# Proyecto Fuengirola 1 (JetStream)

## Comandos de inicio

### Lanzar migrations:

php artisan migrate:fresh (lanza y borra los datos de las tablas)

php artisan migrate --seed (lanza migrations y seeders juntos)

### Crear migration:

php artisan make:migration modify\_user\_id\_nullable\_in\_teams

--table=teams

### Lanzar seeders

php artisan db:seed (Lanza todos los seeders)

php artisan db:seed --class=NombreSeeder (seed especifico)

### Crear seeder:

php artisan make:seeder nombre\_seeder

## Borrado de Caches

php artisan cache:clear

php artisan config:clear

php artisan route:clear

php artisan view:clear

php artisan event:clear

php artisan session:clear

php artisan permission:cache-reset

php artisan optimize

## Borrado de datos de las Tablas

php artisan migrate:fresh

## Modelo User

### Metodo currentTeam()

**Devuelve el equipo actual al que pertenece el usuario**

public function currentTeam()

{

return $this->belongsTo(Team::class);

}

### Metodo allTeams()

**Devuelve todos los equipos al los que pertenece un usuario n**

public function allTeams()

{

return $this->belongsToMany(Team::class, 'team\_user');

}

## Ejemplo de uso

### Ejemplo de uso de relaciones en Jetstream

Gracias a Jetstream, si quieres obtener todos los usuarios de un equipo específico o asignar un equipo a un usuario, puedes hacerlo de forma sencilla:

// Obtener todos los usuarios de un equipo específico

$team = Team::find(1);

$users = $team->allUsers(); // O la relación adecuada según tu configuración

// Asignar un equipo al usuario actual

$user = auth()->user();

$user->switchTeam($team); // Jetstream proporciona este método

## Uso de Roles

### introducción

Es cierto que **Jetstream** no incluye directamente una tabla específica de roles, sino que usa un sistema simplificado de roles a través de la relación de equipos en el modelo User y el campo role en la tabla team\_user. Este campo es lo que Jetstream utiliza para asignar roles de forma básica, permitiéndote definir permisos para cada usuario en un equipo. Sin embargo, si deseas una implementación de roles más detallada, aquí te explico cómo puedes gestionar roles en Jetstream o expandir su funcionalidad.

### Cómo se gestionan los roles en Jetstream

Jetstream maneja roles mediante una **columna role** en la tabla intermedia team\_user, que almacena la relación de muchos-a-muchos entre los usuarios y los equipos. Por defecto, Jetstream define dos roles principales:

* owner (propietario): el usuario que creó el equipo.
* editor o member: cualquier otro miembro del equipo.

Para manejar estos roles, Jetstream incluye métodos en el modelo User y en el modelo Team, permitiendo verificar el rol de un usuario en un equipo determinado.

### Configuración de roles básicos en Jetstream

Algunos métodos comunes que Jetstream incluye para gestionar roles en equipos son:

* hasTeamPermission($team, $permission): verifica si un usuario tiene un permiso específico en un equipo.
* currentTeam(): obtiene el equipo actual del usuario, lo que facilita comprobar el rol en ese contexto.

### Ejemplo de cómo verificar el rol de un usuario en un equipo

Si deseas realizar alguna verificación sobre el rol de un usuario en su equipo actual, puedes hacer lo siguiente:

código

$user = auth()->user();

$team = $user->currentTeam;

if ($user->hasTeamRole($team, 'owner')) {

// El usuario es el propietario del equipo

} elseif ($user->hasTeamRole($team, 'member')) {

// El usuario es un miembro estándar del equipo

}

### Añadir más roles personalizados

Si deseas gestionar roles de forma más detallada, puedes expandir el sistema de roles de Jetstream al agregar roles adicionales en la columna role de team\_user. Luego, puedes definir permisos o capacidades específicos en base a estos roles.

1. **Migración de roles adicionales**: Puedes agregar más roles en team\_user y luego usar esta columna role para verificar permisos.
2. **Definir roles en el modelo Team o en User**: Puedes crear métodos en el modelo Team o User para validar estos roles personalizados.
3. **Método para verificar roles personalizados**: En el modelo User, podrías definir métodos para manejar roles adicionales:

php

Copiar código

public function hasTeamRole($team, $role)

{

return $this->teams()

->where('team\_id', $team->id)

->where('role', $role)

->exists();

}

### Ejemplo: Creación de un sistema de permisos más avanzado

Para un sistema de roles más avanzado, podrías considerar agregar una tabla roles y otra tabla permissions (usando, por ejemplo, **Laravel Permission Package** de Spatie), lo que permite una estructura más flexible para la gestión de permisos. Aquí, roles puede definir múltiples permisos que puedes asignar dinámicamente.

1. **Crear tablas roles y permissions**: Crear migraciones para estas tablas y configurar relaciones adicionales entre ellas y el modelo User.
2. **Uso del paquete Laravel Permission** (opcional): Este paquete permite crear permisos y roles dinámicos sin tener que escribir toda la lógica desde cero.
3. **Asignación de permisos a roles y usuarios**: Puedes asignar permisos a roles, y roles a usuarios, para luego validar permisos específicos según sea necesario.

**En resumen**

Si la funcionalidad de roles de Jetstream es suficiente para tus necesidades, entonces la columna role en team\_user será tu principal mecanismo de control. Pero si necesitas una estructura de permisos y roles más completa, puedes extender Jetstream mediante una tabla de roles y permisos, o considerar un paquete de permisos como el de **Spatie** para una solución más robusta y flexible.

## Configuración de Spatie para Roles

**Instalar**: composer require spatie/laravel-permission

**Publicar**: php artisan vendor:publish

--provider="Spatie\Permission\PermissionServiceProvider"

**Ejecutar Migraciones:** php artisan migrate

**Agregar el trait al modelo User:**

use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable

{

use HasRoles;

// ...

}

**Métodos**:

Después de esto, puedes asignar roles y permisos usando métodos como assignRole, givePermissionTo, etc.Final del formulario

## Enfoque de multi Administradores y multi Propietarios

bien mi idea es que haya al menos tres administradores que pueden ser propietarios de todos los equipos, luego habra algunos equipos que ademas todos los miembros tambien seran propietarios, y luego algunos en los que existiran miembros, propietarios y administradores, ¿que enfoque ves mas oportuno? no me des codigo aun solo dime que enfoque seria el mas oportuno y como funcionaria?

Para este caso, aquí tienes un enfoque que podría cubrir todas tus necesidades de manera flexible y escalable:

**1. Roles Flexibles con Asociación Múltiple de Usuarios a Equipos**

* Implementa un sistema de **roles y permisos personalizados** por equipo que permita tener múltiples administradores, propietarios y miembros para cada equipo.
* Esto se puede hacer usando una **tabla intermedia** que almacene el user\_id, team\_id, y role (por ejemplo, Admin, Propietario, Miembro). Esto permitiría que cada usuario tenga un rol diferente en cada equipo y que el rol pueda ser asignado dinámicamente sin limitar la cantidad de propietarios o administradores por equipo.
* **Ventaja**: El sistema será muy flexible, permitiendo que cualquier usuario tenga roles diferentes en cada equipo según sea necesario.

**2. Propiedad Distribuida por Equipo**

* Para los equipos donde todos los miembros deben ser propietarios, configura una **propiedad compartida**. Esto implica que, en lugar de un único "propietario" del equipo, cualquier miembro del equipo con un rol de Propietario puede realizar tareas de administración.
* Podrías establecer reglas en el modelo de Team para restringir o permitir ciertos permisos según el rol (Admin, Propietario, o Miembro), controlando lo que cada rol puede hacer.
* **Ventaja**: Puedes tener equipos en los que todos los miembros tienen permisos completos, o equipos con una jerarquía clara sin comprometer la propiedad.

**3. Roles Especiales y Escalables en los Equipos**

* Define roles como Admin, Propietario y Miembro dentro de cada equipo. De esta manera, en algunos equipos puedes asignar roles específicos a los usuarios (como ser sólo administrador sin ser propietario).
* Configura roles predeterminados para asignar permisos específicos dentro del equipo (por ejemplo, crear, editar, o eliminar miembros del equipo).
* **Ventaja**: Los roles se pueden asignar a medida que evolucionan las necesidades de cada equipo, manteniendo la integridad y escalabilidad.

**Funcionamiento del Sistema:**

1. **Usuarios y Equipos**: Los usuarios pueden pertenecer a varios equipos con roles distintos en cada uno.
2. **Asignación de Roles**: Cada equipo asigna roles a sus usuarios de acuerdo a sus necesidades. Puedes tener tres administradores en un equipo y sólo uno en otro.
3. **Control de Permisos**: Mediante la lógica de roles, estableces permisos específicos, asegurándote de que la administración y la propiedad sean dinámicas y seguras.

Con esta arquitectura, lograrás flexibilidad y escalabilidad en la asignación de permisos y propiedad, preservando la integridad de la configuración de Jetstream.

## Control de acceso en el Frontend

En el frontend, puedes manejar la visibilidad de los elementos según el rol del usuario en el equipo:

@if(auth()->user()->isUserOwner($team))

<!-- Mostrar acciones de propietario -->

@endif

@if(auth()->user()->isUserAdmin($team))

<!-- Mostrar acciones de administrador -->

@endif

Con esto, tendrás un sistema robusto donde los permisos y la propiedad de los equipos pueden ser administrados de forma distribuida, con la capacidad de asignar diferentes roles a los usuarios dentro de cada equipo sin comprometer la configuración estándar de Jetstream.

Otras acciones:

// Obtener todos los teams de usuario 1

$user = User::find(1);

$userTeams = $user->teams; // Obtiene una colección de equipos

// Agregar un equipo al usuario

$user->teams()->attach($teamId);

// Remover un equipo del usuario

$user->teams()->detach($teamId);

// Verificar si el usuario pertenece a un equipo específico

$isMember = $user->teams->contains($teamId);