

Sistema Multiempresa

KAVAC - ERP para la gestión organizacional

CENDITEL

<http://conocimientolibre.cenditel.gob.ve/licencia-de-software-v-1-3/>

Table of contents

1. Introducción	3
2. Requerimientos del sistema	4
2.1 Requerimientos Funcionales	4
2.2 Requerimientos No Funcionales	8
3. Diagramas	9
3.1 Diagrama de Arquitectura General	9
3.2 Diagrama de Flujo de Autenticación	9
3.3 Diagrama de Casos de Uso	9
3.4 Diagrama Entidad-Relación Simplificado	10
4. Casos de uso	11
5. Propuesta técnica	13
5.1 Arquitectura multitenant	13
5.2 Gestión de bases de datos.	13
6. Limitaciones de multitenancy en Laravel	15
6.1 1. Dificultades en la gestión de datos	15
6.2 2. Escalabilidad	15
6.3 3. Configuración y mantenimiento	15
6.4 4. Limitaciones en la personalización	15
6.5 5. Seguridad	15
7. Intercambio de datos entre clientes (Tenants)	16
7.1 Razones para compartir datos entre clientes	16
7.2 Cómo compartir datos de forma segura	16
8. Paquete Laravel multitenancy	18
9. Implementación de un sistema multitenancy	19
9.1 Crear un nuevo proyecto	19
9.2 Starter Kits	19
9.3 Iniciar el servidor de desarrollo local de Laravel	19
9.4 Configuración del entorno	19
9.5 Instalar tenancy for Laravel	19
9.6 Archivo de rutas de Tenants	20
9.7 Migraciones	21
9.8 Acceder al sistema	21
9.9 Personalización de base de Datos	21
9.10 Repositorio	22
9.11 Como acceder a un tenant desde la app central	22

1. Introducción

Objetivo

El presente documento tiene como objetivo establecer los requerimientos funcionales y no funcionales necesarios para la implementación de una arquitectura **multiempresa** en el sistema ERP KAVAC. Esta funcionalidad permitirá la gestión centralizada de las operaciones del grupo empresarial, al tiempo que garantizará el acceso segregado e independiente a la información y funcionalidades específicas de cada empresa o sucursal, dentro de todos los módulos del sistema.

Alcance

La funcionalidad multiempresa se desarrollará sobre la versión **2.0.0 del sistema ERP KAVAC**, abarcando de forma integral toda la plataforma y sus componentes. Esto incluye los procesos y operaciones ejecutados en cada uno de los módulos del sistema, asegurando que la experiencia del usuario y la lógica de negocio respeten el aislamiento y la identidad individual de cada empresa gestionada.

Definiciones

A continuación, se describen los términos clave y siglas utilizadas en este documento para facilitar su comprensión:

Término/Sigla	Definición
ERP	Enterprise Resource Planning. Sistema de planificación de recursos empresariales.
Multitenant	Arquitectura de software que permite que una única instancia de la aplicación sirva a múltiples clientes (tenants) con aislamiento de datos.
Tenant	Cliente individual (empresa o sucursal) que opera dentro de una instancia multitenant del sistema.
KAVAC	Nombre del sistema ERP objeto del presente documento.
Módulo	Componente funcional del sistema ERP (ej. compras, ventas, inventario, contabilidad, etc.).

2. Requerimientos del sistema

2.1 Requerimientos Funcionales

ID	Nombre del Requerimiento	Descripción	Módulo	Prioridad	Dependencias
RF-01	Gestión de Empresas (Tenants)	Permitir crear, editar, eliminar y listar empresas dentro del sistema, cada una con su configuración independiente.	Aplicación Base	Alta	Ninguna
RF-02	Aislamiento de Datos por Empresa	Garantizar que cada empresa visualice solo su propia información en todos los módulos.	Todos	Alta	RF-01
RF-03	Switch de Empresa Activa	Permitir que un usuario con acceso a múltiples empresas desde la aplicación base pueda cambiar de empresa activa desde la interfaz.	Aplicación Base	Alta	RF-01, RF-02
RF-04	Registro de Usuarios Multitenant	Permitir asignación automática de permisos del administrador de la instancia central a las nuevas empresas creadas.	Aplicación Base	Alta	RF-01
RF-05	Control de Roles y Permisos por Empresa	Gestionar roles y permisos para cada empresa de forma independiente.	Aplicación Base	Alta	RF-04

ID	Nombre del Requerimiento	Descripción	Módulo	Prioridad	Dependencias
RF-06	Asignación de permisos	Permitir la asignación de permisos a usuarios desde la instancia central a cada uno de las empresas.	Aplicación Base	Alta	RF-01, RF-04
RF-07	Registro de Ventas por Empresa	Registrar operaciones de ventas por empresa, aisladas de otras empresas.	Ventas	Media	RF-01, RF-02
RF-08	Reportes Filtrados por Empresa	Generar reportes de compras, ventas, inventario, contabilidad, etc., filtrados por empresa.	Reportes	Alta	RF-06, RF-07
RF-09	Gestión de Inventario por Empresa	Manejar inventarios separados por empresa, con control individual de almacenes, productos y existencias.	Inventario	Alta	RF-01, RF-02
RF-10	Auditoría por Tenant	Registrar logs de acciones realizadas en el sistema, identificando usuario, acción y empresa correspondiente.	Seguridad	Alta	Todos

ID	Nombre del Requerimiento	Descripción	Módulo	Prioridad	Dependencias
RF-11	Configuración de Parámetros por Empresa	Permitir a cada empresa definir sus propios parámetros operativos: moneda, impuestos, unidades, etc.	Configuración	Alta	RF-01
RF-12	Gestión de Sucursales	Posibilidad de que cada empresa tenga múltiples sucursales con inventario, personal y operaciones separadas.	Administración	Media	RF-01
RF-13	Módulo de Facturación por Empresa	Emitir facturas fiscales, electrónicas o manuales para cada empresa, con su propio formato, numeración y control.	Facturación	Alta	RF-01, RF-07
RF-14	Gestión de Proveedores por Empresa	Registrar, consultar y administrar proveedores de forma independiente por empresa.	Compras	Media	RF-01
RF-15	Gestión de Clientes por Empresa	Registrar, consultar y administrar clientes de forma aislada por empresa.	Ventas	Media	RF-01

ID	Nombre del Requerimiento	Descripción	Módulo	Prioridad	Dependencias
RF-16	Control de Accesos por Módulo	Definir a qué módulos puede acceder cada usuario según su empresa y rol asignado.	Seguridad	Alta	RF-04, RF-05
RF-17	Multilenguaje	Posibilidad de visualizar la interfaz en distintos idiomas configurables por usuario o empresa.	UI General	Baja	RF-11
RF-18	Integración con API Externa por Empresa	Integrar servicios externos (facturación, pago, logística) de forma específica para cada empresa, con sus claves y endpoints propios.	Integraciones	Media	RF-01
RF-19	Exportación de Datos por Empresa	Permitir exportar información en formatos CSV, Excel o PDF, con datos únicamente de la empresa activa.	Reportes	Alta	RF-08
RF-20	Dashboard Personalizado por Empresa	Cada empresa debe poder visualizar su propio tablero de control con indicadores y KPIs relevantes.	Reportes/UI	Alta	RF-02, RF-08

2.2 Requerimientos No Funcionales

ID	Nombre del Requerimiento	Descripción	Prioridad
RNF-01	Escalabilidad Horizontal	El sistema debe permitir escalar horizontalmente para soportar múltiples empresas sin afectar el rendimiento.	Alta
RNF-02	Seguridad en el Aislamiento de Datos	Debe garantizarse el aislamiento total de datos entre empresas, evitando accesos cruzados no autorizados.	Alta
RNF-03	Disponibilidad	El sistema debe estar disponible al menos el 99.5% del tiempo mensual.	Alta
RNF-04	Soporte Multinavegador	El sistema debe funcionar correctamente en los navegadores modernos: Chrome, Firefox, Edge y Safari.	Media
RNF-05	Tiempos de Carga	Las páginas principales deben cargar en menos de 3 segundos bajo condiciones normales de uso.	Alta
RNF-06	Compatibilidad Móvil	El sistema debe ser accesible desde dispositivos móviles con una interfaz responsiva.	Media
RNF-07	Logs Centralizados	Toda actividad crítica debe quedar registrada en un sistema de logs centralizado con trazabilidad por empresa.	Alta
RNF-08	Backup Diario	Se debe realizar una copia de seguridad completa de los datos diariamente y por cada empresa.	Alta
RNF-09	Configurabilidad	El sistema debe permitir parametrizar ciertos comportamientos por empresa (moneda, idioma, zona horaria).	Media
RNF-10	Cumplimiento Legal	El sistema debe cumplir con la legislación vigente en materia de protección de datos (ej. GDPR, LOPD).	Alta

3. Diagramas

Los diagramas permiten visualizar de forma clara y estructurada cómo funciona el sistema KAVAC ERP a nivel arquitectónico, funcional y de interacción entre actores. Esta sección incluye los siguientes tipos:

3.1 Diagrama de Arquitectura General

- Usuarios acceden al sistema a través de un portal central.
- Middleware identifica el tenant (empresa) por subdominio, token o cabecera.
- Se establece conexión a la base de datos correspondiente.
- Cada tenant tiene datos y configuraciones aisladas.
- Servicios externos pueden integrarse por tenant.

[Diagrama aquí - puede representarse como texto si no tienes herramienta gráfica]

Cliente → Frontend Vue/Blade → Middleware Tenant → Backend Laravel → DB por Tenant

Middleware Tenant → Servicios API

DB por Tenant → Backup / Logs

3.2 Diagrama de Flujo de Autenticación

1. Usuario accede con su correo y contraseña.
2. El sistema verifica en qué empresa(s) tiene acceso.
3. Si tiene acceso a una sola, se inicia sesión directamente.
4. Si tiene múltiples accesos, se muestra pantalla para seleccionar empresa activa.
5. Se crea la sesión con el contexto del tenant.

[Flujo representado como texto simple o pseudodiagrama]

Inicio Sesión

|

Validar Credenciales

├─ Usuario tiene 1 tenant → Establecer sesión

└─ Usuario tiene múltiples → Mostrar selector → Establecer sesión

3.3 Diagrama de Casos de Uso

Actores Principales: - Administrador del sistema - Usuario de empresa - Gerente de empresa - API externa (facturación, pagos)

Casos de uso clave: - Gestionar empresas - Gestionar usuarios y roles - Registrar compras/ventas - Visualizar reportes - Cambiar empresa activa - Configurar parámetros por tenant

[Representación textual tipo lista, si no hay gráfico disponible]







Administrador: - Crear/editar empresas - Asignar usuarios a empresas - Ver logs de auditoría

Usuario de empresa: - Registrar compras/ventas - Gestionar inventario - Ver reportes

Gerente: - Ver dashboard - Descargar reportes financieros

API externa: - Consultar datos - Enviar documentos electrónicos

3.4 Diagrama Entidad-Relación Simplificado

Tenant  Empresa
 Usuario (con roles por tenant)
 Configuración
 Inventario
 Clientes / Proveedores
 Facturas / Compras / Ventas

4. Casos de uso

(Ejemplo de caso de uso)

Caso de Uso: Consultar información de una sucursal

Actor: Administrador

Precondiciones: El usuario ha iniciado sesión en el sistema con sus credenciales válidas. El usuario tiene los permisos necesarios para consultar la información de la sucursal.

Postcondiciones: El usuario ha visualizado la información solicitada de la sucursal.

Flujo principal:

1. El usuario selecciona la opción "Consultar sucursales" en el menú principal.
2. El sistema presenta una lista de sucursales, filtrable por empresa y otros criterios (nombre, ciudad, etc.).
3. El usuario selecciona la sucursal deseada.
4. El sistema muestra la información detallada de la sucursal, incluyendo:
 - Nombre de la sucursal
 - Dirección completa
 - Datos de contacto (teléfono, correo electrónico)
 - Gerente de la sucursal
 - Empresa a la que pertenece
 - Estado (activa/inactiva)
 - Fecha de creación
 - Otros datos relevantes (ej: número de empleados, ventas totales, etc.)
5. El usuario puede regresar al menú principal o realizar otras acciones.

Flujos alternativos:

- **Usuario sin permisos:** Si el usuario no tiene los permisos necesarios, el sistema muestra un mensaje de error y no permite acceder a la información.
- **Sucursal no encontrada:** Si la sucursal seleccionada no existe, el sistema muestra un mensaje de error.
- **Error de conexión a la base de datos:** Si ocurre un error al conectar con la base de datos, el sistema muestra un mensaje de error e intenta reconectar.

Excepciones:

- Ninguna.

Requisitos no funcionales:

- El sistema debe ser rápido y eficiente en la consulta de información.
- La interfaz de usuario debe ser intuitiva y fácil de usar.
- La información debe ser presentada de forma clara y organizada.

5. Propuesta técnica

5.1 Arquitectura multitenant

Multitenant es un modelo de **arquitectura de software** que permite que una única instancia de una aplicación atienda a múltiples clientes. En otras palabras, un solo conjunto de código puede atender a varios usuarios, manteniendo la información confidencial de cada uno aislada y accesible únicamente por ellos.

Cada cliente del servicio se considera un tenant, lo que posibilita la personalización de ciertos elementos de la aplicación, como los colores de la interfaz, aunque no se modifica el código en sí. Es relevante destacar que, en esta arquitectura, un cliente no tiene que ser un único usuario, sino que puede representar a un grupo de usuarios. Por ejemplo, algunos servicios operan con equipos o grupos, donde un cliente puede disponer de un subdominio para acceder a la aplicación, permitiendo que múltiples usuarios visualicen la información asociada a ese cliente.

5.1.1 Ventajas

- Economía en desarrollo y mantenimiento, ya que los costos son distribuidos entre todos los clientes.
- Fácil actualización, ya que es necesario solo actualizar una sola instancia.
- Seguridad de la información de cada cliente, ya que cuenta con un schema separado para cada uno.
- Optimiza el uso de recursos de los servidores.

5.1.2 Desventajas

- Dificulta el desarrollo de características específicas para un cliente.
- Único punto de falla: si la aplicación tiene un error o falla, fallará para todos los clientes.

5.2 Gestión de bases de datos.

Arquitecturas en las bases de datos.

En Postgres un Schema actúa como un nombre de espacio, lo que permite una organización/separación a la base de datos. (Base de datos -> Schema -> Tabla)

5.2.1 Compartida (Shared)

Una Base de Datos - Un Schema

Esta arquitectura es la que se usa por defecto en la mayoría de aplicaciones, se separa los usuarios de la aplicación por una relación con la tabla de usuarios.

Ventajas

- La capa de datos es fácil de construir (crear las tablas de la base de datos).
- Todos los usuarios usan el mismo dominio de la aplicación.

Desventajas

- Es costoso (Tamaño y cantidad de peticiones a la base de datos).

5.2.2 Separada (Isolated)

Cada cliente (tenant) tiene su propia base de datos

Esta arquitectura permite usar cualquier motor de base de datos, se puede relacionar como el proceso de virtualización en el que cada instancia se implementa para el nuevo cliente.

Ventajas

- Mayor seguridad.
- Fácil recuperación en caso de daño en la base de datos
- Bajo consumo de recursos de procesamiento

Desventajas

- Dificultad para escalar.
- Costoso en recursos de almacenamiento (se necesitan muchas bases de datos).
- Dificultad para compartir información entre clientes.
- Dificultad de actualización (hay que hacer el proceso por cada base de datos).

5.2.3 Híbrida (Semi - Isolated)

Una base de datos - Múltiples Schemas

Esta arquitectura usa lo mejor de las dos anteriores y es posible usarla en PostgreSQL. Esto permite, por ejemplo, que cada uno de los clientes tenga usuarios y sus datos estén separados por cada schema.

Ventajas

- Fácilmente escalable.
- Reducción de costos en almacenamiento y procesamiento.
- Seguro (cada cliente tiene su información separada en el schema).
- Compartir información entre clientes (hay tablas que se pueden acceder por todos los clientes).

Desventajas

- El proceso de recuperación de datos de un schema es más complejo.
- Es menos seguro que las bases de datos separadas.

6. Limitaciones de multitenancy en Laravel

6.1 1. Dificultades en la gestión de datos

- Aislamiento de datos: Asegurarte de que los datos de un inquilino no se mezclen con los de otro puede ser complicado, especialmente si no se implementa correctamente.
- Consultas complejas: Realizar consultas que involucren datos de múltiples inquilinos puede ser más difícil y requerir lógica adicional.

6.2 2. Escalabilidad

- Rendimiento: A medida que el número de inquilinos crece, el rendimiento puede verse afectado si no se optimizan las consultas y la estructura de la base de datos.
- Recursos compartidos: Los inquilinos comparten los mismos recursos del servidor, lo que puede llevar a problemas de rendimiento si uno de ellos consume demasiados recursos.

6.3 3. Configuración y mantenimiento

- Complejidad en la configuración: Configurar un sistema multitenant puede ser más complejo que un sistema monolítico, especialmente en términos de rutas, middleware y configuración de base de datos.
- Actualizaciones: Las actualizaciones del sistema pueden ser más complicadas, ya que debes asegurarte de que todos los inquilinos se vean afectados de manera adecuada.

6.4 4. Limitaciones en la personalización

- Personalización de inquilinos: Si necesitas que cada inquilino tenga características o configuraciones únicas, esto puede requerir un esfuerzo adicional en la implementación.
- Dependencias de terceros: Algunas bibliotecas o paquetes pueden no ser compatibles con un enfoque multitenant, lo que limita las opciones disponibles.

6.5 5. Seguridad

- Vulnerabilidades: Si no se implementa correctamente, puede haber riesgos de seguridad, como la exposición accidental de datos entre inquilinos.
- Autenticación y autorización: Manejar la autenticación y autorización de manera efectiva para múltiples inquilinos puede ser más complicado.

7. Intercambio de datos entre clientes (Tenants)

Es posible compartir datos entre clientes en un sistema multitenant en Laravel, pero **no es la práctica recomendada** y debe hacerse con mucho cuidado y con una justificación clara. Por lo general, la principal ventaja del multitenancy es precisamente el aislamiento de datos.

7.1 Razones para compartir datos entre clientes

7.1.1 Datos globales/comunes:

- Ejemplo: Información de referencia como catálogos de productos, listas de países, etc., que todos los clientes pueden usar.
- Consideración: Estos datos deben ser inmutables o gestionados con cuidado para evitar problemas.

7.1.2 Funcionalidades compartidas:

- Ejemplo: Un sistema de foros donde los clientes pueden interactuar entre sí.
- Consideración: Requiere una cuidadosa gestión de permisos y acceso.

7.1.3 Informes agregados:

- Ejemplo: Generar informes agregados a nivel de la empresa, donde los datos individuales de los clientes se combinan.
- Consideración: Se debe garantizar la privacidad y el anonimato de los datos individuales.

7.2 Cómo compartir datos de forma segura

7.2.1 Base de datos separada (o tablas separadas):

- Opción: Crear una base de datos (o tablas) separada para los datos compartidos.
- Beneficio: Mantiene el aislamiento de los datos de los clientes.

7.2.2 Columnas de "tenant_id" (o similar):

- Opción: Agregar una columna "tenant_id" a las tablas compartidas para identificar a qué cliente pertenece cada dato.
- Consideración: Requiere una cuidadosa gestión de las consultas para evitar fugas de datos.

7.2.3 Roles y permisos:

- Opción: Implementar un sistema de roles y permisos para controlar el acceso a los datos compartidos.
- Consideración: Asegúrate de que los permisos sean precisos y bien gestionados.

7.2.4 Anonimización de datos:

- Opción: Anonimizar los datos individuales antes de compartirlos.
- Beneficio: Protege la privacidad de los clientes.

8. Paquete Laravel multitenancy

En la comunidad de desarrollo entorno a Laravel existen varios paquetes utilizados para la implementación de sistemas multitenant entre ellos tenemos.

SaaSykit Tenancy: Este paquete es ideal para construir aplicaciones SaaS. Ofrece una estructura sólida y es fácil de usar, lo que lo convierte en una excelente opción para nuevos proyectos.

Tenancy for Laravel: Este es uno de los paquetes más populares y completos. Permite una gran flexibilidad en la gestión de bases de datos y es muy bien documentado, lo que facilita su implementación.

Spatie Laravel Multi-tenancy: Este paquete es conocido por su simplicidad y eficacia. Es perfecto si buscas una solución que se integre bien con otros paquetes de Spatie, que son muy utilizados en la comunidad de Laravel.

Tenancy Package: Este es un paquete más simple que ofrece una implementación básica de multitenancy. Es ideal si necesitas algo ligero y no requieres muchas características avanzadas.

Laratrust: Aunque no es exclusivamente un paquete de multitenancy, Laratrust puede ser útil si necesitas gestionar roles y permisos en un entorno multitenant.

Para la presente investigación se implementará un sistema multitenant usando el paquete **Tenancy for Laravel**

[Documentación Tenancy for Laravel](#)

9. Implementación de un sistema multitenancy

9.1 Crear un nuevo proyecto

Crear un nuevo proyecto utilizando el instalador de Laravel

```
Laravel new name-app
```

9.2 Starter Kits

Se puede utilizar un Starter Kit deseado ejemplo:

```
React  
Vue  
Livewire
```

Si utiliza alguno de los Starter Kit debe realizar la configuración pertinente para la gestión de rutas.

9.3 Iniciar el servidor de desarrollo local de Laravel

```
cd name-app  
npm install && npm run build  
composer run dev
```

9.4 Configuración del entorno

Para configurar el entorno de la aplicación se debe gestionar el archivo .env

Configurar base de datos

```
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=laravel  
DB_USERNAME=root  
DB_PASSWORD=
```

Si elige utilizar una base de datos que no sea SQLite, tendrá que crear la base de datos y ejecutar las migraciones de la base de datos de su aplicación:

```
php artisan migrate
```

9.5 Instalar tenancy for Laravel

```
composer require stancl/tenancy  
php artisan tenancy:install  
php artisan migrate
```

Registrar el prestador de servicios en bootstrap/providers.php:

```
return [
    App\Providers\AppServiceProvider::class,
    App\Providers\TenancyServiceProvider::class, // <-- here
];
```

Para personalizar y gestionar los requerimientos de nuestra aplicación, se requiere crear un modelo Tenant y configurar nuestro modelo con el siguiente contenido

```
<?php

namespace App\Models;

use Stanc\Tenancy\Database\Models\Tenant as BaseTenant;
use Stanc\Tenancy\Contracts\TenantWithDatabase;
use Stanc\Tenancy\Database\Concerns\HasDatabase;
use Stanc\Tenancy\Database\Concerns\HasDomains;

class Tenant extends BaseTenant implements TenantWithDatabase
{
    use HasDatabase, HasDomains;
}
```

Como se ha personalizado el Modelo, es necesario realizar una configuración en el archivo **config/tenancy.php**

```
'tenant_model' => \App\Models\Tenant::class,
```

Configurar las rutas

En el archivo de rutas realizamos un ajuste donde las rutas centrales estén registradas únicamente en dominios centrales.

```
foreach (config('tenancy.central_domains') as $domain) {
    Route::domain($domain)->group(function () {
        // your actual routes
    });
}
```

Dominios centrales

En el archivo de configuración **config/tenancy.php** se debe agregar el dominio central

```
'central_domains' => [
    'saas.test', // Add the ones that you use. I use this one with Laravel Valet.
],
```

Los valores predeterminados son :

```
'central_domains' => [
    '127.0.0.1',
    'localhost',
],
```

9.6 Archivo de rutas de Tenants

En el directorio de archivos de Laravel se encuentra el directorio **routes/tenant.php** para la configuración de rutas de los tenants.

Por defecto tenemos la siguiente configuración

```
Route::middleware([
    'web',
    InitializeTenancyByDomain::class,
    PreventAccessFromCentralDomains::class,
])->group(function () {
    Route::get('/', function () {
        return 'This is your multi-tenant application. The id of the current tenant is ' . tenant('id');
    });
});
```

Estas rutas solo serán accesibles en dominios de inquilinos (no centrales)

9.7 Migraciones

En el directorio de archivos de Laravel se incluye la carpeta de migraciones para los tenants, en el directorio **migrations/tenant**.

En este directorio se deben incluir las migraciones que consideren pertinentes para los tenants.

Para ejecutar las migraciones de los tenants ejecuta el siguiente comando:

```
php artisan tenants:migrate
```

Para crear migraciones específicas para cada tenant ejecuta el siguiente comando :

```
php artisan make:migration create_table_name_for_tenant --path=database/migrations/tenant
```

Si necesitas migrar solo un tenant específico, puedes especificarlo con la opción `--tenants`, así:

```
php artisan tenants:migrate --tenants=tenant_id
```

9.8 Acceder al sistema

Accede al sistema con el dominio de cada tenant deseado.

9.9 Personalización de base de Datos

ejemplo:

```
Route::get('prueba', function() {
    $tenant1 = \App\Models\Tenant::create([
        'id' => 'id_tenant',
        'tenancy_db_name' => 'tenantname',
        'tenancy_db_username' => 'tenant1',
        'tenancy_db_password' => 'password',
        'tenancy_db_connection' => 'sqlite'
    ])

    $tenant1->domains()->create(['domain' => 'foo.localhost']);
});
```

9.10 Repositorio

9.11 Como acceder a un tenant desde la app central

```
// Consulto el tenant al cual quiero acceder $tenant = Tenant::first()
```

```
// Le indico a laravel a través del helper que quiero acceder a ese tenant tenancy()->initialize($tenant);
```

Ejemplo:

La consulta `$user = \App\Models\User::first()` ahora devolverá el usuario del tenant al que accedí.

Si queremos finalizar la conexión lo podemos hacer de la siguiente manera:

```
tenancy()->end(); // Finalizo la conexión al tenant // Ahora las consultas volverán a ser sobre el tenant central
```

Viceversa // Acceder desde un tenant a la app central

```
// Lo hago a través de una function anonima
```

```
tenancy()->central(function () { // Aquí dentro las consultas serán sobre el tenant central $user =  
\App\Models\User::first(); });
```