GT Management System - Complete Documentation

Table of Contents

Project Overview

GT Management is a comprehensive business management solution designed to streamline operations for small and medium-sized enterprises. The system provides integrated modules for inventory management, customer relationship management, transaction processing, cheque management, and reporting.

GT Management provides a unified dashboard for managing all aspects of your business, from product inventory to customer relationships and financial transactions. The application is designed with a modern user interface that offers a smooth experience across devices.

Getting Started

To get started with GT Management:

1. Clone the repository
2. Install dependencies with `npm install`
3. Run the development server with `npm run dev`
4. Access the application at `http://localhost:3000`

Features

- **Dashboard**: Comprehensive overview of business metrics and performance indicators
- **Products Management**: Catalog, inventory, and pricing management
- **Customer Management**: Customer database with detailed profiles and history
- **Transaction Processing**: Purchase and sales management with line item tracking
- **Cheque Management**: Track customer and supplier cheques with multiple status options
- **Reports**: Business intelligence and analytical reports
- **Point of Sale (POS)**: Streamlined sales processing interface
- **Settings**: System configuration and user preferences

Architecture

GT Management is built with a modern frontend architecture using Next.js. The application follows a modular approach where components are organized by functionality and reusability.

Key Architectural Components:

- **App Router**: Leverages Next.js App Router for efficient page routing and navigation
- **Component Library**: Custom UI components built on top of Shadcn UI

- **Layouts**: Consistent layouts across pages with shared navigation and structure
- **State Management**: Local state with React hooks and context

Modules

Dashboard

The dashboard provides a quick overview of business performance with:

- Key performance indicators
- Sales and purchase charts
- Recent transactions
- Interactive data visualization

Products

The products module allows for:

- Creating and managing product catalog
- Organizing products by category and brand
- Setting up variations, units, and price groups
- Stock level monitoring

Customers

Customer management includes:

- Customer database with search and filtering
- Customer profiles with purchase history
- Credit limit management
- Communication tracking

Transactions

The transactions module handles:

- Purchase management from suppliers
- Sales record keeping
- Stock adjustments
- Expense tracking
- Transfer management
- Quotation and order processing

Cheques

Cheque management covers:

- Customer cheques tracking
- Supplier cheques management
- Expense cheques
- Guarantee cheques processing
- Status tracking (pending, deposited, cleared, bounced)

Settings

System settings allow for:

- User management and access control
- Business information configuration

- Tax and payment settings
- Notification preferences

Technology Stack

GT Management is built with the following technologies:

- **Frontend Framework**: Next.js 14
- **UI Library**: React with Shadcn UI components
- **Styling**: Tailwind CSS
- **State Management**: React Context API and Hooks
- **Icons**: Lucide React
- **Charts**: React Charts
- **Form Handling**: React Hook Form
- **Date Handling**: date-fns

Component Documentation

This section provides an overview of the key components used in GT Management System.

Layout Components

AppSidebar

The `AppSidebar` component provides the main navigation sidebar for the application.

Props:

- `variant` (string): Determines the style variant of the sidebar ('inset' by default)
- Any additional props are forwarded to the Sidebar component

Usage:

```
<AppSidebar variant="inset" />
```

SiteHeader

The `SiteHeader` component renders the top navigation bar with search, notifications, and user actions.

Usage:

```
<SiteHeader />
```

SidebarProvider

The `SidebarProvider` wraps the application to provide sidebar context and functionality.

Props:

- `style` (object): Custom CSS variables for sidebar dimensions

Usage:

```
<SidebarProvider
  style={{
    "--sidebar-width": "calc(var(--spacing) * 72)",
    "--header-height": "calc(var(--spacing) * 12)"
  }}>
  {children}
</SidebarProvider>
```

UI Components

The application uses a collection of UI components based on Shadcn UI, which are all accessible in the `src/components/ui/` directory. These include:

- `Button` : For interactive button elements
- `Card` : For content containers
- `Input` : For text input fields
- `Select` : For dropdown selection
- `Textarea` : For multi-line text input
- `Sidebar` : For navigation sidebar
- `Table` : For data tables
- `DatePicker` : For date selection

Form Components

Form components are used across the application to create consistent data entry interfaces.

LoginForm

The `LoginForm` component provides a authentication form with username and password fields.

Usage:

```
<LoginForm onSubmit={handleSubmit} />
```

Data Display Components

DataTable

The `DataTable` component provides a sortable, paginated table for displaying structured data.

Props:

- `data` (array): Array of data objects to display

Usage:

```
<DataTable data={data} />
```

ChartAreaInteractive

The `ChartAreaInteractive` component renders an interactive area chart for visualizing time-series data.

Usage:

```
<ChartAreaInteractive />
```

Navigation Components

NavMain

The `NavMain` component renders the main navigation items in the sidebar.

Props:

- `items` (array): Array of navigation items with title, URL, and icon properties

Usage:

```
<NavMain items={navMainItems} />
```

NavProducts, NavTransactions, NavCheques, NavSecondary

These components follow the same pattern as `NavMain` but are specialized for different sections of the application.

Special Components

DraggableCalculator

The `DraggableCalculator` component provides a calculator widget that can be dragged around the interface.

Usage:

```
<DraggableCalculator />
```

PointOfSale

The `PointOfSale` component in the POS section provides a complete interface for processing sales transactions.

Usage:

```
<PointOfSale />
```

User Guide

This section provides detailed instructions for using the GT Management System.

Dashboard

The dashboard is your central hub for monitoring business performance and accessing key features.

Accessing the Dashboard

1. Log in to the system using your credentials
2. You'll be automatically directed to the dashboard, or you can click on the "Dashboard" item in the sidebar

Dashboard Features

- **Key Performance Indicators (KPIs)**: View important metrics at the top of the dashboard
- **Sales Chart**: Interactive chart showing sales performance over time
- **Recent Transactions**: Quick overview of the latest transactions
- **Quick Navigation**: Access common tasks through action buttons

Using the Dashboard

- Hover over chart elements to see detailed information
- Click on any transaction in the recent transactions list to view its details
- Use the date range selector to adjust the time period for displayed data

Products Management

The Products module allows you to manage your inventory and product catalog.

Accessing Products Management

- Click on "Product List" in the sidebar under the Products section

Adding a New Product

1. Navigate to Products → Product List
2. Click the "Add New Product" button
3. Fill in the product details including:
    - Name
    - SKU
    - Category
    - Brand
    - Price information
    - Description
4. Click "Save Product" to add the product to your catalog

Editing a Product

1. Navigate to Products → Product List
2. Find the product you want to edit in the list
3. Click the "Edit" button for that product
4. Update the product information as needed
5. Click "Update Product" to save your changes

Managing Product Categories and Brands

- Navigate to Products → Category to manage product categories
- Navigate to Products → Brands to manage product brands

Customers Management

The Customers module helps you maintain your customer database and track customer information.

Accessing Customer Management

- Click on "Customers" in the main sidebar

Adding a New Customer

1. Navigate to Customers
2. Click the "Add New Customer" button
3. Fill in the customer details including:
    - Full Name
    - Customer Type (Individual/Business)
    - Contact Information
    - Address
    - Credit Limit
    - Tax Information

4. Click "Save Customer" to add the customer to your database

Editing Customer Information

1. Navigate to Customers
2. Find the customer you want to edit in the list
3. Click the "Edit" button for that customer
4. Update the customer information as needed
5. Click "Update Customer" to save your changes

Viewing Customer History

1. Navigate to Customers
2. Find the customer in the list
3. Click on the customer name to view their detailed profile
4. The profile page will show purchase history and payment details

Transactions

The Transactions module handles all purchase and sales records.

Managing Purchases

Creating a New Purchase

1. Navigate to Transactions → Purchase
2. Click "Create New Purchase"
3. Select a supplier from the dropdown
4. Enter reference number and purchase date
5. Select warehouse and status
6. Add products to the purchase:
    - Click "Add Item"
    - Select a product
    - Enter quantity, unit price, discounts, and taxes
7. Add any notes to the purchase
8. Click "Save Purchase" to create the purchase record

Editing a Purchase

1. Navigate to Transactions → Purchase
2. Find the purchase you want to edit
3. Click the "Edit" button for that purchase
4. Make your changes to the purchase details and line items
5. Click "Update Purchase" to save your changes

Cheques Management

The Cheques module helps you track and manage cheques from customers and to suppliers.

Managing Customer Cheques

Adding a New Customer Cheque

1. Navigate to Cheques → Customer Cheques
2. Click "Create New Cheque"
3. Select a customer from the dropdown
4. Enter the cheque amount, number, and bank information
5. Select cheque date and deposit date
6. Choose the current status of the cheque
7. Add any relevant notes

8. Click "Save Cheque" to create the cheque record

Updating Cheque Status

1. Navigate to Cheques → Customer Cheques
2. Find the cheque you want to update
3. Click the "Edit" button for that cheque
4. Change the status as needed (e.g., from "Pending" to "Cleared")
5. Click "Update Cheque" to save the changes

Managing Supplier Cheques

The process is similar to customer cheques but accessed through Cheques → Supplier Cheques.

Point of Sale (POS)

The POS module provides a streamlined interface for processing sales transactions quickly.

Accessing the POS

- Click on "POS" in the main navigation

Processing a Sale

1. Select a customer or create a new one
2. Add products to the cart:
    - Browse the product grid or use the search function
    - Click on products to add them to the cart
    - Adjust quantities as needed
3. Apply any discounts
4. Select payment method
5. Process the payment
6. Print or email receipt

Settings

The Settings module allows you to configure the system according to your business needs.

Accessing Settings

- Click on "Settings" in the main sidebar

Available Settings

- **User Management**: Add, edit, or remove users
- **Business Settings**: Configure business details
- **Tax Settings**: Set up tax rates and rules
- **Payment Methods**: Configure payment options
- **Notification Settings**: Customize system notifications

Technical Architecture

This section outlines the technical architecture of the GT Management System, providing insights into the system design, code organization, and implementation details.

System Overview

GT Management System is a Next.js-based application designed with a modern, component-based architecture. The system follows a modular approach to support maintainability and scalability.

Key Technical Features

- Next.js App Router for efficient page routing
- React components with separation of concerns
- Responsive design using Tailwind CSS
- Client-side state management with React Context
- Progressive enhancement for improved performance

Project Structure

The project is organized into the following main directories:

```
src/
  app/            # Next.js App Router pages and layouts
    dashboard/    # Dashboard module and submodules
    login/        # Authentication pages
    pos/          # Point of Sale module
  components/     # Reusable React components
    ui/           # Base UI components
    pos/          # POS-specific components
  hooks/          # Custom React hooks
  lib/            # Utility functions and shared code
```

Key Files

- `app/layout.js` : Root layout that applies to all pages
- `app/dashboard/layout.js` : Dashboard-specific layout with sidebar
- `components/app-sidebar.jsx` : Main navigation component
- `components/ui/sidebar.jsx` : Base sidebar component

Frontend Architecture

The frontend architecture is built around a component-based model with clear separation of concerns.

Component Hierarchy

1. **Layout Components**: Provide structural elements (app-sidebar.jsx, site-header.jsx)
2. **Feature Components**: Implement specific business features (point-of-sale.jsx, data-table.jsx)
3. **UI Components**: Reusable UI elements (button.jsx, card.jsx, input.jsx)

Data Flow

- Data typically flows from parent components to children through props
- React Context is used for global state (theme, sidebar collapse state)
- Forms use controlled components with local state

Component Design

Components are designed for reusability, composability, and maintainability.

Component Patterns

1. **Container Components**: Manage data and state, then pass to presentation components
2. **Presentation Components**: Render UI based on props with minimal state
3. **Layout Components**: Handle structural aspects of the UI
4. **Higher-Order Components**: Add functionality to existing components

Component Example: AppSidebar

```
export function AppSidebar(props) {
  return (
    <Sidebar collapsible="offcanvas" {...props}>
      <SidebarHeader>
        {/* Header content */}
      </SidebarHeader>
      <SidebarContent>
        <NavMain items={data.navMain} />
        <NavProducts items={data.navProducts} />
        {/* Additional navigation sections */}
      </SidebarContent>
      <SidebarFooter>
        <NavUser user={data.user} />
      </SidebarFooter>
    </Sidebar>
  )
}
```

Routing Strategy

The application uses Next.js App Router for a file-system based routing approach.

Route Organization

- **Dashboard Routes**: `/dashboard/[module]/[action]`
- **Dynamic Routes**: `/dashboard/products/[id]/edit`
- **Nested Layouts**: Layout hierarchy from root to page components

Route Implementation

Each route is implemented as a React component in the appropriate directory structure. For example:

- The product edit page is at `/app/dashboard/products/[id]/edit/page.jsx`
- The customer creation page is at `/app/dashboard/customers/create/page.jsx`

State Management

State management is primarily handled through React's built-in hooks with Context API for global state when needed.

State Management Strategies

- **Component State**: `useState` hook for component-specific state
- **Context API**: For sharing state across component tree (sidebar collapse state)
- **Custom Hooks**: For reusable state logic (e.g., `use-mobile.js`)

UI and Styling

The UI is built with a component library based on Shadcn UI, styled with Tailwind CSS.

Styling Approach

- **Tailwind CSS**: For responsive, utility-first styling
- **CSS Variables**: For theming and consistent design tokens
- **Component-based Styling**: Styles are encapsulated within components

Performance Considerations

The application implements several performance optimizations:

- **Code Splitting**: Using dynamic imports for code splitting
- **Lazy Loading**: For non-critical components
- **Image Optimization**: Using Next.js Image component
- **Memoization**: Using `React.memo()` and `useMemo()` for expensive operations

Deployment

The application is designed for deployment on platforms like Vercel, Netlify, or any static hosting service.

API Documentation

This section outlines the API endpoints used by the GT Management Frontend when communicating with the backend services.

Base URL

All API endpoints are relative to the base URL of the backend service:

```
https://api.gt-management.com/v1
```

Authentication APIs

Login

Endpoint: `POST /auth/login`

Request Body:

```json
{
  "username": "user@example.com",
  "password": "password123"
}
```

Response:

```json
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": "1",
    "name": "John Doe",
    "email": "user@example.com",
    "role": "admin"
  }
}
```

Logout

Endpoint: `POST /auth/logout`

Headers:

```
Authorization: Bearer <token>
```

Response:

```json
{
  "success": true,
  "message": "Successfully logged out"
}
```

Products APIs

Get All Products

Endpoint: `GET /products`

Query Parameters:

- `page` (optional): Page number for pagination
- `limit` (optional): Number of items per page
- `search` (optional): Search term
- `category` (optional): Filter by category ID
- `brand` (optional): Filter by brand ID

Response:

```json
{
  "data": [
    {
      "id": "1",
      "name": "Product 1",
      "sku": "SKU-001",
      "category": {
        "id": "1",
        "name": "Electronics"
      },
      "brand": {
        "id": "1",
        "name": "Brand A"
      },
      "purchasePrice": 100.00,
      "sellingPrice": 150.00,
      "stock": 50
    },
    // Additional products...
  ],
  "meta": {
    "total": 100,
    "page": 1,
    "limit": 10,
    "totalPages": 10
  }
}
```

Get Product by ID

Endpoint: `GET /products/{id}`

Response:

```
{
  "id": "1",
  "name": "Product 1",
  "sku": "SKU-001",
  "category": {
    "id": "1",
    "name": "Electronics"
  },
  "brand": {
    "id": "1",
    "name": "Brand A"
  },
  "purchasePrice": 100.00,
  "sellingPrice": 150.00,
  "description": "Detailed product description...",
  "stockDetails": [
    {
      "warehouseId": "1",
      "warehouseName": "Main Warehouse",
      "quantity": 30
    },
    {
      "warehouseId": "2",
      "warehouseName": "Secondary Warehouse",
      "quantity": 20
    }
  ],
  "createdAt": "2025-01-15T08:30:00Z",
  "updatedAt": "2025-04-10T14:25:30Z"
}
```

Create Product

Endpoint: `POST /products`

Headers:

```
Authorization: Bearer <token>
Content-Type: application/json
```

Request Body:

```json
{
  "name": "New Product",
  "sku": "SKU-NEW",
  "categoryId": "1",
  "brandId": "1",
  "purchasePrice": 80.00,
  "sellingPrice": 120.00,
  "description": "Description of the new product",
  "openingStock": 10,
  "warehouseId": "1"
}
```

Response:

```json
{
  "id": "123",
  "name": "New Product",
  "sku": "SKU-NEW",
  "category": {
    "id": "1",
    "name": "Electronics"
  },
  "brand": {
    "id": "1",
    "name": "Brand A"
  },
  "purchasePrice": 80.00,
  "sellingPrice": 120.00,
  "description": "Description of the new product",
  "stock": 10,
  "createdAt": "2025-04-26T10:15:30Z",
  "updatedAt": "2025-04-26T10:15:30Z"
}
```

Update Product

Endpoint: `PUT /products/{id}`

Headers:

```
Authorization: Bearer <token>
Content-Type: application/json
```

Request Body:

```json
{
  "name": "Updated Product Name",
  "purchasePrice": 85.00,
  "sellingPrice": 125.00,
  "description": "Updated description"
}
```

Response:

```
{
  "id": "123",
  "name": "Updated Product Name",
  "sku": "SKU-NEW",
  "category": {
    "id": "1",
    "name": "Electronics"
  },
  "brand": {
    "id": "1",
    "name": "Brand A"
  },
  "purchasePrice": 85.00,
  "sellingPrice": 125.00,
  "description": "Updated description",
  "stock": 10,
  "createdAt": "2025-04-26T10:15:30Z",
  "updatedAt": "2025-04-26T11:30:15Z"
}
```

Error Handling

All API endpoints follow a consistent error format:

```
{
  "error": true,
  "code": "RESOURCE_NOT_FOUND",
  "message": "The requested resource could not be found",
  "details": {
    "resource": "Product",
    "id": "123"
  }
}
```

Common error codes:

- `UNAUTHORIZED` : Authentication required or invalid token
- `FORBIDDEN` : Insufficient permissions
- `RESOURCE_NOT_FOUND` : Requested resource does not exist
- `VALIDATION_ERROR` : Invalid input data
- `SERVER_ERROR` : Internal server error

Pagination

Endpoints that return collections support pagination with the following query parameters:

- `page` : Page number (default: 1)

- `limit` : Items per page (default: 10)

Response includes metadata:

```
{
  "data": [...],
  "meta": {
    "total": 100,
    "page": 2,
    "limit": 10,
    "totalPages": 10
  }
}
```