Software Documentation

spsyn8lm\_rocklime\_dashboard - Version: 1.0.0  
Authors:

* Dhruv Verma @ROCKLIME

Prepared for: CM TRADING CO.

System Overview: The spsyn8lm\_rocklime\_dashboard is a dashboard application designed to streamline business operations, including product management, order processing, invoicing, and customer interactions. It employs a hybrid database architecture:

* MySQL Database: Manages structured data for entities like users, products, orders, invoices, customers, roles, etc, using Sequelize for ORM.
* MongoDB Database: Handles flexible or time-sensitive data, such as attendances, carts, comments, inventoryHistories, orderItems, otps, refreshTokens, and verificationTokens, using Mongoose for schema management.
* Product Code Generation: A custom script generates unique productCode values for products, ensuring consistency and uniqueness across the system.

Key features include:

* Role-based access control via roles and permissions.
* Hierarchical product categorization using brands, categories, and parentcategories.
* Order and invoice management with customer and address associations.
* Flexible cart and comment systems in MongoDB for enhanced scalability.
* Unique product code generation based on product attributes and brand information.

Database Configuration

* Host: 119.18.54.11
* Server Version: Percona Server 5.7.23-23 (GPL), Revision 500fcf5
* Operating System: Linux
* Default Character Set: utf8 with utf8\_unicode\_ci collation (some tables use utf8mb4 for broader Unicode support)
* Foreign Key Checks: Disabled during schema creation (FOREIGN\_KEY\_CHECKS=0)
* SQL Mode: NO\_AUTO\_VALUE\_ON\_ZERO to prevent automatic incrementing of zero values
* Time Zone: Set to +00:00 (UTC) during schema operation

Database Structure

The database contains multiple tables to manage users, products, orders, invoices, and related entities. Below is a detailed breakdown of each table, including its purpose, columns, constraints, and relationships.

1. addresses

Purpose: Stores address information for users or customers, linked to the users table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| addressId | CHAR(36) | Unique identifier for the address (UUID) | Primary Key |
| street | VARCHAR(255) | Street address | Nullable |
| city | VARCHAR(100) | City name | Nullable |
| state | VARCHAR(100) | State or region | Nullable |
| postalCode | VARCHAR(20) | Postal or ZIP code | Nullable |
| country | VARCHAR(100) | Country name | Nullable |
| createdAt | DATETIME | Timestamp of address creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| userId | CHAR(36) | References the user associated with the address | Foreign Key (users.userId), Nullable |

Constraints:

* Primary Key: addressId
* Foreign Key: userId references users(userId) with ON DELETE SET NULL and ON UPDATE CASCADE
* Index: userId for faster lookups

Engine: InnoDB  
Character Set: utf8mb4

1. brands

Purpose: Manages brand information for products.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier for the brand (UUID) | Primary Key |
| brandSlug | VARCHAR(255) | Unique slug for the brand | Unique |
| brandName | VARCHAR(100) | Name of the brand | Unique, Not Null |
| createdAt | DATETIME | Timestamp of brand creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |

Constraints:

* Primary Key: id
* Unique Keys: brandSlug, brandName

Engine: InnoDB  
Character Set: utf8mb4

1. brand\_parentcategories

Purpose: Links brands to parent categories, enabling hierarchical category management.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| name | VARCHAR(100) | Category name | Unique, Not Null |
| slug | VARCHAR(255) | Unique slug for the category | Unique, Not Null |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| brandId | CHAR(36) | References the associated brand | Foreign Key (brands.id), Nullable |
| parentCategoryId | CHAR(36) | References the parent category | Foreign Key (parentcategories.id), Nullable |

Constraints:

* Primary Key: id
* Unique Keys: name, slug, brandId (ensures one parent category per brand)
* Foreign Keys:
  + brandId references brands(id) with ON DELETE CASCADE and ON UPDATE CASCADE
  + parentCategoryId references parentcategories(id) with ON DELETE CASCADE and ON UPDATE CASCADE
* Index: parentCategoryId

Engine: InnoDB  
Character Set: utf8mb4 with utf8mb4\_unicode\_ci collation

1. brand\_parentcategory\_brands

Purpose: Junction table to manage many-to-many relationships between brands and parent categories.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| brandParentCategoryId | CHAR(36) | References the brand parent category | Primary Key (composite with brandId) |
| brandId | CHAR(36) | References the brand | Primary Key (composite) |

Constraints:

* Primary Key: Composite (brandParentCategoryId, brandId)
* Unique Key: Ensures no duplicate mappings

Engine: InnoDB  
Character Set: utf8 with utf8\_unicode\_ci collation

1. cart

Purpose: Stores user cart data, including items in JSON format.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier for the cart (UUID) | Primary Key |
| user\_id | CHAR(36) | References the user owning the cart | Foreign Key (users.userId), Not Null |
| created\_at | DATETIME | Timestamp of cart creation | Nullable |
| updated\_at | DATETIME | Timestamp of last update | Nullable |
| items | JSON | Stores cart items (products, quantities, etc.) | Not Null |

Constraints:

* Primary Key: id
* Foreign Key: user\_id references users(userId) with ON DELETE CASCADE and ON UPDATE CASCADE
* Index: user\_id

Engine: InnoDB  
Character Set: utf8mb4

1. categories

Purpose: Manages product categories, linked to brands and parent categories.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| categoryId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| name | VARCHAR(100) | Category name | Not Null |
| parentCategoryId | CHAR(36) | References the parent category | Foreign Key (parentcategories.id), Nullable |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| slug | VARCHAR(255) | Unique slug for the category | Not Null |
| brandId | CHAR(36) | References the associated brand | Foreign Key (brands.id), Nullable |

Constraints:

* Primary Key: categoryId
* Unique Key: name and brandId combination (ensures unique category names per brand)
* Foreign Keys:
  + parentCategoryId references parentcategories(id) with ON DELETE NO ACTION and ON UPDATE CASCADE
  + brandId references brands(id) with ON DELETE NO ACTION and ON UPDATE CASCADE
* Indexes: parentCategoryId, brandId

Engine: InnoDB  
Character Set: utf8mb4

1. comments

Purpose: Stores comments related to orders, linked to users and orders.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| orderId | CHAR(36) | References the associated order | Not Null |
| userId | CHAR(36) | References the user who made the comment | Not Null |
| comment | TEXT | Comment content | Not Null |
| createdAt | DATETIME | Timestamp of creation | Nullable |

Constraints:

* Primary Key: id

Engine: InnoDB  
Character Set: utf8 with utf8\_unicode\_ci collation

1. companies

Purpose: Manages company information, with support for parent-child company relationships.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| companyId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| name | VARCHAR(255) | Company name | Unique, Not Null |
| address | TEXT | Company address | Not Null |
| website | VARCHAR(255) | Company website URL | Nullable |
| createdDate | DATE | Date of company creation | Not Null |
| slug | VARCHAR(255) | Unique slug for the company | Unique, Not Null |
| parentCompanyId | CHAR(36) | References the parent company | Foreign Key (companies.companyId), Nullable |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |

Constraints:

* Primary Key: companyId
* Unique Keys: name, slug
* Foreign Key: parentCompanyId references companies(companyId) with ON DELETE SET NULL and ON UPDATE CASCADE
* Index: parentCompanyId

Engine: InnoDB  
Character Set: utf8mb4

1. customers

Purpose: Manages customer information, including financial and vendor details.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| customerId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| name | VARCHAR(100) | Customer name | Not Null |
| email | VARCHAR(100) | Customer email | Unique, Not Null |
| mobileNumber | VARCHAR(20) | Customer mobile number | Not Null |
| companyName | VARCHAR(150) | Associated company name | Nullable |
| address | JSON | Address details in JSON format | Nullable |
| quotations | JSON | Quotation details in JSON format | Nullable |
| invoices | JSON | Invoice details in JSON format | Nullable |
| isVendor | TINYINT(1) | Indicates if the customer is a vendor (0/1) | Default: 0 |
| vendorId | CHAR(36) | References the associated vendor | Foreign Key (vendors.id), Nullable |
| totalAmount | FLOAT | Total invoice amount | Default: 0 |
| paidAmount | FLOAT | Paid amount | Default: 0 |
| balance | FLOAT | Outstanding balance | Default: 0 |
| dueDate | DATETIME | Payment due date | Nullable |
| paymentMode | VARCHAR(50) | Payment method | Nullable |
| invoiceStatus | ENUM | Status of invoices (Paid, Overdue, etc.) | Nullable |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |

Constraints:

* Primary Key: customerId
* Unique Key: email
* Foreign Key: vendorId references vendors(id) with ON DELETE SET NULL and ON UPDATE CASCADE
* Index: vendorId

Engine: InnoDB  
Character Set: utf8mb4

1. invoices

Purpose: Manages invoice records, including billing and product details.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| invoiceId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| billTo | VARCHAR(255) | Billing entity name | Not Null |
| shipTo | CHAR(36) | References shipping address | Foreign Key (addresses.addressId), Nullable |
| amount | DECIMAL(10,2) | Invoice amount | Not Null |
| invoiceDate | DATE | Date of invoice issuance | Not Null |
| dueDate | DATE | Payment due date | Not Null |
| paymentMethod | JSON | Payment method details | Nullable |
| status | ENUM | Invoice status (paid, unpaid, etc.) | Not Null |
| products | JSON | Product details in JSON format | Not Null |
| signatureName | VARCHAR(255) | Name on the invoice signature | Default: 'CM TRADING CO' |
| createdBy | CHAR(36) | References the user who created the invoice | Foreign Key (users.userId), Nullable |
| quotationId | CHAR(36) | References the associated quotation | Foreign Key (quotations.quotationId), Nullable |
| invoiceNo | VARCHAR(255) | Unique invoice number | Unique, Nullable |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| customerId | CHAR(36) | References the associated customer | Foreign Key (customers.customerId), Nullable |

Constraints:

* Primary Key: invoiceId
* Unique Key: invoiceNo
* Foreign Keys:
  + shipTo references addresses(addressId) with ON DELETE SET NULL and ON UPDATE CASCADE
  + createdBy references users(userId) with ON DELETE SET NULL and ON UPDATE CASCADE
  + quotationId references quotations(quotationId) with ON DELETE SET NULL and ON UPDATE CASCADE
  + customerId references customers(customerId) with ON DELETE CASCADE and ON UPDATE CASCADE
* Indexes: shipTo, createdBy, quotationId, customerId

Engine: InnoDB  
Character Set: utf8mb4

1. keywords

Purpose: Stores keywords associated with categories for search or filtering purposes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| keyword | VARCHAR(100) | Keyword text | Unique, Not Null |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| categoryId | CHAR(36) | References the associated category | Foreign Key (categories.categoryId), Not Null |

Constraints:

* Primary Key: id
* Unique Key: keyword
* Foreign Key: categoryId references categories(categoryId) with ON DELETE CASCADE and ON UPDATE CASCADE
* Index: categoryId

Engine: InnoDB  
Character Set: utf8mb4

1. orders

Purpose: Manages order details, including status, priority, and assignments.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| title | VARCHAR(255) | Order title | Not Null |
| pipeline | JSON | Pipeline details in JSON format | Nullable |
| status | ENUM | Order status (CREATED, PREPARING, etc.) | Default: 'CREATED' |
| dueDate | DATE | Order due date | Nullable |
| followupDates | JSON | Follow-up dates in JSON format | Nullable |
| source | VARCHAR(255) | Order source | Nullable |
| priority | ENUM | Priority level (high, medium, low) | Default: 'medium' |
| description | TEXT | Order description | Nullable |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| createdFor | CHAR(36) | References the customer for the order | Foreign Key (customers.customerId), Not Null |
| createdBy | CHAR(36) | References the user who created the order | Foreign Key (users.userId), Not Null |
| assignedTo | CHAR(36) | References the team assigned to the order | Foreign Key (teams.id), Nullable |
| invoiceLink | VARCHAR(500) | Link to the associated invoice | Nullable |
| orderNo | VARCHAR(20) | Unique order number | Unique, Not Null |

Constraints:

* Primary Key: id
* Unique Key: orderNo
* Foreign Keys:
  + createdFor references customers(customerId) with ON DELETE CASCADE and ON UPDATE CASCADE
  + createdBy references users(userId) with ON DELETE CASCADE and ON UPDATE CASCADE
  + assignedTo references teams(id) with ON DELETE SET NULL and ON UPDATE CASCADE
* Indexes: createdFor, createdBy, assignedTo

Engine: InnoDB  
Character Set: utf8mb4

1. parentcategories

Purpose: Manages parent categories for organizing product categories.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| name | VARCHAR(100) | Parent category name | Unique, Not Null |
| slug | VARCHAR(255) | Unique slug for the parent category | Unique, Not Null |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| brandParentCategoryId | CHAR(36) | References the brand parent category | Foreign Key (brand\_parentcategories.id), Nullable |

Constraints:

* Primary Key: id
* Unique Keys: name, slug
* Foreign Key: brandParentCategoryId references brand\_parentcategories(id) with ON DELETE SET NULL and ON UPDATE CASCADE
* Index: brandParentCategoryId

Engine: InnoDB  
Character Set: utf8mb4

1. permissions

Purpose: Defines permissions for role-based access control.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| permissionId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| name | VARCHAR(255) | Human-readable permission name | Unique, Not Null |
| module | VARCHAR(255) | Module associated with the permission | Not Null |
| api | ENUM | API action (view, delete, write, edit, export) | Not Null |
| route | VARCHAR(500) | API route path | Not Null |

Constraints:

* Primary Key: permissionId
* Unique Key: name
* Index: module

Engine: InnoDB  
Character Set: utf8mb4

1. products

Purpose: Manages product details, including metadata and categorization.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| productId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| name | VARCHAR(255) | Product name | Not Null |
| product\_code | VARCHAR(255) | Unique product code | Unique, Not Null |
| quantity | INT | Available quantity | Not Null |
| discountType | ENUM | Discount type (percent, fixed) | Nullable |
| alert\_quantity | INT | Low stock alert threshold | Nullable |
| tax | DECIMAL(5,2) | Tax rate | Nullable |
| description | TEXT | Product description | Nullable |
| images | JSON | Product images in JSON format | Nullable |
| brandId | CHAR(36) | References the brand | Foreign Key (brands.id), Nullable |
| categoryId | CHAR(36) | References the category | Foreign Key (categories.categoryId), Nullable |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| isFeatured | TINYINT(1) | Indicates if the product is featured (0/1) | Default: 0 |
| productType | ENUM | Product type (tiles, sanitary) | Not Null |
| vendorId | CHAR(36) | References the vendor | Foreign Key (vendors.id), Nullable |
| brand\_parentcategoriesId | CHAR(36) | References the brand parent category | Foreign Key (brand\_parentcategories.id), Nullable |
| meta | JSON | Product metadata (key-value pairs) | Nullable |

Constraints:

* Primary Key: productId
* Unique Key: product\_code
* Foreign Keys:
  + brandId references brands(id) with ON DELETE SET NULL and ON UPDATE CASCADE
  + categoryId references categories(categoryId) with ON DELETE SET NULL and ON UPDATE CASCADE
  + vendorId references vendors(id) with ON DELETE SET NULL and ON UPDATE CASCADE
  + brand\_parentcategoriesId references brand\_parentcategories(id) with ON DELETE SET NULL and ON UPDATE CASCADE
* Indexes: brandId, categoryId, vendorId, brand\_parentcategoriesId

Engine: InnoDB  
Character Set: utf8mb4

1. product\_metas

Purpose: Stores metadata definitions for products (e.g., price, dimensions).

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| title | VARCHAR(255) | Metadata field label (e.g., Selling Price) | Not Null |
| slug | VARCHAR(255) | Unique slug for the metadata field | Unique, Nullable |
| fieldType | VARCHAR(255) | Data type (e.g., string, number, mm) | Not Null |
| unit | VARCHAR(255) | Unit of measurement (e.g., inch, pcs) | Nullable |
| createdAt | DATETIME | Timestamp of creation | Nullable |

Constraints:

* Primary Key: id
* Unique Key: slug

Engine: InnoDB  
Character Set: utf8 with utf8\_unicode\_ci collation

1. quotations

Purpose: Manages quotation records for customers.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| quotationId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| document\_title | VARCHAR(255) | Quotation title | Not Null |
| quotation\_date | DATE | Date of quotation issuance | Not Null |
| due\_date | DATE | Quotation due date | Not Null |
| reference\_number | VARCHAR(50) | Reference number | Nullable |
| include\_gst | TINYINT(1) | Indicates if GST is included (0/1) | Not Null |
| gst\_value | DECIMAL(10,2) | GST amount | Nullable |
| products | JSON | Product details in JSON format | Not Null |
| discountType | ENUM | Discount type (percent, fixed) | Nullable |
| roundOff | DECIMAL(10,2) | Round-off amount | Nullable |
| finalAmount | DECIMAL(10,2) | Final quotation amount | Not Null |
| signature\_name | VARCHAR(255) | Signature name | Nullable |
| signature\_image | TEXT | Signature image data | Nullable |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| customerId | CHAR(36) | References the customer | Foreign Key (customers.customerId), Not Null |
| createdBy | CHAR(36) | References the user who created the quotation | Foreign Key (users.userId), Nullable |
| shipTo | CHAR(36) | References the shipping address | Foreign Key (addresses.addressId), Nullable |

Constraints:

* Primary Key: quotationId
* Foreign Keys:
  + customerId references customers(customerId) with ON UPDATE CASCADE
  + createdBy references users(userId) with ON DELETE SET NULL and ON UPDATE CASCADE
  + shipTo references addresses(addressId) with ON DELETE SET NULL and ON UPDATE CASCADE
* Indexes: customerId, createdBy, shipTo

Engine: InnoDB  
Character Set: utf8mb4

1. rolepermissions

Purpose: Junction table for mapping roles to permissions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| roleId | CHAR(36) | References the role | Foreign Key (roles.roleId), Nullable |
| permissionId | CHAR(36) | References the permission | Foreign Key (permissions.permissionId), Nullable |

Constraints:

* Primary Key: id
* Unique Key: Composite (roleId, permissionId)
* Foreign Keys:
  + roleId references roles(roleId) with ON DELETE CASCADE and ON UPDATE CASCADE
  + permissionId references permissions(permissionId) with ON DELETE CASCADE and ON UPDATE CASCADE
* Indexes: permissionId

Engine: InnoDB  
Character Set: utf8mb4

1. roles

Purpose: Manages user roles for access control.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| roleId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| roleName | VARCHAR(100) | Role name | Unique, Not Null |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |

Constraints:

* Primary Key: roleId
* Unique Key: roleName

Engine: InnoDB  
Character Set: utf8mb4

1. signatures

Purpose: Stores user signature details for invoices or quotations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| signatureId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| signature\_name | VARCHAR(255) | Name associated with the signature | Not Null |
| signature\_image | VARCHAR(255) | Signature image path or data | Not Null |
| mark\_as\_default | TINYINT(1) | Indicates if the signature is default (0/1) | Not Null |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| userId | CHAR(36) | References the user | Foreign Key (users.userId), Nullable |

Constraints:

* Primary Key: signatureId
* Foreign Key: userId references users(userId) with ON DELETE SET NULL and ON UPDATE CASCADE
* Index: userId

Engine: InnoDB  
Character Set: utf8mb4

1. teammembers

Purpose: Manages team memberships, linking users to teams and roles.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| teamId | CHAR(36) | References the team | Foreign Key (teams.id), Not Null |
| userId | CHAR(36) | References the user | Not Null |
| userName | VARCHAR(255) | User name | Not Null |
| roleId | VARCHAR(255) | Role identifier | Not Null |
| roleName | VARCHAR(255) | Role name | Not Null |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |

Constraints:

* Primary Key: id
* Foreign Key: teamId references teams(id) with ON DELETE CASCADE and ON UPDATE CASCADE
* Index: teamId

Engine: InnoDB  
Character Set: utf8mb4 with utf8mb4\_bin collation

1. teams

Purpose: Manages teams, including admin details.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| adminId | CHAR(36) | References the admin user | Not Null |
| adminName | VARCHAR(255) | Admin name | Not Null |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| teamName | VARCHAR(255) | Team name | Not Null |

Constraints:

* Primary Key: id
* Unique Key: id

Engine: InnoDB  
Character Set: utf8mb4

24. users

Purpose: Manages user accounts and their details.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| userId | CHAR(36) | Unique identifier (UUID) | Primary Key |
| username | VARCHAR(50) | Unique username | Unique, Not Null |
| name | VARCHAR(100) | User full name | Nullable |
| email | VARCHAR(100) | User email | Unique, Not Null |
| mobileNumber | VARCHAR(20) | User mobile number | Nullable |
| roles | VARCHAR(255) | User roles (default: 'USERS') | Default: 'USERS' |
| status | ENUM | User status (active, inactive, restricted) | Default: 'inactive', Not Null |
| password | VARCHAR(255) | Hashed password | Not Null |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |
| dateOfBirth | DATE | User date of birth | Nullable |
| shiftFrom | TIME | Shift start time | Nullable |
| shiftTo | TIME | Shift end time | Nullable |
| bloodGroup | ENUM | Blood group (A+, A-, etc.) | Nullable |
| addressId | CHAR(36) | References the user’s address | Foreign Key (addresses.addressId), Nullable |
| emergencyNumber | VARCHAR(20) | Emergency contact number | Nullable |
| roleId | CHAR(36) | References the user’s role | Foreign Key (roles.roleId), Nullable |
| isEmailVerified | TINYINT(1) | Indicates if email is verified (0/1) | Default: 0, Not Null |

Constraints:

* Primary Key: userId
* Unique Keys: username, email
* Foreign Keys:
  + addressId references addresses(addressId) with ON DELETE SET NULL and ON UPDATE CASCADE
  + roleId references roles(roleId) with ON UPDATE CASCADE
* Indexes: addressId, roleId

Engine: InnoDB  
Character Set: utf8mb4

25. vendors

Purpose: Manages vendor information, linked to brands.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Description** | **Constraints** |
| id | CHAR(36) | Unique identifier (UUID) | Primary Key |
| vendorId | VARCHAR(255) | Unique vendor identifier | Unique, Not Null |
| vendorName | VARCHAR(255) | Vendor name | Not Null |
| brandId | CHAR(36) | References the brand | Foreign Key (brands.id), Nullable |
| brandSlug | VARCHAR(255) | References the brand slug | Foreign Key (brands.brandSlug), Nullable |
| createdAt | DATETIME | Timestamp of creation | Not Null |
| updatedAt | DATETIME | Timestamp of last update | Not Null |

Constraints:

* Primary Key: id
* Unique Key: vendorId
* Foreign Keys:
  + brandId references brands(id) with ON DELETE CASCADE and ON UPDATE CASCADE
  + brandSlug references brands(brandSlug)
* Indexes: brandId, brandSlug

Engine: InnoDB  
Character Set: utf8mb4

MongoDB Collections

Below is a detailed breakdown of each MongoDB schema/collection, including its purpose, fields, indexes, and relationships with the MySQL database.

1. attendances

Purpose: Tracks user attendance, including clock-in and clock-out times, with a unique record per user per day.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| userId | String | References the user (matches MySQL users.userId) | Required, Indexed |
| status | String | Attendance status (absent, present) | Enum: ["absent", "present"], Default: "absent" |
| clockIn | Date | Clock-in timestamp | Default: null |
| clockOut | Date | Clock-out timestamp | Default: null |
| date | Date | Date of the attendance record | Default: Date.now, Indexed |
| createdAt | Date | Timestamp of record creation | Auto-generated (via timestamps: true) |
| updatedAt | Date | Timestamp of last update | Auto-generated (via timestamps: true) |

Indexes:

* userId: For faster queries by user.
* date: For date-based queries.
* Compound index: { userId: 1, date: 1 } (unique) to ensure one attendance record per user per day.

Relationships:

* userId references users(userId) in the MySQL database, establishing a link between MongoDB attendance records and MySQL user records.

Collection Name: attendances

1. carts

Purpose: Manages user or customer shopping carts, with embedded cart items.Sub-Schema: CartItemSchema

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| productId | String | References the product (matches MySQL products.productId) | Required |
| name | String | Product name (denormalized for convenience) | Optional |
| price | Number | Product price | Optional |
| quantity | Number | Quantity of the product | Default: 1 |
| discount | Number | Discount amount or percentage | Default: 0 |
| tax | Number | Tax amount or percentage | Default: 0 |
| total | Number | Total cost for the item | Optional |

Indexes: None explicitly defined, but Mongoose’s \_id is indexed by default.

Relationships:

* customerId references customers(customerId) in MySQL (optional, as carts may belong to users without a customer record).
* userId references users(userId) in MySQL.
* productId (in CartItemSchema) references products(productId) in MySQL.

Collection Name: carts

1. comments

Purpose: Stores comments on resources (e.g., orders, products, customers), with a limit of three comments per user per resource.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| resourceId | String | References the resource (e.g., MySQL orders.id, products.productId) | Required, Indexed |
| resourceType | String | Type of resource (Order, Product, Customer) | Required, Enum: ["Order", "Product", "Customer"], Indexed |
| userId | String | References the user (matches MySQL users.userId) | Required, Indexed, Ref: "User" (if MongoDB User model exists) |
| comment | String | Comment text | Required, Trimmed, Min: 1, Max: 1000 chars |
| createdAt | Date | Timestamp of creation | Default: Date.now |

Indexes:

* resourceId: For faster queries by resource.
* resourceType: For filtering by resource type.
* userId: For user-based queries.
* Compound index: { resourceId: 1, resourceType: 1, userId: 1 } for optimizing comment limit checks.

Static Methods:

* hasReachedCommentLimit(resourceId, resourceType, userId): Checks if a user has reached the limit of three comments for a specific resource.

Relationships:

* userId references users(userId) in MySQL.
* resourceId references:
  + orders(id) in MySQL for resourceType: Order.
  + products(productId) in MySQL for resourceType: Product.
  + customers(customerId) in MySQL for resourceType: Customer.

Collection Name: comments

1. contacts

Purpose: Stores contact form submissions for customer inquiries.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| firstName | String | First name of the contact | Required, Trimmed |
| lastName | String | Last name of the contact | Optional, Trimmed |
| email | String | Contact email | Required, Trimmed, Email format |
| phone | String | Contact phone number | Optional, Trimmed |
| message | String | Contact message | Required, Trimmed |
| createdAt | Date | Timestamp of submission | Default: Date.now |
| responded | Boolean | Indicates if the inquiry was responded to | Default: false |
| replyMessage | String | Response message | Optional, Trimmed |
| replyDate | Date | Timestamp of response | Optional |

Indexes: None explicitly defined, but Mongoose’s \_id is indexed by default.Relationships: No direct relationships with MySQL tables, but email may correspond to customers.email or users.email for integration purposes.Collection Name: contacts  
Notes: This collection is likely used for a contact form feature, independent of the MySQL database but potentially linked via email matching.

1. inventoryHistories

Purpose: Tracks inventory changes for products (additions or removals).Sub-Schema: History Entry

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| quantity | Number | Quantity changed | Required |
| action | String | Action type (add-stock, remove-stock) | Required, Enum: ["add-stock", "remove-stock"] |
| timestamp | Date | Timestamp of the change | Default: Date.now |

Indexes:

* productId: Unique index to ensure one history document per product.

Relationships:

* productId references products(productId) in MySQL.

Collection Name: inventoryHistories

1. orderItems

Purpose: Stores items associated with orders, replacing the MySQL orders.pipeline JSON field with a structured schema.Sub-Schema: Order Item

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| productId | String | References the product (matches MySQL products.productId) | Required |
| quantity | Number | Quantity ordered | Required |
| discount | Number | Discount amount or percentage | Default: 0 |
| tax | Number | Tax amount or percentage | Default: 0 |
| total | Number | Total cost for the item | Required |

Collection Name: orderItems

1. otps

Purpose: Stores one-time passwords (OTPs) for user verification, with a 5-minute expiration.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| email | String | Email associated with the OTP | Required |
| code | String | OTP code | Required |
| createdAt | Date | Timestamp of creation | Default: Date.now, Expires: 300s (5 mins) |
| updatedAt | Date | Timestamp of last update | Auto-generated (via timestamps: true) |

Indexes:

* TTL index on createdAt to automatically delete documents after 5 minutes.

Relationships:

* email may correspond to users.email or customers.email in MySQL for verification purposes.

Collection Name: otps

1. refreshTokens

Purpose: Stores refresh tokens for user authentication.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| userId | String | References the user (matches MySQL users.userId) | Required, Indexed |
| token | String | Refresh token | Required, Indexed |
| expiresAt | Date | Token expiration date | Required, TTL index |
| createdAt | Date | Timestamp of creation | Default: Date.now |

Indexes:

* userId: For user-based queries.
* token: For token validation.
* TTL index on expiresAt to automatically delete expired tokens.

Relationships:

* userId references users(userId) in MySQL.

Collection Name: refreshTokens  
Notes: Supports JWT-based authentication by storing refresh tokens with automatic expiration.

9. verificationTokensPurpose: Stores tokens for email verification, with expiration.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Constraints** |
| userId | String | References the user (matches MySQL users.userId) | Required, Indexed |
| token | String | Verification token | Required, Indexed |
| email | String | Email being verified | Required |
| isVerified | Boolean | Indicates if the email is verified | Default: false |
| expiresAt | Date | Token expiration date | Required, TTL index |
| createdAt | Date | Timestamp of creation | Default: Date.now |

Indexes:

* userId: For user-based queries.
* token: For token validation.
* TTL index on expiresAt to automatically delete expired tokens.

Relationships:

* userId references users(userId) in MySQL.
* email corresponds to users.email in MySQL.

Collection Name: verificationTokens

MySQL Database Relationships (Sequelize)

The Sequelize setup code defines relationships between MySQL tables using associations like belongsTo, hasMany, and belongsToMany. These relationships are enforced through foreign keys and junction tables, ensuring data integrity and enabling efficient querying. Below, each relationship is described with its purpose, cardinality, constraints, and MongoDB integration where applicable.

* 1. User <-> Address

Purpose: Links users to their addresses, allowing each user to have multiple addresses, while each address is associated with one user.

Cardinality:

* One-to-Many: A User can have many Addresses, but an Address belongs to one User.

Sequelize Definition:

Address.belongsTo(User, { foreignKey: "userId", as: "users" });

User.hasMany(Address, { foreignKey: "userId", as: "addresses" });

MySQL Schema Details:

* Foreign Key: addresses.userId references users.userId.
* Constraints: ON DELETE SET NULL, ON UPDATE CASCADE (if a user is deleted, their addresses’ userId is set to NULL; updates to userId cascade).
* Alias: users (for Address → User), addresses (for User → Address).

MongoDB Integration: None directly, as addresses are stored in MySQL. However, MongoDB’s contacts collection (with email) may indirectly relate to users.email for contact form submissions.

* 1. Role <-> Permission (via RolePermission)

Purpose: Manages role-based access control by linking roles to permissions through a junction table (rolepermissions).

Cardinality:

* Many-to-Many: A Role can have multiple Permissions, and a Permission can be assigned to multiple Roles.

Sequelize Definition:

Role.belongsToMany(Permission, {

through: RolePermission,

foreignKey: "roleId",

otherKey: "permissionId",

as: "permissions",

});

Permission.belongsToMany(Role, {

through: RolePermission,

foreignKey: "permissionId",

otherKey: "roleId",

as: "roles",

});

RolePermission.belongsTo(Role, { foreignKey: "roleId", as: "roles" });

RolePermission.belongsTo(Permission, { foreignKey: "permissionId", as: "permissions" });

Role.hasMany(RolePermission, { foreignKey: "roleId", as: "rolepermissions" });

Permission.hasMany(RolePermission, { foreignKey: "permissionId", as: "rolepermissions" });

MySQL Schema Details:

* Junction Table: rolepermissions with columns id, roleId, permissionId.
* Foreign Keys:
  + rolepermissions.roleId references roles.roleId (ON DELETE CASCADE, ON UPDATE CASCADE).
  + rolepermissions.permissionId references permissions.permissionId (ON DELETE CASCADE, ON UPDATE CASCADE).
* Constraints: Unique composite key on roleId and permissionId to prevent duplicate mappings.
* Aliases: permissions (for Role → Permission), roles (for Permission → Role), rolepermissions (for Role/Permission → RolePermission).

MongoDB Integration: None, as role-based access control is managed entirely in MySQL.

* 1. User <-> Role

Purpose: Assigns roles to users to determine their permissions.

Cardinality:

* Many-to-One: A User belongs to one Role, but a Role can be assigned to many Users.

Sequelize Definition:

Role.hasMany(User, { foreignKey: "roleId", as: "users" });

User.belongsTo(Role, { foreignKey: "roleId", as: "role" });

MySQL Schema Details:

* Foreign Key: users.roleId references roles.roleId (ON UPDATE CASCADE).
* Alias: users (for Role → User), role (for User → Role).

MongoDB Integration: Indirectly affects MongoDB collections like attendances, comments, carts, refreshTokens, and verificationTokens, which reference users.userId and may use role-based logic in the application layer.

* 1. User <-> Order (createdBy)

Purpose: Tracks which user created an order.Cardinality:

* One-to-Many: A User can create many Orders, but an Order is created by one User.

Sequelize Definition:

User.hasMany(Order, { foreignKey: "createdBy", as: "users" });

Order.belongsTo(User, { foreignKey: "createdBy", as: "users" });

MySQL Schema Details:

* Foreign Key: orders.createdBy references users.userId (ON DELETE CASCADE, ON UPDATE CASCADE).
* Alias: users (bidirectional).

MongoDB Integration:

* orderItems.orderId in MongoDB references orders.id in MySQL, indirectly linking to users.userId via orders.createdBy.
* comments.resourceId (with resourceType: Order) in MongoDB may reference orders.id, connecting comments to the user who created the order.
  1. Customer <-> Order (createdFor)

Purpose: Links orders to the customer they were created for.

Cardinality:

* One-to-Many: A Customer can have many Orders, but an Order is created for one Customer.

Sequelize Definition:

Customer.hasMany(Order, { foreignKey: "createdFor", as: "customers" });

Order.belongsTo(Customer, { foreignKey: "createdFor", as: "customers" });

MySQL Schema Details:

* Foreign Key: orders.createdFor references customers.customerId (ON DELETE CASCADE, ON UPDATE CASCADE).
* Alias: customers (bidirectional).

MongoDB Integration:

* orderItems.orderId in MongoDB references orders.id, indirectly linking to customers.customerId via orders.createdFor.
* comments.resourceId (with resourceType: Customer) in MongoDB references customers.customerId.
  1. User <-> Invoice (createdBy)

Purpose: Tracks which user created an invoice.Cardinality:

* One-to-Many: A User can create many Invoices, but an Invoice is created by one User.

Sequelize Definition:

Invoice.belongsTo(User, { foreignKey: "createdBy" });

MySQL Schema Details:

* Foreign Key: invoices.createdBy references users.userId (ON DELETE SET NULL, ON UPDATE CASCADE).
* Alias: None specified.

MongoDB Integration: None directly, as invoices are managed in MySQL.

* 1. User <-> Quotation (createdBy)

Purpose: Tracks which user created a quotation.

Cardinality:

* One-to-Many: A User can create many Quotations, but a Quotation is created by one User.

Sequelize Definition:

User.hasMany(Quotation, { foreignKey: "createdBy" });

Quotation.belongsTo(User, { foreignKey: "createdBy", as: "users" });

MySQL Schema Details:

* Foreign Key: quotations.createdBy references users.userId (ON DELETE SET NULL, ON UPDATE CASCADE).
* Alias: users (for Quotation → User).

MongoDB Integration: None directly, as quotations are managed in MySQL.Notes:

* Ensures traceability of quotation creation.
* The users alias facilitates queries joining Quotation to User.
  1. User <-> Signature

Purpose: Links signatures to users for use in invoices or quotations.

Cardinality:

* One-to-Many: A User can have many Signatures, but a Signature belongs to one User.

Sequelize Definition:

Signature.belongsTo(User, { foreignKey: "userId" });

MySQL Schema Details:

* Foreign Key: signatures.userId references users.userId (ON DELETE SET NULL, ON UPDATE CASCADE).
* Alias: None specified.

MongoDB Integration: None directly, as signatures are managed in MySQL.Notes:

* The hasMany counterpart is not defined, possibly to limit query scope.
* Used to store user-specific signatures for documents.
  1. Product <-> Brand, Category, Vendor, BrandParentCategory, ProductMeta

Purpose: Associates products with their brand, category, vendor, brand parent category, and metadata for categorization and inventory management.

Cardinality:

* Many-to-One (for each):
  + A Product belongs to one Brand, but a Brand can have many Products.
  + A Product belongs to one Category, but a Category can have many Products.
  + A Product belongs to one Vendor, but a Vendor can have many Products.
  + A Product belongs to one BrandParentCategory, but a BrandParentCategory can have many Products.
  + A Product is linked to ProductMeta (though the relationship is non-standard due to constraints: false).

Sequelize Definition:

Product.belongsTo(Brand, { foreignKey: "brandId", as: "brand" });

Product.belongsTo(Category, { foreignKey: "categoryId", as: "categories" });

Product.belongsTo(Vendor, { foreignKey: "vendorId", as: "vendors" });

Product.belongsTo(BrandParentCategory, {

foreignKey: "brand\_parentcategoriesId",

as: "brand\_parentcategories",

});

Product.belongsTo(ProductMeta, {

foreignKey: "meta",

as: "product\_metas",

constraints: false,

});

MySQL Schema Details:

* Foreign Keys:
  + products.brandId references brands.id (ON DELETE SET NULL, ON UPDATE CASCADE).
  + products.categoryId references categories.categoryId (ON DELETE SET NULL, ON UPDATE CASCADE).
  + products.vendorId references vendors.id (ON DELETE SET NULL, ON UPDATE CASCADE).
  + products.brand\_parentcategoriesId references brand\_parentcategories.id (ON DELETE SET NULL, ON UPDATE CASCADE).
  + products.meta (JSON field, not a true foreign key; constraints: false indicates no referential integrity enforcement).
* Aliases: brand, categories, vendors, brand\_parentcategories, product\_metas.

MongoDB Integration:

* carts.items.productId, orderItems.items.productId, and inventoryHistories.productId in MongoDB reference products.productId in MySQL.
* comments.resourceId (with resourceType: Product) references products.productId.
  1. Customer <-> Vendor

Purpose: Links customers to vendors, indicating if a customer is also a vendor.Cardinality:

* Many-to-One: A Customer can be associated with one Vendor, but a Vendor can have many Customers.

Sequelize Definition:

Customer.belongsTo(Vendor, { foreignKey: "vendorId", as: "vendors" });

MySQL Schema Details:

* Foreign Key: customers.vendorId references vendors.id (ON DELETE SET NULL, ON UPDATE CASCADE).
* Alias: vendors.

MongoDB Integration: None directly, but carts.customerId in MongoDB references customers.customerId, which may indirectly relate to vendors.id.

* 1. Customer <-> Quotation

Purpose: Links quotations to the customer they are issued for.

Cardinality:

* One-to-Many: A Customer can have many Quotations, but a Quotation belongs to one Customer.

Sequelize Definition:

Customer.hasMany(Quotation, { foreignKey: "customerId", as: "customerQuotations" });

Quotation.belongsTo(Customer, { foreignKey: "customerId", as: "customers" });

MySQL Schema Details:

* Foreign Key: quotations.customerId references customers.customerId (ON UPDATE CASCADE).
* Alias: customerQuotations (for Customer → Quotation), customers (for Quotation → Customer).

MongoDB Integration: comments.resourceId (with resourceType: Customer) references customers.customerId, indirectly linking to quotations.

* 1. Customer <-> Invoice

Purpose: Links invoices to the customer they are issued for.Cardinality:

* One-to-Many: A Customer can have many Invoices, but an Invoice belongs to one Customer.

Sequelize Definition:

Customer.hasMany(Invoice, { foreignKey: "customerId", onDelete: "CASCADE" });

Invoice.belongsTo(Customer, { foreignKey: "customerId" });

MySQL Schema Details:

* Foreign Key: invoices.customerId references customers.customerId (ON DELETE CASCADE, ON UPDATE CASCADE).
* Alias: None specified.

MongoDB Integration: comments.resourceId (with resourceType: Customer) references customers.customerId, indirectly linking to invoices.Notes:

* The ON DELETE CASCADE ensures invoices are deleted if the customer is removed, maintaining data consistency.
  1. Invoice <-> Address (shipTo)

Purpose: Links invoices to their shipping address.Cardinality:

* Many-to-One: An Invoice belongs to one Address (for shipping), but an Address can be used by many Invoices.

Sequelize Definition:

Invoice.belongsTo(Address, { foreignKey: "shipTo" });

MySQL Schema Details:

* Foreign Key: invoices.shipTo references addresses.addressId (ON DELETE SET NULL, ON UPDATE CASCADE).
* Alias: None specified.

MongoDB Integration: None directly.

* 1. Invoice <-> Quotation

Purpose: Links an invoice to its originating quotation, if applicable.Cardinality:

* One-to-One: An Invoice can be linked to one Quotation, and a Quotation can have one Invoice.

Sequelize Definition:

Invoice.belongsTo(Quotation, { foreignKey: "quotationId", allowNull: true });

Quotation.hasOne(Invoice, { foreignKey: "quotationId" });

MySQL Schema Details:

* Foreign Key: invoices.quotationId references quotations.quotationId (ON DELETE SET NULL, ON UPDATE CASCADE).
* Alias: None specified.

MongoDB Integration: None directly.

* 1. Quotation <-> Address (shipTo)

Purpose: Links quotations to their shipping address.Cardinality:

* Many-to-One: A Quotation belongs to one Address (for shipping), but an Address can be used by many Quotations.

Sequelize Definition:

Quotation.belongsTo(Address, { foreignKey: "shipTo", as: "shippingAddress" });

MySQL Schema Details:

* Foreign Key: quotations.shipTo references addresses.addressId (ON DELETE SET NULL, ON UPDATE CASCADE).
* Alias: shippingAddress.

MongoDB Integration: None directly.

* 1. Brand <-> Vendor

Purpose: Associates vendors with their brands.

Cardinality:

* One-to-Many: A Brand can have many Vendors, but a Vendor belongs to one Brand.

Sequelize Definition:

Brand.hasMany(Vendor, { foreignKey: "brandId" });

Vendor.belongsTo(Brand, { foreignKey: "brandId" });

MySQL Schema Details:

* Foreign Key: vendors.brandId references brands.id (ON DELETE CASCADE, ON UPDATE CASCADE).
* Alias: None specified.
* Note: The commented-out brandSlug associations are redundant and not used.

MongoDB Integration: carts.items.productId and orderItems.items.productId reference products.productId, which links to vendors.id via products.vendorId.

* 1. Brand <-> ParentCategory (via BrandParentCategory)

Purpose: Manages the many-to-many relationship between brands and parent categories.Cardinality:

* Many-to-Many: A Brand can be associated with multiple ParentCategories, and a ParentCategory can be linked to multiple Brands.

Sequelize Definition:

Brand.belongsToMany(ParentCategory, {

through: BrandParentCategory,

foreignKey: "brandId",

otherKey: "parentCategoryId",

as: "parentcategories",

});

ParentCategory.belongsToMany(Brand, {

through: BrandParentCategory,

foreignKey: "parentCategoryId",

otherKey: "brandId",

as: "brands",

});

MySQL Schema Details:

* Junction Table: brand\_parentcategories with columns id, brandId, parentCategoryId.
* Foreign Keys:
  + brand\_parentcategories.brandId references brands.id (ON DELETE CASCADE, ON UPDATE CASCADE).
  + brand\_parentcategories.parentCategoryId references parentcategories.id (ON DELETE CASCADE, ON UPDATE CASCADE).
* Aliases: parentcategories (for Brand → ParentCategory), brands (for ParentCategory → Brand).

MongoDB Integration: products.brand\_parentcategoriesId links to brand\_parentcategories.id, indirectly connecting to MongoDB’s carts and orderItems via products.productId.

* 1. BrandParentCategory <-> Brand (via BrandParentCategoryBrand)

Purpose: Manages additional mappings between brands and brand parent categories.Cardinality:

* Many-to-Many: A BrandParentCategory can be linked to multiple Brands, and a Brand can be associated with multiple BrandParentCategories.

Sequelize Definition:

BrandParentCategory.belongsToMany(Brand, {

through: BrandParentCategoryBrand,

foreignKey: "brandParentCategoryId",

otherKey: "brandId",

as: "brands",

});

Brand.belongsToMany(BrandParentCategory, {

through: BrandParentCategoryBrand,

foreignKey: "brandId",

otherKey: "brandParentCategoryId",

as: "brandParentCategories",

});

MySQL Schema Details:

* Junction Table: brand\_parentcategory\_brands with columns brandParentCategoryId, brandId.
* Constraints: Composite primary key (brandParentCategoryId, brandId) with a unique constraint to prevent duplicates.
* Aliases: brands (for BrandParentCategory → Brand), brandParentCategories (for Brand → BrandParentCategory).

MongoDB Integration: Indirectly via products.brand\_parentcategoriesId.

19. BrandParentCategory <-> ParentCategoryPurpose: Links brand parent categories to parent categories for hierarchical organization.Cardinality:

* One-to-Many: A BrandParentCategory can have many ParentCategories, but a ParentCategory belongs to one BrandParentCategory.

Sequelize Definition:

BrandParentCategory.hasMany(ParentCategory, {

foreignKey: "brandParentCategoryId",

as: "parentCategories",

});

ParentCategory.belongsTo(BrandParentCategory, {

foreignKey: "brandParentCategoryId",

as: "brandParentCategory",

});

MySQL Schema Details:

* Foreign Key: parentcategories.brandParentCategoryId references brand\_parentcategories.id (ON DELETE SET NULL, ON UPDATE CASCADE).
* Aliases: parentCategories (for BrandParentCategory → ParentCategory), brandParentCategory (for ParentCategory → BrandParentCategory).

MongoDB Integration: Indirectly via products.brand\_parentcategoriesId.Notes:

* Supports complex category hierarchies for products.
  1. Category <-> Brand, ParentCategory

Purpose: Associates categories with brands and parent categories for product organization.Cardinality:

* Many-to-One (for each):
  + A Category belongs to one Brand, but a Brand can have many Categories.
  + A Category belongs to one ParentCategory, but a ParentCategory can have many Categories.

Sequelize Definition:

Category.belongsTo(Brand, { foreignKey: "brandId", as: "brand" });

Category.belongsTo(ParentCategory, { foreignKey: "parentCategoryId", as: "parentcategories" });

Brand.hasMany(Category, { foreignKey: "brandId", as: "categories" });

ParentCategory.hasMany(Category, { foreignKey: "parentCategoryId", as: "categories" });

MySQL Schema Details:

* Foreign Keys:
  + categories.brandId references brands.id (ON DELETE NO ACTION, ON UPDATE CASCADE).
  + categories.parentCategoryId references parentcategories.id (ON DELETE NO ACTION, ON UPDATE CASCADE).
* Aliases: brand, parentcategories, categories.

MongoDB Integration:

* keywords.categoryId in MySQL references categories.categoryId, and products.categoryId links to categories.categoryId, which connects to MongoDB’s carts and orderItems via products.productId.
  1. Team <-> Order

Purpose: Assigns orders to teams for processing.Cardinality:

* One-to-Many: A Team can have many Orders, but an Order is assigned to one Team.

Sequelize Definition:

Team.hasMany(Order, { foreignKey: "assignedTo", as: "team" });

Order.belongsTo(Team, { foreignKey: "assignedTo", as: "team" });

MySQL Schema Details:

* Foreign Key: orders.assignedTo references teams.id (ON DELETE SET NULL, ON UPDATE CASCADE).
* Alias: team (bidirectional).

MongoDB Integration: orderItems.orderId references orders.id, indirectly linking to teams.id.

* 1. Keyword <-> Category

Purpose: Links keywords to categories for search or filtering purposes.Cardinality:

* Many-to-One: A Keyword belongs to one Category, but a Category can have many Keywords.

Sequelize Definition:

Keyword.belongsTo(Category, { foreignKey: "categoryId", as: "categories" });

Category.hasMany(Keyword, { foreignKey: "categoryId" });

MySQL Schema Details:

* Foreign Key: keywords.categoryId references categories.categoryId (ON DELETE CASCADE, ON UPDATE CASCADE).
* Alias: categories (for Keyword → Category).

MongoDB Integration: Indirectly via products.categoryId, which connects to MongoDB’s carts and orderItems.Notes:

* Enhances product discoverability through keyword associations.

MongoDB Integration with MySQL RelationshipsThe MongoDB collections integrate with the MySQL database through UUID-based references, leveraging the flexibility of NoSQL for specific use cases. Below are the key integration points:

1. Users (users.userId in MySQL):
   * Referenced by:
     + attendances.userId (MongoDB): Tracks user attendance.
     + comments.userId (MongoDB): Links comments to users.
     + carts.userId (MongoDB): Associates carts with users.
     + refreshTokens.userId (MongoDB): Stores refresh tokens for user authentication.
     + verificationTokens.userId (MongoDB): Manages email verification tokens.
   * MySQL Relationships: Users are central to orders (createdBy), quotations (createdBy), invoices (createdBy), signatures, and roles.
2. Customers (customers.customerId in MySQL):
   * Referenced by:
     + carts.customerId (MongoDB): Optional link to customers for carts.
     + comments.resourceId (with resourceType: Customer) in MongoDB.
   * MySQL Relationships: Customers are linked to orders (createdFor), invoices, quotations, and vendors.
3. Products (products.productId in MySQL):
   * Referenced by:
     + carts.items.productId (MongoDB): Cart items reference products.
     + orderItems.items.productId (MongoDB): Order items reference products.
     + inventoryHistories.productId (MongoDB): Tracks product inventory changes.
     + comments.resourceId (with resourceType: Product) in MongoDB.
   * MySQL Relationships: Products are linked to brands, categories, vendors, and brand parent categories.
4. Orders (orders.id in MySQL):
   * Referenced by:
     + orderItems.orderId (MongoDB): Stores order items.
     + comments.resourceId (with resourceType: Order) in MongoDB.
   * MySQL Relationships: Orders are linked to users (createdBy), customers (createdFor), and teams (assignedTo).
5. Emails (no direct MySQL table, but users.email and customers.email):
   * Referenced by:
     + otps.email (MongoDB): For OTP verification.
     + verificationTokens.email (MongoDB): For email verification.
     + contacts.email (MongoDB): For contact form submissions.

Product Code Generation (REFER ./SEEDERS/GENERATEPRODUCTCODE.JS)

The productCode is a unique identifier for each product in the products table, stored as a VARCHAR(255) with a unique constraint. The script generates this code dynamically based on product attributes, ensuring it is unique across the dataset and follows a structured format. The generated code is used in the MySQL database and may be referenced in MongoDB collections (e.g., carts.items.productId, orderItems.items.productId) to link to products.productId, although the script specifically updates the productCode field.The script processes a JSON file (new.json) containing product data, assigns a productCode to products lacking one, and saves the updated data to product.json. The product code format combines a prefix (derived from product attributes and a mock brand) with a unique suffix to ensure uniqueness.

Product Code Format

The productCode follows a structured format: E + ProductType(2 chars) + BrandName(2 chars) + Last4 + Suffix(3 digits).Format Breakdown

1. Prefix (E): A static character E to indicate the code is for an external or e-commerce product.
2. ProductType (2 characters): The first two characters of the productType field (from the MySQL products table, an ENUM with values like tiles, sanitary), converted to uppercase.
3. BrandName (2 characters): The first two characters of the brand name, hardcoded as "Colston" in the script (from mockBrand.brandName), converted to uppercase.
4. Last4 (4 digits): A four-digit number derived from either:
   * The company\_code field (if present) in the format CMP-XXXX, where XXXX is extracted.
   * The size field from the product’s meta JSON (specifically, the value associated with the key "06857cb5-3fbe-404b-bdef-657c8ae7c345"), cleaned of asterisks and spaces, with the last four characters used. If unavailable, defaults to "0000".
5. Suffix (3 digits): A three-digit number (padded with leading zeros, e.g., 001, 002) to ensure uniqueness for products with the same prefix. The suffix increments until a unique code is found.

Example Product Code: ETICO1234001

* E: Static prefix.
* TI: First two characters of productType (tiles).
* CO: First two characters of brandName (Colston).
* 1234: Last four digits from company\_code (CMP-1234) or meta.size.
* 001: Unique three-digit suffix.

Product Code Generation Logic

The generateCode function is responsible for creating a unique productCode for each product. Below is a detailed explanation of how it works, based on the script:1. Input Parameters

* product: An object from new.json containing fields like company\_code, productType, and meta (a JSON object with metadata, e.g., size).
* existingCodes: An array of all existing productCode values in the JSON dataset to ensure uniqueness.
* mockBrand: A hardcoded object { brandName: "Colston" } used to derive part of the prefix.

2. Last4 Digit Extraction

The Last4 component (four digits) is derived as follows:

* If company\_code exists:
  + The company\_code is expected to follow the format CMP-XXXX (e.g., CMP-1234).
  + A regular expression (\d{4}(?!.\*\d)) extracts the last four digits (e.g., 1234).
  + If no match is found, Last4 defaults to "0000".
* If company\_code is absent:
  + The script looks for a size value in the meta JSON under the key "06857cb5-3fbe-404b-bdef-657c8ae7c345".
  + The size string (e.g., "12\*34 mm") is cleaned by removing asterisks (\*) and spaces, resulting in a string like "1234mm".
  + The last four characters are taken (e.g., 34mm), or "0000" if the cleaned string is shorter or unavailable.

3. Prefix Construction

The prefix is constructed by concatenating:

* Static E.
* First two characters of productType (e.g., tiles → TI, sanitary → SA), uppercase.
* First two characters of mockBrand.brandName (Colston → CO), uppercase.
* The Last4 digits from step 2.

Example: For a product with productType: "tiles", company\_code: "CMP-1234", and mockBrand.brandName: "Colston", the prefix is ETICO1234.4. Suffix GenerationTo ensure uniqueness:

* The script collects all existing productCode values that start with the generated prefix (e.g., ETICO1234).
* It extracts the suffixes (three-digit numbers, e.g., 001, 002) from these codes using a regular expression (/^\d{3}$/).
* The script starts with suffix = 1 and increments until it finds a suffix not present in the existing codes.
* The suffix is padded with leading zeros to three digits (e.g., 1 → 001).

Example: If ETICO1234001 and ETICO1234002 exist, the next code is ETICO1234003.5. Final CodeThe final productCode is the prefix plus the suffix (e.g., ETICO1234001).6. Script ExecutionThe updateProductJson function:

* Loads new.json into jsonData.
* Collects existing productCode values to avoid duplicates.
* Iterates through each product in jsonData:
  + Skips products with an existing productCode.
  + Generates a new productCode using generateCode.
  + Assigns the new code to the product and adds it to existingCodes.
* Saves the updated data to product.json.

Example Scenarios

1. Product with company\_code:
   * Input: { productType: "tiles", company\_code: "CMP-5678", meta: {} }
   * Last4: 5678 (from CMP-5678).
   * Prefix: ETICO5678 (E + TI + CO + 5678).
   * Suffix: 001 (assuming no existing codes with this prefix).
   * Output: productCode: "ETICO5678001".
2. Product with meta.size:
   * Input: { productType: "sanitary", meta: { "06857cb5-3fbe-404b-bdef-657c8ae7c345": "12\*34 mm" } }
   * Last4: 34mm (from cleaned 1234mm).
   * Prefix: ESACO34mm (E + SA + CO + 34mm).
   * Suffix: 001.
   * Output: productCode: "ESACO34mm001".
3. Product with neither:
   * Input: { productType: "tiles", meta: {} }
   * Last4: 0000 (default).
   * Prefix: ETICO0000.
   * Suffix: 001.
   * Output: productCode: "ETICO0000001".

static.cmtradingco.com - CDN and File Management System

Purpose

This document provides comprehensive documentation for https://static.cmtradingco.com, a Content Delivery Network (CDN) and file management system integrated with the spsyn8lm\_rocklime\_dashboard application and the cmtradingco.com main website. The purpose of this documentation is to:

* Describe the system's role as a CDN for hosting static assets such as product images, invoice PDFs, CSS, project images, and catalogues.
* Detail the file management capabilities, including upload, update, and deletion processes accessible to the operations team and non-technical users.
* Explain the unique file naming convention using company\_code or UUIDs to ensure human-readable and unique identifiers.
* Document the emailing system utilizing no-reply@static.cmtradingco.com for dashboard and website-related communications.
* Serve as a reference for developers, operations teams, and stakeholders to manage, maintain, and extend the system.

Scope

The documentation covers:

* CDN Functionality: Hosting and delivery of static files, including /product\_images, /invoice\_pdfs, and other builds.
* File Management: Upload, update, and deletion workflows, with a focus on accessibility for the operations team.
* File Naming Conventions: Use of company\_code or pre-generated UUIDs for unique and human-readable file names.
* Email Integration: Configuration and usage of no-reply@static.cmtradingco.com for automated emails.
* Integration: Linkages with cmtradingco.com and the spsyn8lm\_rocklime\_dashboard.
* Exclusions: Detailed deployment configurations, specific API endpoints, and advanced CDN optimization techniques are placeholders for future sections, pending additional input.

System Overview

https://static.cmtradingco.com serves as a dual-purpose platform: a Content Delivery Network (CDN) for distributing static assets and a file management system for uploading, updating, and deleting files. It is integral to the spsyn8lm\_rocklime\_dashboard application and the cmtradingco.com main website, providing a centralized repository for:

* Product Images: Primarily stored under /product\_images, linked to the dashboard’s products table.
* Invoice PDFs: Stored under /invoice\_pdfs, generated from the dashboard’s invoices table.
* CSS, Images, and Builds: Supporting the main website and dashboard UI.
* Project Images and Catalogues: Additional assets for marketing and operational use.

The system ensures files are uniquely identified using company\_code (from the MySQL products.company\_code field) or pre-generated UUIDs, making them human-readable and accessible to the operations team for manual management.

Additionally, no-reply@static.cmtradingco.com is configured as the primary email address for automated notifications and communications related to the dashboard and website, enhancing operational efficiency.Key features include:

* Scalable CDN delivery of static assets across global regions.
* Intuitive file management interface for non-technical users.
* Unique file naming to prevent duplication and maintain originality.
* Automated emailing system integrated with dashboard workflows.

.env

PORT=4000

REFRESH\_SECRET=

JWT\_SECRET=eyJhbGciOiJIUzI1NiJ9.eyJSb2xlIjoiQWRtaW4iLCJJc3N1ZXIiOiJJc3N1ZXIiLCJVc2VybmFtZSI6IkphdmFJblVzZSIsImV4cCI6MTc0NDI4MDc5MSwiaWF0IjoxNzQ0MjgwNzkxfQ.-xVmFv-Z71vNBcPPTHq8c286GuhaeUNi3MSB6u2StVk

MAILCHIMP\_KEY=00c2a687ba5d05bc7d55c6ecc2e2d106-us14

MAILCHIMP\_LIST\_KEY=

MAILGUN\_KEY=

MAILGUN\_DOMAIN=

MAILGUN\_EMAIL\_SENDER=

GOOGLE\_CLIENT\_ID=

GOOGLE\_CLIENT\_SECRET=

GOOGLE\_CALLBACK\_URL=http://localhost:3000/api/auth/google/callback

FACEBOOK\_CLIENT\_ID=

FACEBOOK\_CLIENT\_SECRET=

FACEBOOK\_CALLBACK\_URL=http://localhost:3000/api/auth/facebook/callback

CLIENT\_URL=http://localhost:3000

BASE\_API\_URL=

AWS\_ACCESS\_KEY\_ID=

AWS\_SECRET\_ACCESS\_KEY=

AWS\_REGION=us-east-2

AWS\_BUCKET\_NAME=

GOOGLE\_APPLICATION\_CREDENTIALS=./teamoffice.json

ADMIN\_EMAIL=admin@teamoffice.in

ADMIN\_PASSWORD=teamOffice@2024

DB\_NAME=spsyn8lm\_rocklime\_dashboard

DB\_USER=spsyn8lm\_rocklime\_dash

DB\_PASSWORD=rocklime@2025

DB\_HOST=119.18.54.11

DB\_PORT=3306

RAZORPAY\_KEYID=rzp\_live\_B87penn0KAZyRM

RAZORPAY\_SECRETKEY=1Lf68qNq8JOdLIGezAivFmIy

MONGO\_URI=mongodb+srv://dhruvermafz:A18AL87fFIor4MlC@cluster0.dbacq.mongodb.net/rocklime-dashboard?retryWrites=true&w=majority&appName=Cluster0

SMTP\_PASS="KB\_\*d-~[!ST="

FTP\_HOST=119.18.54.19

FTP\_PORT=21

FTP\_USER=static@static.cmtradingco.com

FTP\_PASSWORD="21=?,#ep4PmN"

FTP\_BASE\_URL=https://static.cmtradingco.com

APP\_HOST=dashboard-rocklime.vercel.app