Housingram Backend - Complete Documentation

Table of Contents

---

1. Overview

Housingram Backend is a robust, enterprise-grade multi-tenant property management system built with Node.js, Express, and PostgreSQL using a schema-per-tenant isolation strategy.

Key Features

- Multi-tenant architecture with complete data isolation
- Role-based access control (RBAC) - 4 roles
- JWT-based authentication
- Automated schema provisioning for new tenants
- Comprehensive audit logging
- Property and unit management with booking system
- Super Admin dashboard with cross-tenant statistics
- Winston logging with automatic log rotation
- Interactive Swagger API documentation
- Docker containerization
- Cloud deployment guides (Railway, Render, DigitalOcean)

Tech Stack

- **Runtime**: Node.js 18+
- **Framework**: Express.js
- **Database**: PostgreSQL 15+ with Sequelize
- **Migrations**: Umzug
- **Validation**: Joi
- **Authentication**: JWT
- **Password Hashing**: bcryptjs
- **Logging**: Winston with daily log rotation
- **API Documentation**: Swagger/OpenAPI 3.0
- **Containerization**: Docker & Docker Compose

User Roles

1. Super Admin (Public Schema)

- Onboard and manage tenants
- View aggregated statistics across all tenants
- Activate/deactivate tenant accounts

2. Admin (Tenant Schema)

- Full access within tenant
- User management
- Project and unit CRUD operations

3. Sales (Tenant Schema)

- View projects and units
- Book available units
- View audit logs

4. Viewer (Tenant Schema)

- Read-only access to projects and units

---

2. Architecture

Multi-Tenancy Strategy

Schema-per-Tenant

- Each tenant (builder/developer) gets a dedicated PostgreSQL schema
- Complete data segregation between tenants for security and compliance
- Public schema stores tenant metadata and Super Admin users
- Tenant schemas contain users, projects, units, and audit logs

Design Patterns

Repository Pattern: Data access layer separated from business logic

DTO (Data Transfer Objects): Input/output data transformation and validation

Service Layer: Business logic encapsulation

Middleware Chain: Authentication → Tenant Resolution → RBAC → Audit Logging

Database Schema

Public Schema

- tenants - Tenant/builder information
    - id, name, contact_email, contact_phone, subscription_type, schema_name, is_active, created_at, updated_at

- users - Super Admin users only
    - id, name, email, password_hash, role (always 'Super Admin'), is_active, created_at, updated_at

Tenant Schema (e.g., tenant_1, tenant_2)

- users - Tenant-specific users (Admin, Sales, Viewer)
    - id, name, email, password_hash, role, is_active, created_at, updated_at, deleted_at
- projects - Property projects
    - id, name, location, description, total_units, status, is_active, created_at, updated_at, deleted_at
- units - Individual units within projects
    - id, project_id, unit_number, floor, area, bedrooms, bathrooms, price, status, booked_by, booked_at, sold_at, is_active, created_at, updated_at, deleted_at
- audit_logs - Audit trail of all operations
    - id, user_id, user_name, action, entity, entity_id, old_values, new_values, ip_address, user_agent, created_at

Request Flow

```
User Request
  ↓
Authentication Middleware (JWT verification)
  ↓
Tenant Resolver (Schema identification and switching)
  ↓
RBAC Check (Permission verification)
  ↓
Validation (Joi schema validation)
  ↓
Controller (Request handling)
  ↓
Service (Business logic)
  ↓
Repository (Data access)
  ↓
Database (PostgreSQL with schema context)
  ↓
Response DTO (Data transformation)
  ↓
Audit Logger (Automatic logging for mutations)
  ↓
JSON Response
```

---

3. Quick Start

Option 1: Docker (Recommended)

The fastest way to get started:

```
# Clone the repository
git clone <repository-url>
cd housingram-backend

# Create environment file
cp .env.example .env

# Start everything with Docker
docker-compose up -d

# Run migrations
docker-compose exec app npm run migrate:public
docker-compose exec app npm run seed:super-admin

# Check logs
docker-compose logs -f app
```

Access the application:

- API: http://localhost:3000
- Health Check: http://localhost:3000/api/health
- **Swagger API Docs**: http://localhost:3000/api-docs

Default Super Admin Credentials:

- Email: superadmin@housingram.com
- Password: SuperAdmin@123

Option 2: Manual Installation

Prerequisites:

- Node.js >= 18.x
- PostgreSQL >= 15.x
- npm or yarn

Steps:

1. Clone and install dependencies:

```
git clone <repository-url>
cd housingram-backend
npm install
```

2. Setup environment variables:

```
cp .env.example .env
# Edit .env with your database credentials and JWT secret
```

3. Create PostgreSQL database:

```
psql -U postgres
CREATE DATABASE housingram_db;
\q
```

4. Run migrations and seed:

```
npm run migrate:public
npm run seed:super-admin
```

5. Start the server:

```
# Development
npm run dev

# Production
npm start
```

---

4. API Endpoints

Authentication

Super Admin Login

```
POST /api/v1/auth/super-admin/login
Content-Type: application/json

{
  "email": "superadmin@housingram.com",
  "password": "SuperAdmin@123"
}

Response:
{
  "success": true,
  "message": "Login successful",
  "data": {
    "user": { ... },
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

Tenant User Login

```
POST /api/v1/auth/login
Content-Type: application/json

{
  "email": "admin@tenant.com",
  "password": "password123"
}
```

Refresh Token

```
POST /api/v1/auth/refresh
Authorization: Bearer <your-jwt-token>
```

Super Admin - Tenants

Create Tenant

```
POST /api/v1/super-admin/tenants
Authorization: Bearer <super-admin-token>

{
  "name": "ABC Builders Pvt Ltd",
  "contact_email": "contact@abcbuilders.com",
  "contact_phone": "+91-9876543210",
  "subscription_type": "Premium",
  "admin_name": "Amit Sharma",
  "admin_email": "amit@abcbuilders.com",
  "admin_password": "SecurePassword123!"
}
```

List All Tenants

```
GET /api/v1/super-admin/tenants?page=1&limit=10
Authorization: Bearer <super-admin-token>
```

Get Tenant Statistics

```
GET /api/v1/super-admin/stats
Authorization: Bearer <super-admin-token>

Response:
{
  "totalTenants": 5,
  "activeTenants": 4,
  "inactiveTenants": 1,
  "totalProjects": 25,
  "totalUnits": 500,
  "availableUnits": 320,
  "bookedUnits": 150,
  "soldUnits": 30
}
```

Tenant - Projects

Create Project

```
POST /api/v1/projects
Authorization: Bearer <tenant-admin-token>

{
  "name": "Green Valley Residences",
  "location": "Sector 42, Gurgaon, Haryana",
  "description": "Premium 3BHK and 4BHK apartments",
  "total_units": 150,
  "status": "Under Construction"
}
```

List Projects

```
GET /api/v1/projects?page=1&limit=10&status=Under Construction
Authorization: Bearer <tenant-token>
```

Get Project by ID

```
GET /api/v1/projects/:id
Authorization: Bearer <tenant-token>
```

Update Project

```
PATCH /api/v1/projects/:id
Authorization: Bearer <tenant-admin-token>

{
  "status": "Completed"
}
```

Tenant - Units

Create Unit

```
POST /api/v1/units
Authorization: Bearer <tenant-admin-token>

{
  "project_id": "a3bb189e-8bf9-3888-9912-ace4e6543002",
  "unit_number": "A-101",
  "floor": 1,
  "area": 1250.5,
  "bedrooms": 3,
  "bathrooms": 2,
  "price": 4500000,
  "status": "Available"
}
```

List Units with Filters

```
GET /api/v1/units?status=Available&min_price=3000000&max_price=5000000&bedrooms=3
Authorization: Bearer <tenant-token>
```

Book Unit

```
POST /api/v1/units/:id/book
Authorization: Bearer <tenant-sales-token>

{
  "customer_name": "Rajesh Kumar",
  "customer_email": "rajesh.kumar@example.com",
  "customer_phone": "+91-9876543210"
}
```

Tenant - Audit Logs

Get Audit Logs

```
GET /api/v1/audit-logs?action=CREATE&entity=unit&start_date=2024-01-01
Authorization: Bearer <tenant-admin-token>
```

---

5. Deployment

Local Docker Deployment

```
# Start services
docker-compose up -d

# Stop services
docker-compose down

# View logs
docker-compose logs -f app

# Rebuild after code changes
docker-compose up -d --build

# Access container shell
docker-compose exec app sh
```

Cloud Platforms

Railway (Recommended - Free Tier)

1. Create a Railway account at railway.app
2. Create a new project from GitHub repo

3. Add PostgreSQL database service
4. Configure environment variables:
    - NODE_ENV=production
    - JWT_SECRET=<your-secret-key>
    - JWT_EXPIRY=24h
5. Railway auto-detects Node.js and deploys
6. Run migrations via Railway shell

Render (Free Tier Available)

1. Create Render account at render.com
2. Create PostgreSQL database (note Internal Database URL)
3. Create Web Service from repository
4. Configure:
    - Build Command: npm install
    - Start Command: npm start
    - Environment variables (including DATABASE_URL)
5. Deploy and run migrations

DigitalOcean App Platform

1. Create DigitalOcean account
2. Create managed PostgreSQL database
3. Create App from GitHub repository
4. Attach PostgreSQL database
5. Configure environment variables
6. Deploy

Environment Configuration

Required environment variables:

```
NODE_ENV=production
PORT=3000
DB_HOST=localhost
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=your_secure_password
DB_NAME=housingram_db
JWT_SECRET=your-super-secret-jwt-key-minimum-32-characters
JWT_EXPIRY=24h
```

Generate JWT Secret:

```
openssl rand -base64 32
```

Production Checklist

- [ ] Change default super admin password

- [ ] Use strong, randomly generated JWT_SECRET
- [ ] Use strong database password
- [ ] Enable HTTPS
- [ ] Set NODE_ENV to production
- [ ] Review and update CORS settings if needed
- [ ] Run all migrations successfully
- [ ] Verify database backups are configured
- [ ] Test all API endpoints
- [ ] Verify authentication and authorization
- [ ] Configure log aggregation
- [ ] Set up error tracking (optional: Sentry)
- [ ] Configure uptime monitoring

---

6. Logging

Winston Logging System

Features

- **Multiple Transports**:
    - Console (colorized, development)
    - File (error.log - errors only)
    - File (combined.log - all logs)
    - Daily rotate (application-DATE.log)
- **Log Rotation**:
    - Automatic daily rotation
    - 5MB max file size for error/combined
    - 20MB for application logs
    - Automatic compression (zip)
    - Configurable retention (14 days)
- **Structured Logging**:
    - JSON format for easy parsing
    - Contextual information
    - Request logging middleware
    - Error logging middleware

Log Levels

- error - Error events
- warn - Warning messages
- info - Informational messages
- http - HTTP requests
- debug - Debug information

Usage Examples

```
const logger = require('./src/config/logger');

// General logging
logger.info('Application started');
logger.error('Database connection failed', { error: err.message });

// HTTP request logging (automatic via middleware)
logger.logRequest(req, statusCode, responseTime);

// Authentication logging
logger.logAuth(email, success, reason, ipAddress);

// Database query logging
logger.logDbQuery(query, duration);
```

Log Files

- logs/error.log - Error logs only
- logs/combined.log - All logs
- logs/application-2024-10-23.log - Daily application logs
- logs/exceptions.log - Unhandled exceptions
- logs/rejections.log - Unhandled promise rejections

---

7. Testing

Postman Collection

A comprehensive Postman collection is provided: Housingram_API.postman_collection.json

Features:

- All API endpoints organized by module
- Pre-configured environment variables
- Auto-saving of tokens and IDs
- Detailed descriptions for each endpoint
- Example requests and responses

Testing Workflow

1. **Import Collection and Environment**:
    - Import Housingram_API.postman_collection.json
    - Import Housingram_Environment.postman_environment.json
2. **Test Super Admin Features**:
    - Login as Super Admin (token auto-saves)
    - Create tenant (tenant_id and admin credentials auto-save)
    - View statistics
    - Activate/deactivate tenants

3. **Test Tenant Features**:
    - Login as tenant admin (token auto-saves)
    - Create projects (project_id auto-saves)
    - Create units (unit_id auto-saves)
    - Create sales user
    - Login as sales user
    - Book units
    - View audit logs

Interactive Swagger Documentation

Access at: http://localhost:3000/api-docs

- Complete API reference
- Try-it-out functionality
- Request/response examples
- Schema definitions
- Authentication flow testing

---

8. Implementation Details

Modules Implemented

1. Tenants Module

Location: src/modules/tenants/

Features:

- Create tenant with automatic schema provisioning
- List all tenants with pagination
- Get tenant by ID
- Update tenant information
- Activate/Deactivate tenants
- Tenant statistics

Architecture:

- DTOs: Input validation and output transformation
- Repository: Database operations
- Service: Business logic
- Controller: Request handling
- Routes: API endpoints with RBAC

2. Users Module

Location: src/modules/users/

Features:

- Super Admin authentication (public schema)

- Tenant user authentication (cross-schema search)
- User CRUD operations
- Token refresh
- Password hashing and verification

3. Projects Module

Location: src/modules/projects/

Features:

- Project CRUD operations
- Pagination and filtering
- Soft delete
- Automatic audit logging

4. Units Module

Location: src/modules/units/

Features:

- Unit CRUD operations
- Advanced filtering (status, price range, bedrooms, project)
- Unit booking workflow
- Status tracking (Available, Booked, Sold)
- Automatic audit logging

5. Audit Logs Module

Location: src/modules/auditLogs/

Features:

- View audit logs with filtering
- Track all mutations (CREATE, UPDATE, DELETE, BOOK)
- Capture user context, IP address, user agent
- Store old and new values for comparison

Middleware

1. **Authentication** (src/middleware/auth.js):
    - JWT token verification
    - User context injection
2. **Tenant Resolver** (src/middleware/tenantResolver.js):
    - Schema identification from token
    - Automatic schema switching
    - Tenant status verification
3. **RBAC** (src/middleware/rbac.js):
    - Role-based access control

    - Permission matrix enforcement
    - Resource-action verification

4. **Validation** (src/middleware/validate.js):
    - Joi schema validation
    - Request body/query/params validation

5. **Audit Logger** (src/middleware/auditLogger.js):
    - Automatic audit trail creation
    - Captures all mutations
    - Non-blocking asynchronous logging

Security Features

- JWT-based stateless authentication
- Bcrypt password hashing with salt rounds (10)
- Role-based access control (RBAC)
- Schema-level data isolation
- Automatic audit logging of all mutations
- Input validation using Joi schemas
- SQL injection protection via parameterized queries
- Non-root Docker user for containerization

Database Migrations

Public Schema Migrations

```
npm run migrate:public       # Run migrations
npm run migrate:public:down  # Rollback migrations
```

Tenant Schema Migrations

```
npm run migrate:tenant up <schema_name>  # Run on specific tenant
npm run migrate:tenant up all            # Run on all tenants
npm run migrate:tenant down <schema_name> # Rollback on specific tenant
```

Seeding

```
npm run seed:super-admin     # Create initial Super Admin
npm run setup                # Run public migrations + seed
```

Project Structure

```
housingram-backend/
├─── src/
│    ├─── config/
│    │    ├─── db.js              # Database configuration
│    │    ├─── roles.js           # RBAC configuration
│    │    ├─── logger.js          # Winston configuration
│    │    └─── swagger.js         # Swagger configuration
│    ├─── controllers/
│    │    └─── superAdminStats.controller.js
│    ├─── middleware/
│    │    ├─── auth.js            # JWT authentication
│    │    ├─── tenantResolver.js  # Schema resolution
│    │    ├─── rbac.js            # RBAC
│    │    ├─── validate.js        # Validation
│    │    ├─── auditLogger.js     # Audit logging
│    │    └─── requestLogger.js   # HTTP logging
│    ├─── migrations/
│    │    ├─── public/            # Public schema migrations
│    │    ├─── tenant/            # Tenant schema migrations
│    │    ├─── umzug-public.js
│    │    └─── umzug-tenant.js
│    ├─── modules/
│    │    ├─── tenants/           # Tenant management
│    │    ├─── users/             # User management
│    │    ├─── projects/          # Project management
│    │    ├─── units/             # Unit management
│    │    └─── auditLogs/         # Audit log retrieval
│    ├─── routes/
│    │    └─── index.js           # Main router
│    └─── utils/
│         ├─── errors.js          # Custom error classes
│         ├─── response.js        # Response utilities
│         └─── schemaManager.js   # Schema management
├─── scripts/
│    └─── seed-super-admin.js     # Super Admin seeder
├─── logs/                       # Winston log files
├─── Dockerfile                  # Docker build configuration
├─── docker-compose.yml          # Docker orchestration
├─── .dockerignore               # Docker build exclusions
├─── server.js                   # Application entry point
└─── package.json                # Dependencies and scripts
```

Summary

Housingram Backend provides a complete, production-ready multi-tenant property management system with:

Core Features

- 5 complete modules (Tenants, Users, Projects, Units, Audit Logs)
- 40+ API endpoints

- Complete RBAC with 4 roles
- Automatic audit trail
- Schema-per-tenant isolation
- Industry-standard architecture

Developer Experience

- Comprehensive documentation
- Interactive Swagger API documentation
- Docker containerization for easy setup
- Postman collection for API testing

Production Ready

- Winston logging with rotation
- Docker deployment with health checks
- Cloud deployment guides
- Security best practices
- Error handling and validation
- Database migrations with version control

The system is ready for deployment and can handle multiple tenants with complete data isolation, proper access control, comprehensive logging, and easy deployment options!

For detailed guides, see:

- README.md - Main documentation
- DEPLOYMENT.md - Deployment guide
- LOGGING_GUIDE.md - Winston logging
- POSTMAN_TESTING_GUIDE.md - API testing
- IMPLEMENTATION_SUMMARY.md - Technical details

Interactive API Documentation: http://localhost:3000/api-docs

---

Document Version: 1.0
Last Updated: October 2024