Technical Document

Database Technical Documentation
System Overview
The system appears to be a Campaign Management Platform with integrated customer messaging capabilities. The database is implemented using MySQL with Prisma as the ORM.

Core Entities

1. User Management
User
- Primary entity for system users
- Authenticated via Google (OAuth)
- Role-based access control implementation
- Key fields:
  - `id`: Primary key
  - `googleId`: Unique identifier from Google Auth
  - `email`: Unique email address
  - `name`: User's full name
  - `roles`: Many-to-many relation with Role entity

Role & Access Control
- Implements RBAC (Role-Based Access Control)
- `WhitelistedEmail` system for controlling access
- Many-to-many relationship between Users and Roles
- Custom join table `UserRole` for role assignments

2. Campaign Management
Campaign
- Central entity for marketing campaigns

- Status lifecycle: DRAFT → ACTIVE → PAUSED → COMPLETED
- Comprehensive tracking of creation and updates
- Key features:
  - Budget management
  - Date range control (startDate, endDate)
  - Message templating
  - Audience targeting
  - Performance tracking

Campaign Statistics
- Tracks key performance metrics:
  - Impressions, clicks, conversions
  - Financial metrics: cost, CTR, CPC, CPA, ROI
  - Timestamped entries for trend analysis

3. Audience Management
Customer
- Stores customer profile and interaction data
- Tracks:
  - Contact information
  - Purchase history
  - Engagement metrics (visits, last visit)
  - Total spending

AudienceSegment
- Enables targeted campaign creation
- Dynamic filtering system
- Tracks audience size
- Links to campaigns via `CampaignAudienceSegment`

4. Messaging System
Message

- Handles campaign message delivery
- Status tracking: PENDING → SENT → DELIVERED/FAILED
- Comprehensive delivery tracking with receipts
- Links messages to both campaigns and customers

Database Relationships

One-to-Many Relationships
1. User → Campaign (as creator)
2. User → Campaign (as updater)
3. Campaign → CampaignStats
4. Customer → Order
5. Campaign → Message
6. Customer → Message

Many-to-Many Relationships
1. User ↔ Role (via UserRole)
2. Campaign ↔ AudienceSegment (via CampaignAudienceSegment)

Indexing Strategy
- Primary focus on foreign key relationships
- Performance optimization for:
  - Campaign status queries
  - Message status tracking
  - Customer email lookups
  - Order status and date-based queries

Audit and History Tracking
CampaignHistory
- Comprehensive change tracking
- Stores:
  - Old and new values

  - Update timestamp
  - User responsible for changes

Data Types and Constraints
- Appropriate use of Text fields for large content (messageTemplate, content)
- Float for financial calculations
- DateTime for temporal data
- Enums for status fields:
  - CampaignStatus
  - MessageStatus
  - OrderStatus

Performance Considerations
1. Indexed foreign keys for efficient joins
2. Composite indexes on frequently queried combinations
3. Text fields appropriately sized using @db.Text
4. Timestamp tracking for all major entities

Security Features
1. Role-based access control
2. Email whitelisting for access control
3. Audit trail for campaign changes
4. User action tracking

Recommendations for Optimization
1. Consider partitioning for:
   - CampaignStats (by date)
   - Messages (by status/date)
   - Orders (by status/date)
2. Implement archiving strategy for:
   - Completed campaigns

   - Delivered messages
   - Old campaign statistics
3. Consider adding:
   - Soft delete functionality
   - Cache tables for audience segment calculations
   - Message template versioning

API Technical Documentation

Overview
The API implements a RESTful architecture for a CRM system with campaign management, customer data handling, and audience segmentation capabilities. It uses Express.js with Passport for authentication and implements role-based access control.

Authentication
OAuth 2.0 with Google
- **GET** `/auth/google`
  - Initiates Google OAuth flow
  - Scope: email, profile
  - Forces account selection

- **GET** `/auth/google/callback`
  - Handles OAuth callback
  - Success: Redirects to `/dashboard`
  - Failure: Redirects to `/login`

Session Management
- **GET** `/logout`
  - Ends user session
  - Clears cookies
  - Returns success message

Core API Endpoints

Dashboard
- **GET** `/dashboard`
  - Returns user profile with roles
  - Authentication: Required

- Response includes:
  ```typescript
  {
    success: boolean
    user: {
      id: number
      googleId: string
      email: string
      name: string
      roles: Array<{id: number, name: string}>
    }
  }
  ```

Campaign Management
Base path: `/api/campaign`

Method	Endpoint	Access Roles	Description
POST	`/`	Admin, Manager	Create new campaign
GET	`/`	All authenticated	List campaigns
GET	`/:id`	All authenticated	Get campaign details
PUT	`/:id`	Admin, Manager	Update campaign
PATCH	`/:id/status`	Admin, Manager	Update campaign status
PUT	`/:id/stats`	Admin, Manager	Update campaign statistics
DELETE	`/:id`	Admin	Delete campaign

Messaging System
Base path: `/api/message`

Method	Endpoint	Access Roles	Description

POST	`/send`	Admin, Manager	Send campaign message
POST	`/delivery-status`	Admin, Manager	Update message delivery status
GET	`/campaign/:campaignId/stats`	Admin, Manager	Get campaign message statistics
GET	`/list`	Admin, Manager	List all messages

Data Ingestion
Base path: `/api/data-ingestion`

Customer Endpoints
Method	Endpoint	Access Roles	Description
POST	`/customers`	Admin, Manager	Ingest customer data
GET	`/customers`	Admin, Manager, Viewer	List all customers
GET	`/customers/metrics`	Admin, Manager	Get customer metrics
GET	`/customers/:id`	Admin, Manager, Viewer	Get customer details
PUT	`/customers/:id`	Admin, Manager	Update customer
DELETE	`/customers/:id`	Admin	Delete customer

Order Endpoints
Method	Endpoint	Access Roles	Description
POST	`/orders`	Admin, Manager	Ingest order data
GET	`/orders`	Admin, Manager, Viewer	List all orders
GET	`/orders/metrics`	Admin, Manager	Get order metrics
GET	`/orders/:id`	Admin, Manager, Viewer	Get order details
PUT	`/orders/:id`	Admin, Manager	Update order
DELETE	`/orders/:id`	Admin	Delete order

Audience Segmentation
Base path: `/api/audience-segmentation`

Method	Endpoint	Access Roles	Description
POST	`/segments`	Admin, Manager	Create audience segment
PUT	`/segments/:id`	Admin, Manager	Update segment
DELETE	`/segments/:id`	Admin, Manager	Delete segment
GET	`/segments`	All authenticated	List segments
GET	`/segments/:id`	All authenticated	Get segment details
POST	`/segments/:id/validate-size`	Admin, Manager	Validate segment size

Metrics
Base path: `/api/metrics`

Method	Endpoint	Description
GET	`/customer`	Get customer metrics
GET	`/order`	Get order metrics
POST	`/calculate/:type`	Calculate specific metric type

Authorization Levels

Role Hierarchy
1. **Admin**: Full system access
2. **Manager**: Campaign and customer management
3. **Analyst**: Read access to campaigns and analytics
4. **Viewer**: Basic read access

Security Features

1. **Authentication Middleware**
   - Session-based authentication
   - OAuth 2.0 with Google
   - Session cookie management

2. **Authorization Middleware**
   - Role-based access control (RBAC)
   - Endpoint-specific role requirements
   - Granular permission control

3. **Data Validation**
   - Request validation middleware
   - Campaign validation
   - Customer/Order data validation

Technical Implementation Details

Middleware Stack
1. Authentication check
2. Role authorization
3. Request validation
4. Route handling

Services Integration
- Redis for caching and queue management
- Prisma for database operations
- Express for routing
- Passport for authentication

Error Handling
- Consistent error response format
- Session management error handling

- Authentication failure handling
- Authorization failure handling

Performance Considerations

1. **Caching Strategy**
   - Redis integration for audience segments
   - Query result caching
   - Session storage

2. **Queue Management**
   - Audience segmentation processing
   - Message delivery queuing
   - Metrics calculation

3. **Database Optimization**
   - Prisma query optimization
   - Indexed lookups
   - Relationship eager loading

API Best Practices
1. RESTful resource naming
2. Consistent error responses
3. Proper HTTP method usage
4. Authentication/Authorization separation
5. Validation middleware
6. Queue-based processing for heavy operations

System Configuration Documentation

Architecture Overview
The system implements a distributed architecture with multiple services:
- MySQL Database (via Prisma)
- Redis Cache
- RabbitMQ Message Broker
- Google OAuth Authentication
- Express Web Server

Service Configurations

1. Database Configuration (MySQL/Prisma)
```typescript
class DatabaseConfig {
  private static prisma: PrismaClient | null = null;
  // Singleton pattern implementation
  // Connection management
  // Auto-reconnect handling
}
```

Features:
- Singleton pattern for connection management
- Automatic connection handling
- Error handling and logging
- Graceful disconnection support

2. Redis Cache Configuration
```typescript
class RedisConfig {
```

  private static client: Redis;
  // TLS support
  // Connection pooling
  // Ping verification
}

Features:
- TLS security enabled
- Connection pooling
- Health check via PING
- Environment-based configuration
- Password authentication

Environment Variables:
- `REDIS_HOST`: Redis server host
- `REDIS_PORT`: Redis server port
- `REDIS_PASSWORD`: Authentication password

3. RabbitMQ Message Broker Configuration
```typescript
class RabbitMQConfig {
  private static connection: Connection;
  private static channel: Channel;
  // AMQP connection management
  // Exchange and queue setup
}
```

Exchange Configuration:
Exchange Name	Type	Purpose

data-ingestion	direct	Customer and order data ingestion
campaign-exchange	direct	Campaign management operations
message-exchange	direct	Message delivery system
metrics-exchange	direct	Metrics processing

Queue Bindings:

1. **Data Ingestion:**
   - customer-queue → data-ingestion (customer)
   - order-queue → data-ingestion (order)
   - customer-update-queue → data-ingestion (customer.update)
   - customer-delete-queue → data-ingestion (customer.delete)
   - order-update-queue → data-ingestion (order.update)
   - order-delete-queue → data-ingestion (order.delete)

2. **Campaign Management:**
   - campaign-creation-queue → campaign-exchange (campaign.create)
   - campaign-stats-queue → campaign-exchange (campaign.stats)

3. **Messaging:**
   - message-queue → message-exchange (message.send)

4. **Metrics:**
   - customer-metrics-queue → metrics-exchange (customer.metrics)
   - order-metrics-queue → metrics-exchange (order.metrics)

Environment Variables:
- `RABBITMQ_DEFAULT_USER`: RabbitMQ username
- `RABBITMQ_DEFAULT_PASS`: RabbitMQ password
- `RABBITMQ_HOST`: RabbitMQ server host
- `RABBITMQ_VHOST`: Virtual host

4. Authentication Configuration (Google OAuth)
```typescript
passport.use(new GoogleStrategy({...}))
```

Features:
- Google OAuth 2.0 integration
- Role-based access control
- Email whitelisting
- Session management
- User serialization/deserialization

Configuration Settings:
- Callback URL:
`https://crm-app-xeno-1.onrender.com/auth/google/callback`
- Required scopes: email, profile
- Automatic role assignment
- Whitelisted email verification

Environment Variables:
- `GOOGLE_CLIENT_ID`: Google OAuth client ID
- `GOOGLE_CLIENT_SECRET`: Google OAuth client secret

System Integration Diagram

```mermaid
graph TD
    A[Express App] --> B[Auth Service]
    A --> C[Redis Cache]
    A --> D[RabbitMQ]
    A --> E[MySQL/Prisma]
```

    B --> F[Google OAuth]
    D --> G[Data Ingestion Queue]
    D --> H[Campaign Queue]
    D --> I[Message Queue]
    D --> J[Metrics Queue]

    C --> K[Session Cache]
    C --> L[Data Cache]

    E --> M[User Data]
    E --> N[Business Data]
```

## Security Considerations

1. **Database Security:**
   - Connection pooling
   - Prepared statements via Prisma
   - Automatic query sanitization

2. **Redis Security:**
   - TLS encryption
   - Password authentication
   - Protected mode enabled

3. **RabbitMQ Security:**
   - AMQPS (TLS) connection
   - Virtual host isolation
   - Queue-specific permissions

4. **OAuth Security:**
```

   - Secure callback URLs
   - Email whitelisting
   - Role-based access control
   - Session management

Deployment Considerations

1. **Environment Variables:**
   - Separate configurations for development/production
   - Sensitive data management
   - Service endpoints configuration

2. **Service Dependencies:**
   - Database initialization
   - Queue declaration
   - Exchange binding
   - Cache warming

3. **Health Checks:**
   - Database connection monitoring
   - Redis ping verification
   - RabbitMQ channel status
   - OAuth service availability

4. **Error Handling:**
   - Service reconnection logic
   - Graceful degradation
   - Error logging and monitoring

Monitoring Recommendations

1. **Service Health:**

   - Database connection status
   - Redis connection status
   - RabbitMQ channel status
   - Queue depths

2. **Performance Metrics:**
   - Database query times
   - Cache hit rates
   - Message processing rates
   - Authentication response times

3. **Error Tracking:**
   - Failed connections
   - Authentication failures
   - Queue processing errors
   - Cache misses

4. **Resource Usage:**
   - Database connections
   - Redis memory usage
   - RabbitMQ queue sizes
   - Session counts