GarageGuru Documentation Archive

# GarageGuru - Complete Documentation Archive

**Document**: Documentation Archive 4 of 5  
**Creation Date**: August 30, 2025  
**Content**: ALL project documentation, guides, and deployment information  
**Purpose**: Complete rebuild guide and technical documentation

This document contains comprehensive documentation for rebuilding and understanding the GarageGuru system.

## Table of Contents

1. [Project Overview](#project-overview)
2. [System Architecture](#system-architecture)
3. [Database Design](#database-design)
4. [Step-by-Step Rebuild Guide](#step-by-step-rebuild-guide)
5. [Environment Configuration](#environment-configuration)
6. [Deployment Instructions](#deployment-instructions)
7. [API Documentation](#api-documentation)
8. [Development Workflow](#development-workflow)

## Project Overview

### System Purpose

GarageGuru is a comprehensive multi-tenant garage management system designed to streamline automotive service operations. It provides complete business management tools for garage owners, including customer tracking, inventory management, job scheduling, invoicing, and financial analytics.

### Key Features

* **Multi-tenant Architecture**: Complete data isolation between garages
* **Role-based Access Control**: Super admin, garage admin, and mechanic staff levels
* **Customer Management**: Comprehensive customer database with service history
* **Job Card System**: Service request tracking from creation to completion
* **Inventory Control**: Spare parts management with barcode scanning and low-stock alerts
* **Invoice Generation**: Professional PDF invoices with WhatsApp integration
* **Sales Analytics**: Revenue tracking and profit calculations
* **Real-time Notifications**: System alerts for low stock and milestones
* **Barcode Scanning**: Multi-format barcode support with mobile optimization

### Technology Stack Summary

* **Frontend**: React 18 + TypeScript + Wouter + Shadcn/UI + Tailwind CSS
* **Backend**: Express.js + TypeScript + JWT Authentication + PostgreSQL
* **Database**: Render.com PostgreSQL with Drizzle ORM
* **Build Tools**: Vite + ESBuild + TypeScript compiler
* **UI Libraries**: Radix UI primitives + Lucide icons
* **External Services**: Gmail SMTP + CloudFlare R2 + WhatsApp Business API

## System Architecture

### Frontend Architecture

client/  
├── src/  
│ ├── components/ # Reusable UI components  
│ │ ├── ui/ # Shadcn/UI base components  
│ │ └── layout/ # Layout components (Navigation, Sidebar)  
│ ├── pages/ # Route-based page components  
│ │ ├── auth/ # Authentication pages  
│ │ ├── dashboard/ # Dashboard and analytics  
│ │ ├── customers/ # Customer management  
│ │ ├── inventory/ # Spare parts management  
│ │ ├── jobs/ # Job cards and service tracking  
│ │ └── invoices/ # Invoice management  
│ ├── hooks/ # Custom React hooks  
│ ├── lib/ # Utility libraries and configurations  
│ └── contexts/ # React context providers

### Backend Architecture

server/  
├── index.ts # Main server entry point  
├── routes.ts # API route handlers (2500+ lines)  
├── storage.ts # Database operations layer (1000+ lines)  
├── migrations.ts # Database schema management  
├── gmailEmailService.ts # Email service integration  
├── invoice-renderer.ts # PDF generation service  
├── middleware.ts # Authentication and authorization  
└── db.ts # PostgreSQL connection pool

### Database Schema

9 Core Tables:  
├── garages # Multi-tenant garage information  
├── users # User accounts with role-based access  
├── customers # Customer database per garage  
├── spare\_parts # Inventory management per garage  
├── job\_cards # Service requests and tracking  
├── invoices # Generated invoices with PDF tokens  
├── notifications # System notifications per garage  
├── otp\_records # MFA and password reset tokens  
└── audit\_logs # Security and action logging

## Database Design

### Multi-tenant Data Isolation

Every table (except users and garages) includes a garage\_id foreign key to ensure complete data separation between different garage businesses. This design allows: - Complete data privacy between garages - Independent operation of multiple garage businesses - Scalable architecture for growing customer base

### Key Relationships

garages (1) → (N) users  
garages (1) → (N) customers  
garages (1) → (N) spare\_parts  
garages (1) → (N) job\_cards  
garages (1) → (N) invoices  
garages (1) → (N) notifications  
  
customers (1) → (N) job\_cards  
job\_cards (1) → (N) invoices  
users (1) → (N) job\_cards (as completed\_by)

### Data Types and Constraints

* **UUID Primary Keys**: All tables use gen\_random\_uuid() for security
* **JSONB Storage**: Spare parts in job cards stored as structured JSON
* **Decimal Precision**: All monetary values use DECIMAL(10,2) for accuracy
* **Timestamp Tracking**: Created/updated timestamps on all records
* **Referential Integrity**: Foreign key constraints ensure data consistency

## Step-by-Step Rebuild Guide

### Prerequisites

1. **Node.js 20+** - Runtime environment
2. **PostgreSQL Database** - Render.com or any PostgreSQL provider
3. **Gmail Account** - For SMTP email services (optional)
4. **Code Editor** - VS Code recommended with TypeScript support

### 1. Environment Setup

# Clone or create project directory  
mkdir garageguru-rebuild  
cd garageguru-rebuild  
  
# Initialize Node.js project  
npm init -y  
  
# Install all dependencies (80+ packages)  
npm install [see package.json dependencies list]

### 2. Database Configuration

# Set up PostgreSQL database  
# Option A: Use Render.com PostgreSQL (recommended)  
# Option B: Local PostgreSQL installation  
# Option C: Docker PostgreSQL container  
  
# Configure database connection  
export DATABASE\_URL="postgresql://user:pass@host:port/database"

### 3. Project Structure Creation

# Create directory structure  
mkdir -p client/src/{components,pages,hooks,lib,contexts}  
mkdir -p client/src/components/{ui,layout}  
mkdir -p server  
mkdir -p shared  
mkdir -p attached\_assets  
  
# Copy all source files from archives  
# (Refer to Archive\_01\_Frontend\_Complete.md and Archive\_02\_Backend\_Complete.md)

### 4. Configuration Files

Copy all configuration files: - package.json - Dependencies and scripts - tsconfig.json - TypeScript configuration - vite.config.ts - Build tool configuration - tailwind.config.ts - Styling configuration - drizzle.config.ts - Database ORM configuration

### 5. Database Migration

# Run database migrations  
npm run dev  
# Migrations run automatically on server start  
# Creates all 9 tables with proper schema

### 6. Development Server

# Start development server  
npm run dev  
# Server runs on http://localhost:5000  
# Frontend and backend served together

### 7. Production Build

# Build for production  
npm run build  
# Creates optimized build in /dist directory

## Environment Configuration

### Required Environment Variables

# Database Connection  
DATABASE\_URL="postgresql://user:password@host:port/database"  
  
# JWT Authentication  
JWT\_SECRET="YourSecureJWTSecretKey"  
  
# Email Configuration (Optional)  
GMAIL\_USER="your-gmail@gmail.com"  
GMAIL\_APP\_PASSWORD="your-app-specific-password"  
  
# Super Admin Access  
ADMIN\_ACTIVATION\_CODE="your-admin-code"  
STAFF\_ACTIVATION\_CODE="your-staff-code"  
SUPER\_ADMIN\_EMAIL="admin@yourdomain.com"  
  
# Environment  
NODE\_ENV="development" or "production"

### Optional Environment Variables

# File Storage (if using cloud storage)  
CLOUDINARY\_CLOUD\_NAME="your-cloud-name"  
CLOUDINARY\_API\_KEY="your-api-key"  
CLOUDINARY\_API\_SECRET="your-api-secret"  
  
# WhatsApp Integration (if using)  
WHATSAPP\_API\_KEY="your-whatsapp-api-key"  
WHATSAPP\_PHONE\_NUMBER="your-business-phone"

## Deployment Instructions

### Render.com Deployment (Recommended)

1. **Create Web Service** on Render.com
2. **Connect GitHub Repository**
3. **Configure Build Settings**:
   * Build Command: npm run build
   * Start Command: npm start
   * Environment: Node
4. **Add Environment Variables** (as listed above)
5. **Deploy**

### Alternative Deployment Options

* **Vercel**: Serverless deployment with edge functions
* **Heroku**: Container-based deployment
* **DigitalOcean**: VPS with PM2 process management
* **AWS**: EC2 with Load Balancer and RDS
* **Google Cloud**: App Engine with Cloud SQL

## API Documentation

### Authentication Endpoints

POST /api/auth/login # User authentication  
POST /api/auth/register # User registration (with codes)  
POST /api/auth/change-password # Password change with MFA  
POST /api/auth/request-access # Access request for new users

### Garage Management

GET /api/garages # List garages (filtered by role)  
POST /api/garages # Create new garage  
PUT /api/garages/:id # Update garage information

### Customer Management

GET /api/customers/:garageId # List customers  
POST /api/customers # Create customer  
PUT /api/customers/:id # Update customer  
GET /api/customers/search/:query # Search customers

### Inventory Management

GET /api/spare-parts/:garageId # List parts  
POST /api/spare-parts # Add part  
PUT /api/spare-parts/:id # Update part  
DELETE /api/spare-parts/:id # Delete part  
GET /api/spare-parts/low-stock/:garageId # Low stock alert

### Job Card System

GET /api/job-cards/:garageId # List job cards  
POST /api/job-cards # Create job card  
PUT /api/job-cards/:id # Update job card  
PUT /api/job-cards/:id/complete # Mark as completed

### Invoice Generation

GET /api/invoices/:garageId # List invoices  
POST /api/invoices # Generate invoice  
GET /api/invoices/:id/pdf # Download PDF  
POST /api/invoices/:id/whatsapp # Send via WhatsApp

### Analytics

GET /api/analytics/sales/:garageId # Sales statistics  
GET /api/analytics/monthly/:garageId # Monthly data  
GET /api/analytics/profit/:garageId # Profit calculations

## Development Workflow

### Code Organization

* **Frontend**: React components with TypeScript
* **Backend**: Express.js with middleware pattern
* **Database**: Drizzle ORM with type-safe queries
* **Styling**: Tailwind CSS with Shadcn/UI components
* **State Management**: TanStack Query + React Context

### Development Commands

npm run dev # Start development server  
npm run build # Build for production  
npm run start # Start production server  
npm run check # TypeScript type checking  
npm run db:push # Push database schema changes

### Testing Strategy

* **Frontend Testing**: Component testing with React Testing Library
* **Backend Testing**: API endpoint testing with Jest/Supertest
* **Integration Testing**: End-to-end testing with Playwright
* **Database Testing**: Migration testing and data integrity checks

*This documentation provides everything needed to rebuild and understand the GarageGuru system. For specific implementation details, refer to the source code archives.*