Tenancy Guide for Operation Management System (OMS)

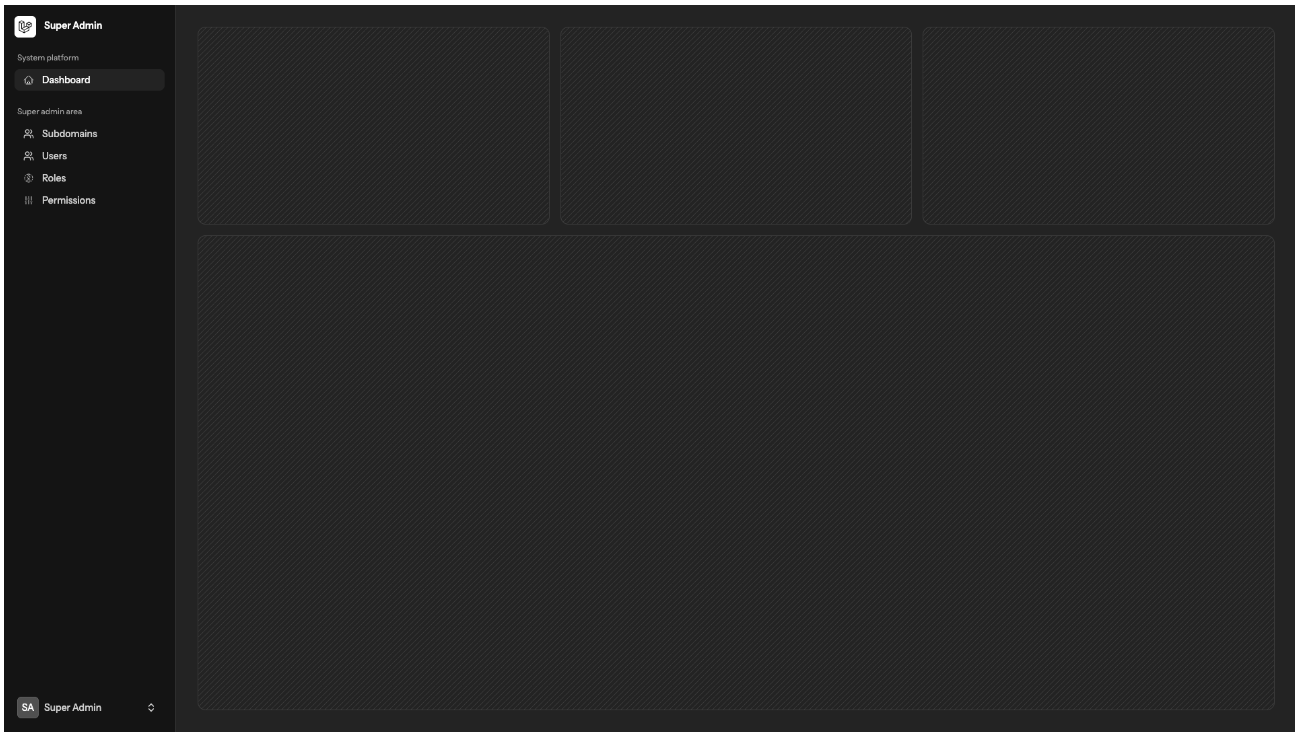
Table of Contents

Introduction

This guide explains how multi-tenancy works within the Operation Management System (OMS). It's designed to help both end users and managers understand how the system organizes and separates data for different organizations while sharing the same application infrastructure.

Central Dashboard

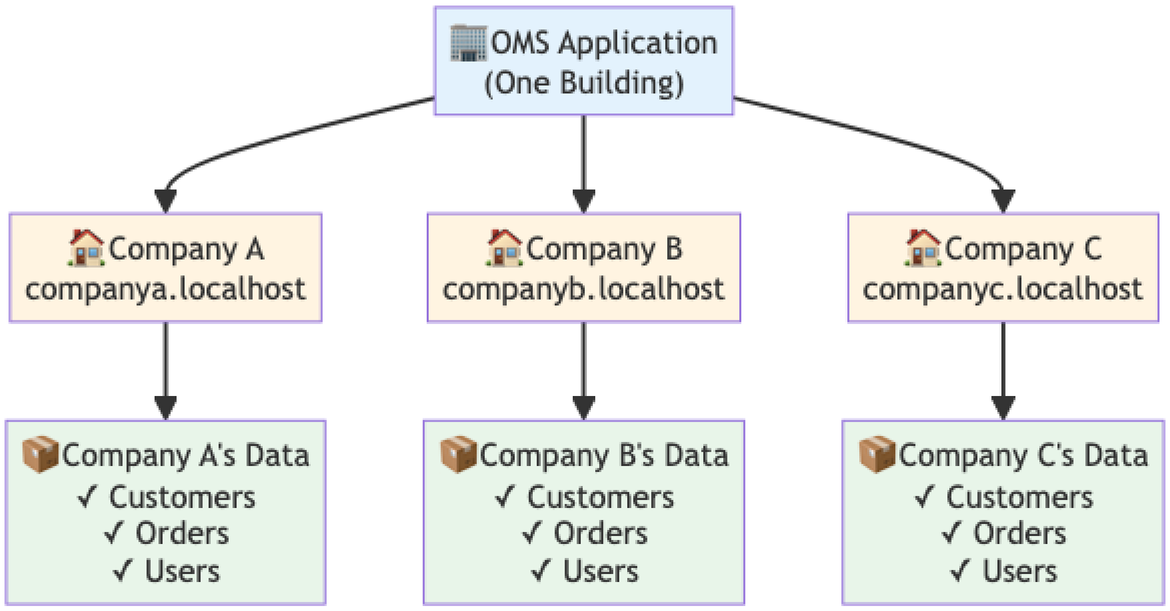
The central dashboard is where system administrators manage tenants and subdomains:



What is Tenancy?

Tenancy (or multi-tenancy) is an architecture where a single instance of the OMS application serves multiple organizations (called "tenants"). Each tenant's data is isolated from others, ensuring security and privacy while allowing efficient resource sharing.

Think of it like an apartment building: multiple families (tenants) live in the same building (application), but each has their own private space with locked doors.


🏢OMS Application (One Building)
🏠Company A companya.localhost
🏠Company B companyb.localhost
🏠Company C companyc.localhost
📦Company A's Data ✓ Customers ✓ Orders ✓ Users
📦Company B's Data ✓ Customers ✓ Orders ✓ Users
📦Company C's Data ✓ Customers ✓ Orders ✓ Users

How Tenancy Works in OMS

OMS uses a **single-database, multi-tenant architecture** with **subdomain-based tenant identification**. Here's what that means:

Key Concepts

1. **Central Domain**: The main application domain (e.g., `localhost`)
2. **Subdomains**: Each tenant gets a unique subdomain (e.g., `company1.localhost`, `company2.localhost`)
3. **System Users**: Users who manage tenants and have access to the central application
4. **Tenant Users**: Users who belong to one or more tenants and access tenant-specific areas

Architecture Components

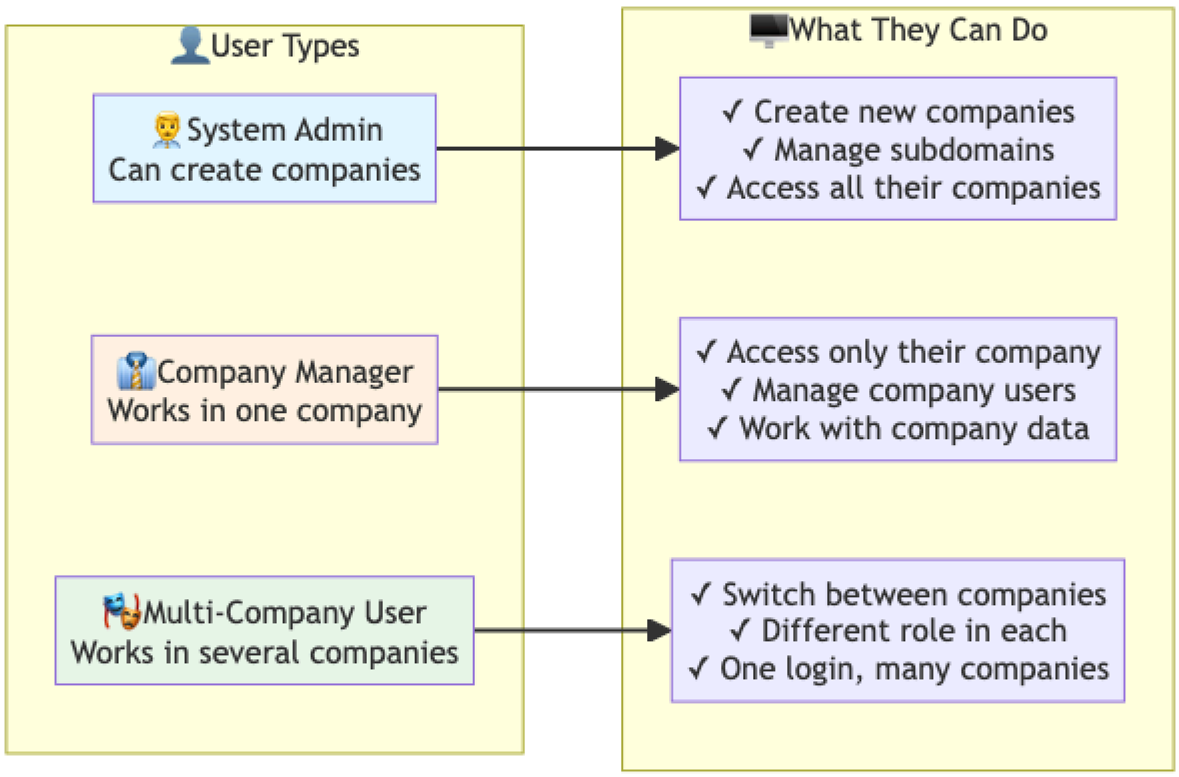
```
Central Application (localhost)
├── System Admin creates tenants
├── Manages subdomains
└── Oversees tenant users

Tenant Applications (subdomain.localhost)
├── Company-specific login
├── Isolated data access
├── Tenant-specific users
└── Role-based permissions
```

User Roles and Access Levels

User Types

OMS recognizes three types of users through the `users` table attributes:



1. System Users (`is_system = true`)

- Created in the central application
- Can create and manage subdomains/tenants
- Have a unique `system_id` that groups their tenants
- Access routes: `subdomains.index` , `subdomains.create` , `subdomains.edit`

2. Tenant Users (`is_tenant = true`)

- Associated with one or more tenants via `tenant_user` pivot table
- Can only access tenant-specific areas through subdomains
- Login through tenant subdomain (e.g., `company1.localhost/login`)
- Data access is automatically scoped to their tenant

3. Dual Role Users (`is_system = true` AND `is_tenant = true`)

- Can access both central application and tenant areas
- Created when a system user creates their first subdomain
- Can switch between system management and tenant access

User Identification

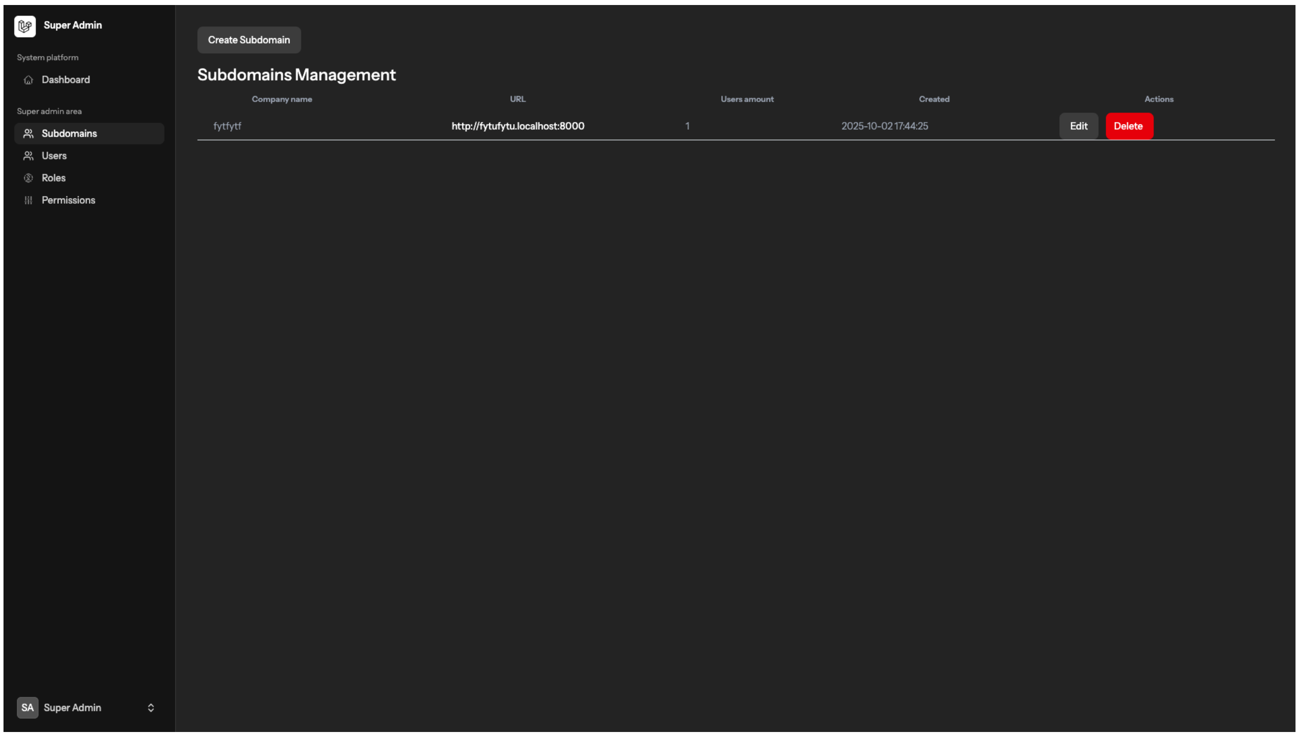
Each user has a `system_id` which:

- Groups all tenants created by the same system user
- Allows system users to only see and manage their own tenants
- Is automatically assigned on user creation (`User::getNextSystemIdOrDefault()`)

Managing Subdomains (Tenants)

Viewing Subdomains

System users can view all their subdomains from the central dashboard:



Subdomains Management

Company name	URL	Users amount	Created	Actions
fytfytf	http://fytufytu.localhost:8000	1	2025-10-02 17:44:25	Edit Delete

Location: `Subdomains → All Subdomains`

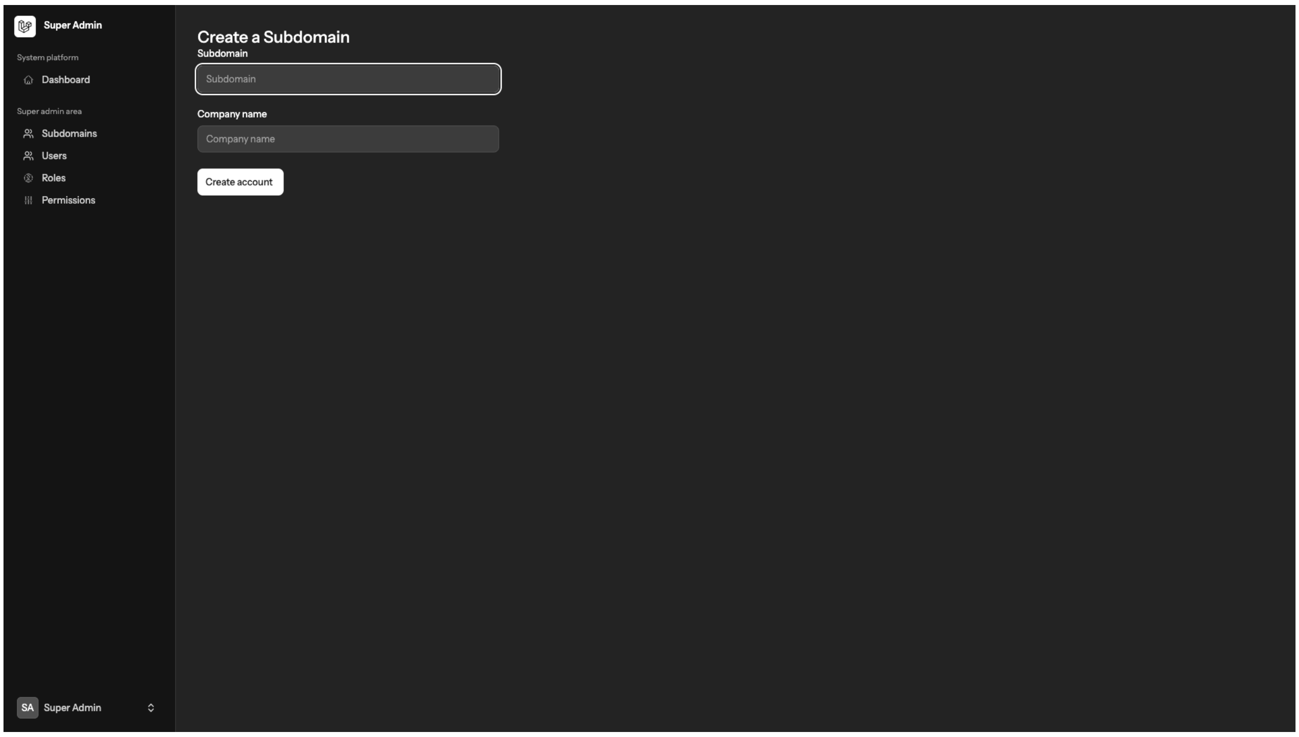
What You See:

- List of all subdomains created by your `system_id`
- Each subdomain shows:
    - Company name
    - Subdomain URL
    - Number of users
    - Creation date

Code Reference: `app/Livewire/Subdomains/Index.php:44-49`

Creating a Subdomain/Tenant

Who Can Do This: System users or admins



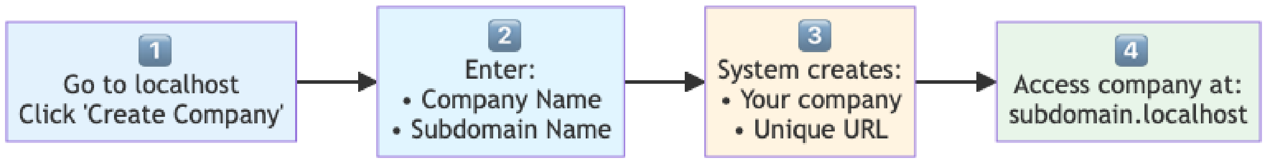
Process:

1. Navigate to `Subdomains → Create New` in the central application
2. Fill in the form:
   - **Company Name**: The tenant's organization name
   - **Subdomain**: A unique identifier (max 8 alphanumeric characters)
3. Click "Create"

What Happens Behind the Scenes:

```
// app/Livewire/Subdomains/Create.php:24-36
```

1. A new `Tenant` record is created with the company name
2. A `Domain` record is created linking to the tenant
3. The subdomain becomes `{subdomain}.localhost`
4. The creating user is automatically attached to the tenant
5. The user's `is_tenant` flag is set to `true`


1 Go to localhost Click 'Create Company'
2 Enter: • Company Name • Subdomain Name
3 System creates: • Your company • Unique URL
4 Access company at: subdomain.localhost

Editing a Subdomain

Who Can Do This: System users who created the subdomain

What You Can Edit:

- Company name
- Subdomain identifier (if not already in use)

Code Reference: `app/Livewire/Subdomains/Edit.php`

Deleting a Subdomain

Warning: This is a destructive action!

What Gets Deleted:

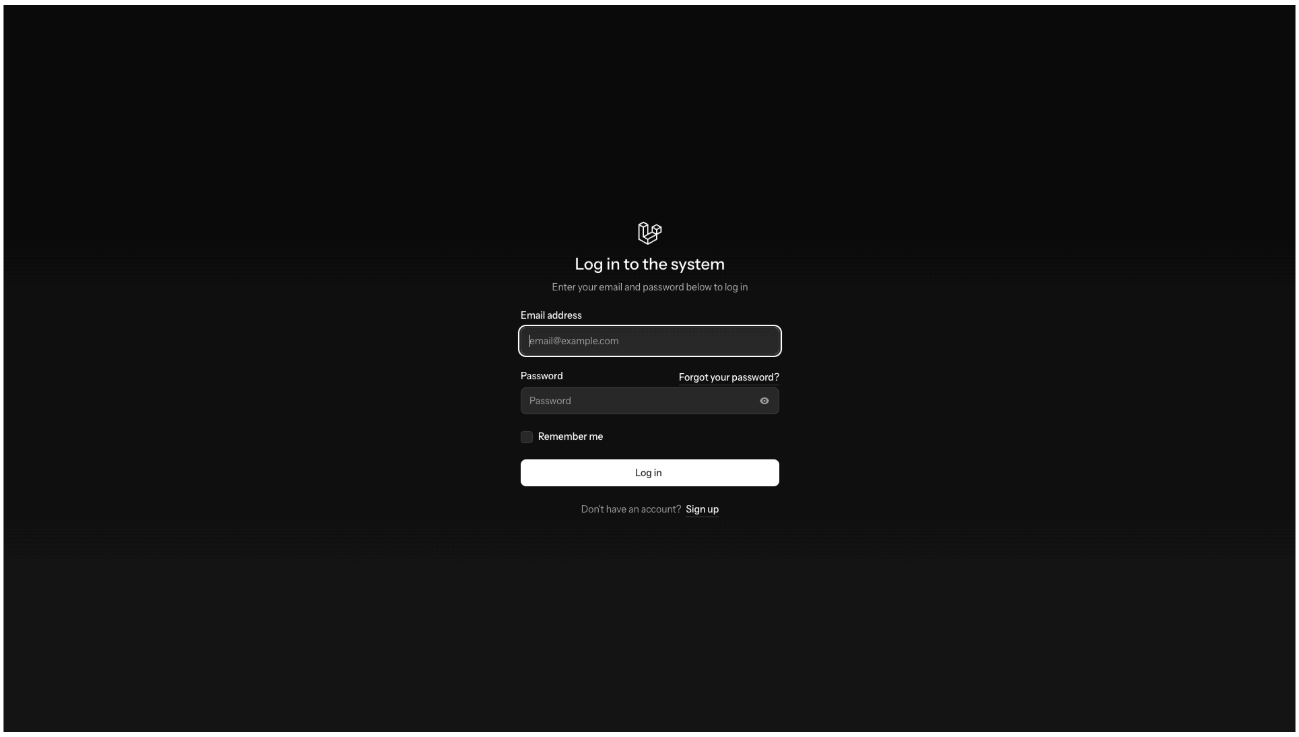
1. All user associations with the tenant (`tenant_user` pivot records)
2. The tenant record itself
3. The domain/subdomain record

Code Reference: `app/Livewire/Subdomains/Index.php:22-39`

Tenant User Management

Accessing Tenant Areas

Once a subdomain is created, tenant users can access their specific area:



Log in to the system

Enter your email and password below to log in

Email address: email@example.com

Password: Password

Adding Users to a Tenant

Location: Within a tenant subdomain → Users → Create New

Process:

1. Access the tenant subdomain (e.g., `company1.localhost`)
2. Navigate to User Management
3. Fill in user details (name, email, password)
4. Assign roles and permissions using Spatie's permission system

Key Features:

- Email uniqueness is validated **per tenant** (not globally)
- Users can belong to multiple tenants

- Each tenant manages its own user roles and permissions

Email Validation Rule: `app/Rules/UniqueEmailInTenant.php:20-31`

User-Tenant Relationships

The `tenant_user` pivot table connects users to tenants:

```
tenant_user
├── tenant_id (foreign key to tenants)
└── user_id (foreign key to users)
```

Model Relationship: `app/Models/Tenant.php:14-17` and `app/Models/User.php:90-93`

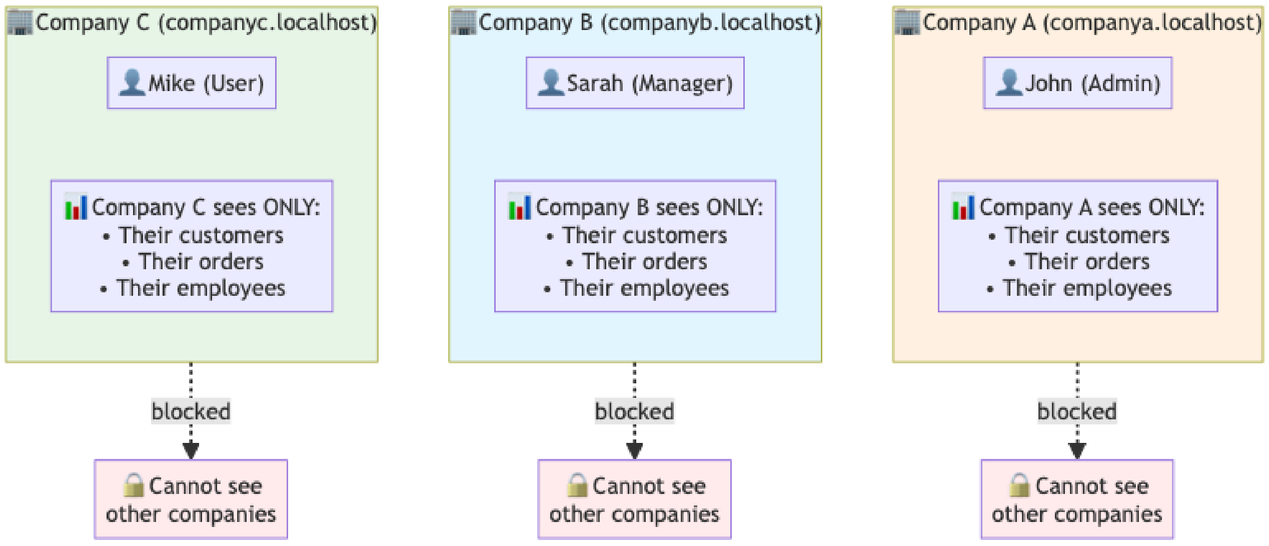
Auto-Login Feature

System users can seamlessly access tenant areas through signed URLs:

Route: `app/Http/Controllers/AuthController.php:11-17`

This allows switching from central to tenant context without re-authentication.

Data Isolation and Security


Company C (companyc.localhost)
Mike (User)
Company C sees ONLY: • Their customers • Their orders • Their employees
blocked
Cannot see other companies
Company B (companyb.localhost)
Sarah (Manager)
Company B sees ONLY: • Their customers • Their orders • Their employees
blocked
Cannot see other companies
Company A (companya.localhost)
John (Admin)
Company A sees ONLY: • Their customers • Their orders • Their employees
blocked
Cannot see other companies

How Data is Isolated

OMS uses multiple layers to ensure tenant data isolation:

1. Domain-Based Tenancy Initialization

```
// routes/tenant.php:23-27
Route::middleware([
    'web',
    InitializeTenancyByDomain::class,
    PreventAccessFromCentralDomains::class,
])
```

What This Does:

- `InitializeTenancyByDomain` : Identifies the tenant from the subdomain
- `PreventAccessFromCentralDomains` : Blocks access from `localhost` to tenant routes

2. Tenant Scope

The `TenantScope` automatically filters queries:

```
// app/Models/Scopes/TenantScope.php:14-17
public function apply(Builder $builder, Model $model): void
{
    $model->tenants()->where('tenant_id', tenant('id'));
}
```

3. System ID Filtering

Subdomains are filtered by `system_id` :

```
// app/Livewire/Subdomains/Index.php:44-49
$subdomains = Domain::with(['tenant'])
    ->where('system_id', auth()->user()->system_id)
    ->orderBy('created_at', 'desc')
    ->paginate(10);
```

Tenancy Bootstrappers

Configuration: `config/tenancy.php:30-36`

Active bootstrappers:

- **CacheTenancyBootstrapper**: Isolates cache per tenant (tagged with `tenant_{id}`)
- **FilesystemTenancyBootstrapper**: Isolates file storage per tenant
- **QueueTenancyBootstrapper**: Ensures queued jobs maintain tenant context

Note: DatabaseTenancyBootstrapper is disabled, indicating a single-database architecture.

Routes Separation

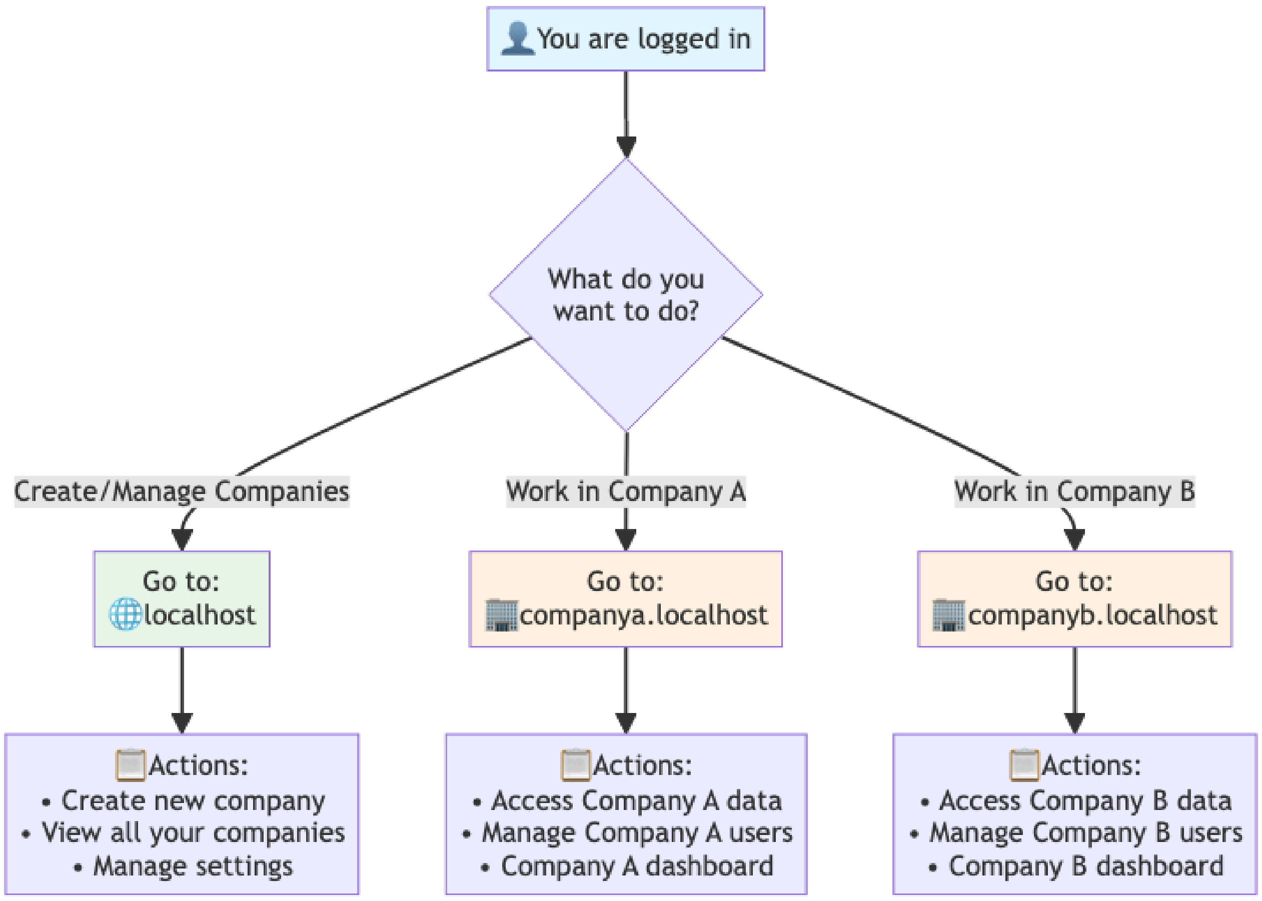
Central Routes (`routes/web.php`):

- Subdomain management
- User management (central)
- Roles & permissions (central)
- Settings

Tenant Routes (`routes/tenant.php`):

- Tenant-specific login/register
- Tenant dashboard
- Tenant-specific settings
- Application features scoped to tenant

Switching Between Tenants


👤You are logged in
What do you want to do?
Create/Manage Companies
Work in Company A
Work in Company B
Go to: 🌐localhost
Go to: 🏢companya.localhost
Go to: 🏢companyb.localhost
📋Actions: • Create new company • View all your companies • Manage settings
📋Actions: • Access Company A data • Manage Company A users • Company A dashboard
📋Actions: • Access Company B data • Manage Company B users • Company B dashboard

For System Users

1. **Access Central Application**: `localhost/dashboard`
2. **View Subdomains**: Navigate to `Subdomains`
3. **Select Tenant**: Click on a subdomain to manage or redirect
4. **Auto-Login**: Use the redirect feature to access tenant area with current authentication

For Tenant Users

1. **Direct Access**: Navigate to `{subdomain}.localhost/login`
2. **Login**: Use tenant-specific credentials
3. **Work Within Tenant**: All actions are automatically scoped to this tenant

Switching Mechanism

Redirect Component: `app/Livewire/Subdomains/Redirect.php:12-19`

This shows all subdomains available to the current user based on their `system_id`.

Technical Architecture

Database Schema

Core Tables

tenants

- `id` : Unique tenant identifier
- `companyName` : Organization name

- Standard Laravel timestamps

domains

- `id` : Domain identifier
- `domain` : Full domain (e.g., `company1.localhost`)
- `tenant_id` : Foreign key to tenants
- `system_id` : Foreign key to user's system_id
- `user_id` : Foreign key to user who created it

users

- `id` : User identifier
- `name` , `email` , `password` : Standard auth fields
- `is_system` : Boolean - can manage tenants
- `is_tenant` : Boolean - belongs to tenants
- `system_id` : Grouping identifier for tenant isolation

tenant_user (pivot)

- `tenant_id` : Foreign key to tenants
- `user_id` : Foreign key to users

Models and Relationships

Tenant Model

```
// app/Models/Tenant.php
class Tenant extends BaseTenant implements TenantWithDatabase
{
    use HasDatabase, HasDomains;

    public function users(): BelongsToMany
}
```

Domain Model

```
// app/Models/Domain.php
class Domain extends BaseDomain
{
    public function tenant(): BelongsTo
    public function systemUser(): BelongsTo
    public function getSubdomainAttribute()
}
```

User Model

```
// app/Models/User.php
class User extends Authenticatable
{
    use HasRoles; // Spatie Permissions

    public function isSystem(): bool
    public function isTenant(): bool
    public function tenants(): BelongsToMany
    public function domains(): HasMany
}
```

Middleware Stack

Tenant Routes Priority (`app/Providers/TenancyServiceProvider.php:131-146`):

1. `PreventAccessFromCentralDomains` (highest priority)
2. `InitializeTenancyByDomain`
3. `InitializeTenancyBySubdomain`
4. Other tenancy middleware

Event Lifecycle

Tenant Creation (`app/Providers/TenancyServiceProvider.php:24-38`):

- `CreatingTenant` → Before tenant is created
- `TenantCreated` → After creation (can run jobs like DB migration, seeding)

Tenant Deletion (`app/Providers/TenancyServiceProvider.php:43-50`):

- `DeletingTenant` → Before deletion
- `TenantDeleted` → After deletion (runs `DeleteDatabase` job if configured)

Tenancy Initialization (`app/Providers/TenancyServiceProvider.php:70-82`):

- `InitializingTenancy` → Before tenancy is set up
- `TenancyInitialized` → Triggers `BootstrapTenancy` listener
- `TenancyBootstrapped` → All bootstrappers have run
- `EndingTenancy` → When switching back to central context
- `TenancyEnded` → Triggers `RevertToCentralContext` listener

Best Practices

For System Administrators

1. **Subdomain Naming**:
    - Use clear, memorable names (max 8 characters)
    - Alphanumeric only (no special characters)
    - Consider using abbreviations for long company names
2. **Tenant Management**:
    - Review tenant list regularly
    - Archive or delete inactive tenants to maintain performance
    - Document which tenants are active/testing/production
3. **User Assignment**:
    - Only add users to tenants they need access to
    - Use the `system_id` to keep your tenants organized
    - Remember: deleting a tenant removes all user associations

For Tenant Managers

1. **User Management**:
    - Assign appropriate roles using Spatie permissions
    - Regular audit of user access

    - Remove users who no longer need access

2. **Data Security**:
    - Users see only data within their tenant
    - Email addresses can be reused across tenants
    - Each tenant has isolated cache, files, and queues

3. **Access Patterns**:
    - Always use the subdomain URL to access tenant areas
    - Bookmark `{subdomain}.localhost/dashboard` for quick access
    - Use auto-login features when switching from central app

For Developers

1. **Tenant Context**:
    - Always check `tenant('id')` to get current tenant
    - Use `tenancy()` helper for tenant instance
    - Apply `system_id` filtering for central app queries

2. **Testing**:
    - Test both central and tenant routes separately
    - Verify data isolation between tenants
    - Test auto-login and switching mechanisms

3. **Queues and Jobs**:
    - Queue jobs automatically maintain tenant context
    - Use `QueueTenancyBootstrapper` to ensure isolation
    - Test queued jobs in tenant context

4. **Cache and Storage**:
    - Cache is automatically tagged per tenant
    - Storage paths are suffixed with `tenant{id}`
    - Clear tenant-specific cache: `Cache::tags('tenant1')->flush()`

Frequently Asked Questions

Q: Can a user belong to multiple tenants?

A: Yes, through the `tenant_user` pivot table. A user can be associated with multiple tenants and switch between them.

Q: What happens to tenant data when a tenant is deleted?

A: The tenant record, domain, and all user associations are deleted. However, the actual users remain in the system (only the `tenant_user` pivot records are removed).

Q: Can two tenants have users with the same email address?

A: Yes, email uniqueness is validated per tenant using `UniqueEmailInTenant` rule, not globally.

Q: How do I access a tenant area as a system admin?

A: System admins can use the auto-login feature from the subdomains list in the central application.

Q: Are permissions tenant-specific?

A: Yes, OMS uses Spatie Laravel Permission, which can be scoped per tenant. Each tenant manages its own roles and permissions.

Q: What's the difference between `system_id` and `tenant_id` ?

A:

- `system_id` : Groups tenants created by the same system user (for central app filtering)
- `tenant_id` : The unique identifier for each tenant organization

Q: Can I customize the central domain?

A: Yes, edit `config/tenancy.php` key `central_domains` to change from `localhost` to your production domain.

Support and Further Information

For technical support or questions about OMS tenancy:

1. Review the code references provided throughout this guide
2. Check test files for usage examples:
   - `tests/Feature/Livewire/Subdomains/`
   - `tests/Feature/Livewire/Users/`
3. Consult Laravel Tenancy documentation: https://tenancyforlaravel.com/
4. Review Spatie Permission docs: https://spatie.be/docs/laravel-permission/

Document Version: 2.0 **Last Updated**: 2025-10-11 **Application**: Operation Management System (OMS)