpact(organizationId),
      implementationPlan: await this.generateImplementationPlan(),
    };
  }
}
```

10. Data Analytics and Reporting

10.1 Analytics Data Pipeline

```
// Analytics Pipeline Configuration
interface AnalyticsPipeline {
  dataIngestion: {
    sources: ['postgresql', 'mongodb', 'redis', 'external_apis'];
    frequency: 'real-time' | 'batch';
    transformation: ETLProcessor;
  };

  dataWarehouse: {
    storage: 'clickhouse' | 'bigquery';
    schema: AnalyticsSchema;
    partitioning: 'daily' | 'monthly';
  };

  processing: {
    streamProcessing: 'kafka_streams';
    batchProcessing: 'spark';
    mlPipeline: 'kubeflow';
  };
}

// Real-time Analytics Processor
class AnalyticsProcessor {
  async processOrderEvent(event: OrderEvent): Promise<void> {
    // Update real-time metrics
    await this.updateRealtimeMetrics(event);

    // Update aggregations
    await this.updateHourlyAggregations(event);
    await this.updateDailyAggregations(event);

    // Trigger ML predictions if needed
    if (this.shouldTriggerPrediction(event)) {
      await this.triggerDemandForecast(event.organizationId);
    }
  }

  async generateBusinessIntelligence(organizationId: string):
Promise<BusinessIntelligenceReport> {
    const timeRange = { start: moment().subtract(30, 'days'), end: moment() };

    const [
      salesMetrics,
      inventoryMetrics,
      kitchenMetrics,
      customerMetrics,
      financialMetrics,
    ] = await Promise.all([
      this.calculateSalesMetrics(organizationId, timeRange),
      this.calculateInventoryMetrics(organizationId, timeRange),
      this.calculateKitchenMetrics(organizationId, timeRange),
      this.calculateCustomerMetrics(organizationId, timeRange),
      this.calculateFinancialMetrics(organizationId, timeRange),
    ]);

    return {
      summary: this.generateExecutiveSummary(salesMetrics, financialMetrics),
      salesAnalysis: salesMetrics,
      inventoryAnalysis: inventoryMetrics,
      operationalEfficiency: kitchenMetrics,
      customerInsights: customerMetrics,
      financialPerformance: financialMetrics,
      recommendations: await this.generateActionableRecommendations(organizationId),
      generatedAt: new Date(),
    };
  }
}
```

10.2 Automated Reporting System

```
// Report Generation Service
class ReportingEngine {
  private reportSchedules: Map<string, ReportSchedule> = new Map();

  async scheduleReport(config: ReportConfig): Promise<void> {
    const schedule = cron.schedule(config.cronExpression, async () => {
      await this.generateAndDistributeReport(config);
    });

    this.reportSchedules.set(config.id, { config, schedule });
  }

  async generateDailyPLStatement(organizationId: string, date: Date):
Promise<PLStatement> {
    const [revenue, costs, expenses] = await Promise.all([
      this.calculateDailyRevenue(organizationId, date),
      this.calculateDailyCosts(organizationId, date),
      this.calculateDailyExpenses(organizationId, date),
    ]);

    const grossProfit = revenue.total - costs.cogs;
    const netProfit = grossProfit - expenses.total;

    return {
      date,
      organizationId,
      revenue: {
        foodSales: revenue.food,
        beverageSales: revenue.beverages,
        otherRevenue: revenue.other,
        total: revenue.total,
      },
      costs: {
        foodCosts: costs.food,
        beverageCosts: costs.beverages,
        laborCosts: costs.labor,
        overheadCosts: costs.overhead,
        totalCOGS: costs.cogs,
      },
      grossProfit: {
        amount: grossProfit,
        margin: (grossProfit / revenue.total) * 100,
      },
      expenses: {
        rent: expenses.rent,
        utilities: expenses.utilities,
        marketing: expenses.marketing,
        other: expenses.other,
        total: expenses.total,
      },
      netProfit: {
        amount: netProfit,
        margin: (netProfit / revenue.total) * 100,
      },
      kpis: {
        foodCostRatio: (costs.food / revenue.food) * 100,
        laborCostRatio: (costs.labor / revenue.total) * 100,
        customerCount: await this.getCustomerCount(organizationId, date),
        averageTicket: revenue.total / await this.getTransactionCount(organizationId, date),
      },
    };
  }
}
```

11. Deployment and Infrastructure

11.1 Containerization Strategy

```
# Multi-stage Dockerfile for Node.js services
FROM node:18-alpine AS builder
WORKDIR /app

# Copy package files
COPY package*.json ./
COPY tsconfig.json ./

# Install dependencies
RUN npm ci --only=production && npm cache clean --force

# Copy source code
COPY src/ src/

# Build application
RUN npm run build

# Production stage
FROM node:18-alpine AS production
WORKDIR /app

# Install dumb-init for proper signal handling
RUN apk add --no-cache dumb-init

# Create non-root user
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001

# Copy built application
COPY --from=builder --chown=nodejs:nodejs /app/dist ./dist
COPY --from=builder --chown=nodejs:nodejs /app/node_modules ./node_modules
COPY --from=builder --chown=nodejs:nodejs /app/package.json ./

# Set user
USER nodejs

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD node dist/health-check.js

# Expose port
EXPOSE 3000

# Start application
```

```
ENTRYPOINT ["dumb-init", "--"]
CMD ["node", "dist/main.js"]
```

11.2 Kubernetes Deployment Configuration

```yaml
# Complete Kubernetes deployment with monitoring
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  NODE_ENV: "production"
  LOG_LEVEL: "info"
  REDIS_HOST: "redis-service"
  DB_HOST: "postgres-service"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: restaurant-crm-api
  labels:
    app: restaurant-crm-api
    version: v1
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  selector:
    matchLabels:
      app: restaurant-crm-api
  template:
    metadata:
      labels:
        app: restaurant-crm-api
        version: v1
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "3000"
        prometheus.io/path: "/metrics"
    spec:
      serviceAccountName: restaurant-crm-sa
      containers:
      - name: api
        image: restaurant-crm/api:latest
        ports:
        - containerPort: 3000
          name: http
        envFrom:
        - configMapRef:
            name: app-config
        - secretRef:
            name: app-secrets
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health/live
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3
        readinessProbe:
          httpGet:
            path: /health/ready
            port: 3000
          initialDelaySeconds: 5
          periodSeconds: 5
          timeoutSeconds: 3
          failureThreshold: 3
        securityContext:
          runAsNonRoot: true
          runAsUser: 1001
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
---
apiVersion: v1
kind: Service
metadata:
  name: restaurant-crm-api-service
  labels:
    app: restaurant-crm-api
spec:
  selector:
    app: restaurant-crm-api
  ports:
  - name: http
    port: 80
    targetPort: 3000
    protocol: TCP
  type: ClusterIP
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: restaurant-crm-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  tls:
  - hosts:
    - api.restaurant-crm.com
    secretName: restaurant-crm-tls
  rules:
  - host: api.restaurant-crm.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: restaurant-crm-api-service
            port:
              number: 80
```

12. Disaster Recovery and Backup Strategy

12.1 Backup Configuration

```
// Automated Backup Service
class BackupService {
  private backupSchedules = {
    database: '0 2 * * *', // Daily at 2 AM
    files: '0 3 * * *',    // Daily at 3 AM
    logs: '0 1 * * *',     // Daily at 1 AM
  };

  async initializeBackupJobs(): Promise<void> {
    // Database backups
    cron.schedule(this.backupSchedules.database, async () => {
      await this.performDatabaseBackup();
    });

    // File backups
    cron.schedule(this.backupSchedules.files, async () => {
      await this.performFileBackup();
    });

    // Log backups
    cron.schedule(this.backupSchedules.logs, async () => {
      await this.performLogBackup();
    });
  }

  async performDatabaseBackup(): Promise<void> {
    const timestamp = moment().format('YYYYMMDD-HHmmss');
    const backupFile = `db-backup-${timestamp}.sql`;

    try {
      // Create database dump
      await this.execShellCommand(`pg_dump ${process.env.DATABASE_URL} >
/backups/${backupFile}`);

      // Compress backup
      await this.execShellCommand(`gzip /backups/${backupFile}`);

      // Upload to cloud storage
      await this.uploadToCloudStorage(`/backups/${backupFile}.gz`,
`database/${backupFile}.gz`);

      // Clean old backups (keep last 30 days)
      await this.cleanOldBackups('database', 30);

      this.logger.info('Database backup completed successfully', { backupFile });
    } catch (error) {
      this.logger.error('Database backup failed', { error, backupFile });
      await this.notificationService.sendAlert('backup-failed', { type: 'database', error });
    }
  }
}
```

12.2 High Availability Configuration

```
# Database High Availability with PostgreSQL
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
```

```yaml
metadata:
  name: postgres-cluster
spec:
  instances: 3

  postgresql:
    parameters:
      max_connections: "200"
      shared_buffers: "256MB"
      effective_cache_size: "1GB"

  bootstrap:
    initdb:
      database: restaurant_crm
      owner: app_user

  storage:
    size: 100Gi
    storageClass: fast-ssd

  monitoring:
    enabled: true

  backup:
    retentionPolicy: "30d"
    barmanObjectStore:
      s3Credentials:
        accessKeyId:
          name: backup-s3-credentials
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: backup-s3-credentials
          key: SECRET_ACCESS_KEY
      destinationPath: "s3://restaurant-crm-backups/postgresql"
---
# Redis High Availability
apiVersion: databases.spotahome.com/v1
kind: RedisFailover
metadata:
  name: redis-cluster
spec:
  sentinel:
    replicas: 3
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 200m
        memory: 256Mi
  redis:
    replicas: 3
    resources:
      requests:
        cpu: 200m
        memory: 256Mi
      limits:
        cpu: 500m
        memory: 512Mi
    storage:
      persistentVolumeClaim:
        metadata:
          name: redis-storage
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 10Gi
```

13. Testing Strategy

13.1 Testing Architecture

```
// Comprehensive Testing Strategy
interface TestingStrategy {
  unitTests: {
    framework: 'jest';
    coverage: 90; // minimum coverage percentage
    testTypes: ['service', 'repository', 'utility', 'validation'];
  };

  integrationTests: {
    framework: 'supertest + jest';
    testTypes: ['api', 'database', 'external_services'];
    testContainers: 'testcontainers';
  };

  e2eTests: {
    framework: 'playwright';
    environments: ['staging', 'production'];
    scenarios: ['user_workflows', 'business_processes'];
  };
  performanceTests: {
    framework: 'k6';
    testTypes: ['load', 'stress', 'spike', 'volume'];
    metrics: ['response_time', 'throughput', 'error_rate'];
  };

  securityTests: {
    framework: 'owasp-zap';
    testTypes: ['vulnerability_scan', 'penetration_test'];
    compliance: ['owasp-top-10', 'pci-dss'];
  };
}

// Unit Test Examples
describe('ProcurementService', () => {
  let service: ProcurementService;
  let mockRepository: jest.Mocked<ProcurementRepository>;
  let mockEventBus: jest.Mocked<EventBus>;

  beforeEach(async () => {
    const module = await Test.createTestingModule({
      providers: [
        ProcurementService,
        {
          provide: ProcurementRepository,
          useFactory: () => ({
            create: jest.fn(),
            findById: jest.fn(),
            update: jest.fn(),
            delete: jest.fn(),
          }),
        },
        {
          provide: EventBus,
          useFactory: () => ({
            publish: jest.fn(),
          }),
        },
      ],
    }).compile();

    service = module.get<ProcurementService>(ProcurementService);
    mockRepository = module.get(ProcurementRepository);
    mockEventBus = module.get(EventBus);
  });

  describe('createPurchaseOrder', () => {
    it('should create purchase order and publish event', async () => {
```

```
    // Arrange
    const createDto = {
      supplierId: 'supplier-123',
      items: [{ name: 'Item 1', quantity: 10, unitPrice: 5.00 }],
      totalAmount: 50.00,
    };
    const expectedPO = { id: 'po-123', ...createDto, status: 'draft' };

    mockRepository.create.mockResolvedValue(expectedPO);

    // Act
    const result = await service.createPurchaseOrder('org-123', createDto);

    // Assert
    expect(result).toEqual(expectedPO);
    expect(mockRepository.create).toHaveBeenCalledWith(createDto);
    expect(mockEventBus.publish).toHaveBeenCalledWith({
      eventType: 'procurement.purchase_order.created',
      organizationId: 'org-123',
      purchaseOrderId: 'po-123',
      totalAmount: 50.00,
      supplierId: 'supplier-123',
      items: createDto.items,
      timestamp: expect.any(Date),
    });
   });

   it('should throw error when supplier not found', async () => {
    // Arrange
    const createDto = {
      supplierId: 'invalid-supplier',
      items: [{ name: 'Item 1', quantity: 10, unitPrice: 5.00 }],
      totalAmount: 50.00,
    };

    mockRepository.create.mockRejectedValue(new Error('Supplier not found'));

    // Act & Assert
    await expect(service.createPurchaseOrder('org-123', createDto))
      .rejects.toThrow('Supplier not found');
    expect(mockEventBus.publish).not.toHaveBeenCalled();
   });
  });
});

// Integration Test Example
describe('Procurement API Integration', () => {
  let app: INestApplication;
let dbContainer: StartedTestContainer;
let redisContainer: StartedTestContainer;

beforeAll(async () => {
  // Start test containers
  dbContainer = await new PostgreSqlContainer()
    .withDatabase('test_db')
    .withUsername('test_user')
    .withPassword('test_pass')
    .start();

  redisContainer = await new RedisContainer().start();

  // Setup test app
  const moduleRef = await Test.createTestingModule({
    imports: [AppModule],
  })
  .overrideProvider(DatabaseConfig)
  .useValue({
    host: dbContainer.getHost(),
    port: dbContainer.getPort(),
    database: 'test_db',
    username: 'test_user',
    password: 'test_pass',
  })
  .compile();

  app = moduleRef.createNestApplication();
  await app.init();
});

afterAll(async () => {
  await app.close();
  await dbContainer.stop();
  await redisContainer.stop();
});

describe('POST /api/v1/procurement/purchase-orders', () => {
  it('should create purchase order with valid data', async () => {
    const createDto = {
      supplierId: 'supplier-123',
      items: [{ name: 'Test Item', quantity: 5, unitPrice: 10.00 }],
      totalAmount: 50.00,
    };

    const response = await request(app.getHttpServer())
      .post('/api/v1/procurement/purchase-orders')
      .send(createDto)
```

```
        .set('Authorization', `Bearer ${await getValidToken()}`)
        .expect(201);

      expect(response.body).toMatchObject({
        id: expect.any(String),
        status: 'draft',
        totalAmount: 50.00,
        supplierId: 'supplier-123',
      });
    });
  });
});
```

13.2 Performance Testing Configuration

```
// K6 Performance Test Scripts
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Rate } from 'k6/metrics';

// Custom metrics
export let errorRate = new Rate('errors');

export let options = {
  stages: [
    { duration: '2m', target: 100 }, // Ramp up to 100 users
    { duration: '5m', target: 100 }, // Stay at 100 users
    { duration: '2m', target: 200 }, // Ramp up to 200 users
    { duration: '5m', target: 200 }, // Stay at 200 users
    { duration: '2m', target: 0 },   // Ramp down to 0 users
  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% of requests under 500ms
    http_req_failed: ['rate<0.1'],    // Error rate under 10%
    errors: ['rate<0.1'],
  },
};

const BASE_URL = 'https://api.restaurant-crm.com';

export function setup() {
  // Setup test data
  const authResponse = http.post(`${BASE_URL}/api/v1/auth/login`, {
    email: 'test@example.com',
    password: 'testpassword',
  });
  return { token: authResponse.json('accessToken') };
}

export default function(data) {
  const headers = {
    'Authorization': `Bearer ${data.token}`,
    'Content-Type': 'application/json',
  };

  // Test scenarios
  group('Inventory Management', () => {
    // Get inventory list
    let response = http.get(`${BASE_URL}/api/v1/stock/inventory`, { headers });
    check(response, {
      'inventory list status 200': (r) => r.status === 200,
      'inventory list response time < 500ms': (r) => r.timings.duration < 500,
    }) || errorRate.add(1);

    sleep(1);

    // Create stock transfer
    response = http.post(`${BASE_URL}/api/v1/stock/transfers/outgoing`,
      JSON.stringify({
        toDepartment: 'processing',
        items: [
          { inventoryItemId: 'item-123', requestedQuantity: 10 }
        ],
        notes: 'Performance test transfer'
      }),
      { headers }
    );

    check(response, {
      'create transfer status 201': (r) => r.status === 201,
      'create transfer response time < 1s': (r) => r.timings.duration < 1000,
    }) || errorRate.add(1);
  });

  group('Order Processing', () => {
    // Create order
    let response = http.post(`${BASE_URL}/api/v1/pos/orders`,
      JSON.stringify({
        customerId: 'customer-123',
        orderType: 'dine_in',
        items: [
          { menuItemId: 'menu-item-1', quantity: 2, unitPrice: 15.00 }
        ],
        totalAmount: 30.00
      }),
      { headers }
    );

    check(response, {
      'create order status 201': (r) => r.status === 201,
      'create order response time < 800ms': (r) => r.timings.duration < 800,
    }) || errorRate.add(1);

    const orderId = response.json('id');

    // Update order status
    response = http.put(`${BASE_URL}/api/v1/kitchen/orders/${orderId}/status`,
      JSON.stringify({ status: 'preparing' }),
      { headers }
    );

    check(response, {
      'update order status 200': (r) => r.status === 200,
      'update order response time < 300ms': (r) => r.timings.duration < 300,
    }) || errorRate.add(1);
  });

  sleep(Math.random() * 2 + 1); // Random sleep between 1-3 seconds
}
```

14. Security Implementation Details

14.1 Advanced Security Middleware

```
// Security Middleware Stack
class SecurityMiddleware {
 // Request sanitization
 static sanitizeInput() {
  return (req: Request, res: Response, next: NextFunction) => {
   // Remove potential XSS payloads
   const sanitize = (obj: any): any => {
    if (typeof obj === 'string') {
     return DOMPurify.sanitize(obj);
    }
    if (Array.isArray(obj)) {
     return obj.map(sanitize);
    }
    if (obj && typeof obj === 'object') {
     const sanitized: any = {};
     for (const key in obj) {
          sanitized[key] = sanitize(obj[key]);
        }
        return sanitized;
      }
      return obj;
    };

    req.body = sanitize(req.body);
    req.query = sanitize(req.query);
    next();
  };
}

// SQL injection prevention
static preventSQLInjection() {
  return (req: Request, res: Response, next: NextFunction) => {
    const sqlInjectionPattern =
/(\b(ALTER|CREATE|DELETE|DROP|EXEC(UTE)?|INSERT|SELECT|UNION|UPDATE)\b)/i;

    const checkForSQLInjection = (value: string): boolean => {
      return sqlInjectionPattern.test(value);
    };

    const validateObject = (obj: any): boolean => {
      if (typeof obj === 'string') {
        return checkForSQLInjection(obj);
      }
      if (Array.isArray(obj)) {
        return obj.some(validateObject);
      }
      if (obj && typeof obj === 'object') {
        return Object.values(obj).some(validateObject);
      }
      return false;
    };

    if (validateObject(req.body) || validateObject(req.query)) {
      return res.status(400).json({
        error: 'Invalid request parameters',
        code: 'SECURITY_VIOLATION'
      });
    }

    next();
  };
}

// Rate limiting with Redis
```

```
static advancedRateLimit() {
  const limiter = rateLimit({
    store: new RedisStore({
      sendCommand: (...args: string[]) => redisClient.call(...args),
    }),
    windowMs: 15 * 60 * 1000, // 15 minutes
    max: async (req: Request) => {
      // Different limits based on user type
      if (req.user?.role === 'SUPER_ADMIN') return 2000;
      if (req.user?.role === 'MANAGER') return 1000;
      return 500; // Regular users
    },
    message: {
      error: 'Too many requests',
      retryAfter: 15 * 60 * 1000
    },
    standardHeaders: true,
    legacyHeaders: false,
    keyGenerator: (req: Request) => {
      return `${req.user?.id || req.ip}:${req.user?.organizationId || 'anonymous'}`;
    },
    skip: (req: Request) => {
      // Skip rate limiting for health checks
      return req.path.startsWith('/health');
    }
  });

  return limiter;
}

// Request validation and audit logging
static auditLogger() {
  return (req: Request, res: Response, next: NextFunction) => {
    const startTime = Date.now();

    // Log request
    const requestLog = {
      timestamp: new Date(),
      method: req.method,
      url: req.url,
      userAgent: req.get('User-Agent'),
      ip: req.ip,
      userId: req.user?.id,
      organizationId: req.user?.organizationId,
      correlationId: req.headers['x-correlation-id'] || uuidv4(),
    };

    // Store correlation ID in async context
```

```
      AsyncContext.run(requestLog.correlationId, () => {
        res.on('finish', () => {
          const responseTime = Date.now() - startTime;

          // Log response
          const responseLog = {
            ...requestLog,
            statusCode: res.statusCode,
            responseTime,
            contentLength: res.get('Content-Length'),
          };

          // Log to audit system
          auditLogger.info('API Request', responseLog);

          // Alert on suspicious activity
          if (responseTime > 5000 || res.statusCode >= 500) {
            alertingService.sendAlert('api_performance_issue', responseLog);
          }
        });

        next();
      });
    };
  }
}

// Data encryption service
class EncryptionService {
  private readonly algorithm = 'aes-256-gcm';
  private readonly keyDerivationSalt = process.env.ENCRYPTION_SALT;

  async encryptSensitiveData(data: string, context: string = 'default'):
Promise<EncryptedData> {
    const key = await this.deriveKey(context);
    const iv = crypto.randomBytes(16);

    const cipher = crypto.createCipher(this.algorithm, key);
    cipher.setAAD(Buffer.from(context));

    let encrypted = cipher.update(data, 'utf8', 'hex');
    encrypted += cipher.final('hex');

    const authTag = cipher.getAuthTag();

    return {
      encrypted,
      iv: iv.toString('hex'),
      authTag: authTag.toString('hex'),
      algorithm: this.algorithm,
    };
  }

  async decryptSensitiveData(encryptedData: EncryptedData, context: string = 'default'):
Promise<string> {
    const key = await this.deriveKey(context);

    const decipher = crypto.createDecipher(this.algorithm, key);
    decipher.setAAD(Buffer.from(context));
    decipher.setAuthTag(Buffer.from(encryptedData.authTag, 'hex'));

    let decrypted = decipher.update(encryptedData.encrypted, 'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return decrypted;
  }

  private async deriveKey(context: string): Promise<Buffer> {
    const baseKey = process.env.ENCRYPTION_KEY;
    const salt = Buffer.from(`${this.keyDerivationSalt}:${context}`);

    return new Promise((resolve, reject) => {
      crypto.pbkdf2(baseKey, salt, 100000, 32, 'sha256', (err, derivedKey) => {
        if (err) reject(err);
        else resolve(derivedKey);
      });
    });
  }
}
```

14.2 API Security Headers Configuration

```
// Comprehensive security headers
const securityHeaders = helmet({
  // Content Security Policy
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'unsafe-inline'", 'https://cdnjs.cloudflare.com'],
      styleSrc: ["'self'", "'unsafe-inline'", 'https://fonts.googleapis.com'],
      imgSrc: ["'self'", 'data:', 'https:'],
      connectSrc: ["'self'", 'wss:', 'https:'],
      fontSrc: ["'self'", 'https://fonts.gstatic.com'],
      objectSrc: ["'none'"],
      mediaSrc: ["'self'"],
      frameSrc: ["'none'"],
    },
  },

  // HTTP Strict Transport Security
  hsts: {
    maxAge: 31536000, // 1 year
    includeSubDomains: true,
    preload: true,
  },

  // X-Frame-Options
  frameguard: { action: 'deny' },

  // X-Content-Type-Options
  noSniff: true,

  // X-XSS-Protection
  xssFilter: true,

  // Referrer Policy
  referrerPolicy: { policy: 'strict-origin-when-cross-origin' },

  // Hide X-Powered-By header
  hidePoweredBy: true,
});
```

15. Monitoring and Alerting System

15.1 Comprehensive Monitoring Stack

```
// Monitoring Service Configuration
class MonitoringService {
  private prometheus = client.register;
  private metrics = {
    // Business Metrics
    orderProcessingTime: new client.Histogram({
      name: 'order_processing_duration_seconds',
      help: 'Time to process orders from creation to completion',
      labelNames: ['organization_id', 'order_type'],
      buckets: [0.1, 0.5, 1, 2, 5, 10, 30, 60],
    }),

    inventoryAccuracy: new client.Gauge({
      name: 'inventory_accuracy_percentage',
      help: 'Inventory accuracy percentage',
      labelNames: ['organization_id', 'item_category'],
    }),

    revenuePerHour: new client.Counter({
      name: 'revenue_total',
      help: 'Total revenue generated',
      labelNames: ['organization_id', 'payment_method'],
    }),

    // Technical Metrics
    httpRequestDuration: new client.Histogram({
      name: 'http_request_duration_seconds',
      help: 'HTTP request duration in seconds',
      labelNames: ['method', 'route', 'status_code'],
      buckets: [0.001, 0.01, 0.1, 0.5, 1, 2, 5],
    }),

    databaseQueryDuration: new client.Histogram({
      name: 'database_query_duration_seconds',
      help: 'Database query execution time',
      labelNames: ['query_type', 'table'],
      buckets: [0.001, 0.01, 0.05, 0.1, 0.5, 1],
    }),

    cacheHitRatio: new client.Gauge({
      name: 'cache_hit_ratio',
      help: 'Cache hit ratio percentage',
      labelNames: ['cache_type', 'cache_key_pattern'],
    }),
  };

  // Alert definitions
  private alertRules = {
    highErrorRate: {
      metric: 'http_request_errors_per_second',
      threshold: 10,
      duration: '5m',
      severity: 'critical',
      message: 'High error rate detected',
    },

    slowDatabaseQueries: {
      metric: 'database_query_duration_seconds',
      threshold: 1,
      percentile: 95,
      duration: '5m',
      severity: 'warning',
      message: 'Slow database queries detected',
    },

    lowInventoryAccuracy: {
      metric: 'inventory_accuracy_percentage',
      threshold: 85,
      operator: 'less_than',
      duration: '15m',
      severity: 'warning',
      message: 'Inventory accuracy below threshold',
    },

    orderProcessingDelay: {
      metric: 'order_processing_duration_seconds',
      threshold: 600, // 10 minutes
      percentile: 90,
      duration: '10m',
      severity: 'critical',
      message: 'Order processing delays detected',
    },
  };

  async recordBusinessMetric(metricName: string, value: number, labels: Record<string,
string>): Promise<void> {
    const metric = this.metrics[metricName];
    if (!metric) {
      this.logger.warn(`Unknown metric: ${metricName}`);
      return;
    }

    if (metric instanceof client.Histogram) {
      metric.observe(labels, value);
    } else if (metric instanceof client.Gauge) {
      metric.set(labels, value);
    } else if (metric instanceof client.Counter) {
      metric.inc(labels, value);
    }
  }

  async evaluateAlerts(): Promise<void> {
    for (const [alertName, rule] of Object.entries(this.alertRules)) {
      const isTriggered = await this.evaluateAlertRule(rule);

      if (isTriggered) {
        await this.triggerAlert(alertName, rule);
      }
    }
  }
  private async evaluateAlertRule(rule: AlertRule): Promise<boolean> {
    // Query Prometheus for metric values
    const query = this.buildPromQLQuery(rule);
    const result = await this.queryPrometheus(query);

    return this.checkThreshold(result, rule);
  }

  private async triggerAlert(alertName: string, rule: AlertRule): Promise<void> {
    const alert: Alert = {
      name: alertName,
      severity: rule.severity,
      message: rule.message,
      timestamp: new Date(),
      labels: rule.labels || {},
      annotations: {
        runbook_url: `https://docs.restaurant-crm.com/runbooks/${alertName}`,
        dashboard_url: `https://monitoring.restaurant-crm.com/d/${alertName}`,
      },
    };

    // Send to multiple channels
    await Promise.all([
      this.sendSlackAlert(alert),
      this.sendPagerDutyAlert(alert),
      this.sendEmailAlert(alert),
      this.logAlert(alert),
    ]);
  }
}
```

15.2 Business Intelligence Dashboard

```
// Real-time Business Intelligence Service
class BusinessIntelligenceService {
  async generateRealTimeDashboard(organizationId: string): Promise<DashboardData> {
    const timeRange = {
      start: moment().startOf('day'),
      end: moment(),
    };

    const [
      currentMetrics,
      historicalTrends,
      predictions,
      alerts,
    ] = await Promise.all([
      this.getCurrentMetrics(organizationId, timeRange),
      this.getHistoricalTrends(organizationId, timeRange),
      this.getAIPredictions(organizationId),
      this.getActiveAlerts(organizationId),
    ]);

    return {
      organizationId,
      lastUpdated: new Date(),
      currentMetrics: {
        revenue: {
          today: currentMetrics.revenue.today,
          target: currentMetrics.revenue.target,
          variance: currentMetrics.revenue.variance,
          trend: historicalTrends.revenue.trend,
        },
        orders: {
          count: currentMetrics.orders.count,
          averageValue: currentMetrics.orders.averageValue,
          completionRate: currentMetrics.orders.completionRate,
          processingTime: currentMetrics.orders.averageProcessingTime,
        },
        inventory: {
          accuracy: currentMetrics.inventory.accuracy,
          turnoverRate: currentMetrics.inventory.turnoverRate,
          stockOuts: currentMetrics.inventory.stockOuts,
          wasteRate: currentMetrics.inventory.wasteRate,
        },
        kitchen: {
          efficiency: currentMetrics.kitchen.efficiency,
          averageCookTime: currentMetrics.kitchen.averageCookTime,
          qualityScore: currentMetrics.kitchen.qualityScore,
          equipmentUtilization: currentMetrics.kitchen.equipmentUtilization,
        },
        financial: {
          grossProfit: currentMetrics.financial.grossProfit,
          netProfit: currentMetrics.financial.netProfit,
          foodCostRatio: currentMetrics.financial.foodCostRatio,
          laborCostRatio: currentMetrics.financial.laborCostRatio,
        },
      },
      predictions: {
        demand: predictions.demand,
        revenue: predictions.revenue,
        inventory: predictions.inventory,
        staffing: predictions.staffing,
      },
      alerts: alerts.map(alert => ({
        type: alert.type,
        severity: alert.severity,
        message: alert.message,
        timestamp: alert.timestamp,
        acknowledged: alert.acknowledged,
      })),
      recommendations: await this.generateRecommendations(organizationId, currentMetrics),
    };
  }

  private async generateRecommendations(
    organizationId: string,
    metrics: CurrentMetrics
  ): Promise<Recommendation[]> {
    const recommendations: Recommendation[] = [];

    // Menu optimization recommendations
    if (metrics.financial.foodCostRatio > 35) {
      recommendations.push({
        type: 'menu_optimization',
        priority: 'high',
        title: 'High Food Cost Ratio Detected',
        description: 'Your food cost ratio is above the recommended 35%. Consider reviewing
menu pricing or supplier costs.',
        actions: [
          'Review high-cost menu items',
          'Negotiate better supplier prices',
          'Consider portion size adjustments',
        ],
        estimatedImpact: {
          metric: 'food_cost_ratio',
          improvement: '5-8%',
          timeframe: '2-4 weeks',
        },
      });
    }

    // Inventory optimization
    if (metrics.inventory.accuracy < 90) {
      recommendations.push({
        type: 'inventory_management',
        priority: 'medium',
        title: 'Inventory Accuracy Below Target',
        description: 'Current inventory accuracy is below the 90% target. Improve stock control
processes.',
        actions: [
          'Implement cycle counting',
          'Train staff on proper stock procedures',
          'Review stock transfer validation process',
        ],
        estimatedImpact: {
          metric: 'inventory_accuracy',
          improvement: '10-15%',
          timeframe: '3-6 weeks',
        },
      });
    }

    // Kitchen efficiency recommendations
    if (metrics.kitchen.efficiency < 80) {
      recommendations.push({
        type: 'operational_efficiency',
        priority: 'medium',
        title: 'Kitchen Efficiency Improvement Opportunity',
        description: 'Kitchen efficiency is below optimal levels. Consider workflow
optimizations.',
        actions: [
          'Analyze kitchen workflow bottlenecks',
          'Optimize equipment placement',
          'Implement better task scheduling',
        ],
        estimatedImpact: {
          metric: 'kitchen_efficiency',
          improvement: '15-20%',
          timeframe: '4-8 weeks',
        },
      });
    }

    return recommendations;
  }
}
```

16. Final Architecture Summary

This comprehensive server-side system design for the Restaurant CRM Platform provides:

16.1 Architecture Highlights

- **Microservices Architecture**: 10+ specialized services with clear boundaries
- **Event-Driven Communication**: RabbitMQ-based messaging for loose coupling
- **Multi-tenant SaaS**: Organization-level data isolation and scaling
- **Real-time Capabilities**: WebSocket integration for live updates
- **AI/ML Integration**: TensorFlow and OpenAI for intelligent insights

- **High Availability**: Database clustering, Redis failover, auto-scaling
- **Security-First**: JWT authentication, RBAC, data encryption, audit logging
- **Observability**: Comprehensive monitoring, logging, and alerting

16.2 Scalability Features

- Horizontal scaling with Kubernetes HPA
- Database partitioning and read replicas
- Multi-level caching strategy
- CDN integration for static assets
- Event-driven architecture for loose coupling

16.3 Reliability & Performance

- 99.9% uptime SLA with health checks and failover
- <200ms API response times with optimized queries
- Automated backup and disaster recovery
- Performance monitoring and alerting
- Load testing and capacity planning

16.4 Security & Compliance

- Multi-factor authentication and RBAC
- Data encryption at rest and in transit
- PCI-DSS compliance for payments
- GDPR compliance for data protection
- Security headers and vulnerability scanning

This system design provides a robust, scalable, and maintainable foundation for the Restaurant CRM Platform that can handle the complex workflows and real-time requirements outlined in the PRD while ensuring security, reliability, and performance at scale.