PROYECTO UMBRELLA

ORGANIZACIÓN Y
PATRONES DE DISEÑO.

Commits semánticos.

- Raíz del mono repositorio ‘npm run co’.
  - **Fix**: Un commit de tipo fix corrige un error en la base del código
  - **Feat**: un commit de tipo feat introduce nuevas características en la base del código

Estructura carpetas.

- App
  - + Http
    - Controllers
      - + (lista módulos umbrella4)
    - Requests
      - + (lista módulos umbrella4)
  - + Models
    - (lista módulos umbrella4)
  - + Services
    - (lista módulos umbrella4)
      - + Contracts
      - + Events
      - + Interfaces
      - + Jobs
      - + Class
      - + Context
      - + Factories
- Databases
  - + Factories: siempre y cuando se vaya a generar datos faker (no aplica si tienes datos ya claros).
  - + Migrations/tenant: algunos inquilinos tienen diferentes columnas en tablas, deben separarse en carpeta, ejemplo: carpeta claro, carpeta etb.
  - + Seeder: Datos semilla base que deben existir a penas se levanta el servidor. (Claro, ETB)
- Routes
  - + api
    - (lista módulos umbrella4 con rutas .php) – estos archivos deben ser llamados dentro de la ruta padre tenant.php

Rutas

Cada módulo con su archivo de rutas.

```php
Route::middleware([
    InitializeTenancyByDomain::class,
    PreventAccessFromCentralDomains::class,
])->group(function () {
    // require base_path('routes/api.php');
    require base_path('routes/api/System/SysTyping.php');
    require base_path('routes/api/roles.php');
    require base_path('routes/api/menu.php');
    require base_path('routes/api/users.php');
    require base_path('routes/api/Stock/configMode.php');
});
```

```php
Route::group(['middleware' => ['jwt.verify']], function () {
    Route::prefix('roles')->controller(RoleController::class)->group(function () {
        Route::get('/all', 'all');
        Route::post('/store', 'store');
        Route::put('/update/{id}', 'update');
        Route::delete('/destroy/{id}', 'destroy');
        Route::get('/show/{id}', 'show');
        Route::post('/{id}/permissions/update', 'updatePermissions');
        Route::post('/{id}/permissions/sync', 'syncPermissions');
        Route::get('/{id}/users', 'getUsersByRoleId');
    });
    Route::prefix('permissions')->controller(PermissionController::class)->group(function () {
        Route::get('/all', 'all');
        Route::get('/all/{id}', 'getPermissionsByActionAndModuleForRole');
    });
});
```

Respuestas JSON

Los front no deben sufrir por recibir diferentes estructuras en respuestas.

Errores
400, 404, 500, etc.

Al finalizar:
200, 201, etc.

```php
/**
 * Manejar un intento de validación fallido.
 *
 * @param \Illuminate\Contracts\Validation\Validator $validator
 * @return \Illuminate\Contracts\Validation\Validator
 */
0 references | 0 overrides
protected function failedValidation(Validator $validator)
{
    throw new HttpResponseException(response()->json([
        'code' => 400,
        'message' => 'Comprobar información',
        'errors' => $validator->errors(),
    ], 400));
}
```

```php
catch(Exception $e) {
            Log::error($e->getMessage() . ' - ' . $e->getLine() . ' - ' . $e->getFile());
            return Response::json([
                'code' => $e->getCode(),
                'status' => 'error',
                'message' => 'Error en la generación de la solicitud.',
                'errors' => ['error' => $e->getMessage()]
            ], $e->getCode() == 0 ? 500 : $e->getCode(), [], JSON_PRETTY_PRINT);
        }
```

Relación controladores y servicios

- Los controladores deben contener lógica puntual y deben ser los únicos en contener los try catch, de ahí parte a llamar los servicios que son el detalle.
- Los servicios deben contener exponer excepciones validadas.

```
throw new Exception($errorMessage, 422);
```

Seguridad de middleware

- Dentro de los constructores de controladores, se debe definir los acceso a los métodos con permisos

```php
class RoleController extends Controller
{
    0 references | 0 overrides
    public function __construct()
    {
        $this->middleware('role_or_permission:root|Access-Roles.browse')->only('all');
        $this->middleware('role_or_permission:root|Access-Roles.read')->only(['show', 'getUsersByRoleId']);
        $this->middleware('role_or_permission:root|Access-Roles.add')->only('store');
        $this->middleware('role_or_permission:root|Access-Roles.edit')->only(['update', 'updatePermissions', 'syncPermissions']);
        $this->middleware('role_or_permission:root|Access-Roles.delete')->only('destroy');
    }
```

Clases extensas

- Es importante evitar centralizar todo en un solo archivo y que se vuelva extenso.
- No llegaremos a un patrón de diseño atómico, pero si evitar las grandes extensiones.

Variables

- Para variables y métodos usar el camel case dentro de laravel.
- Datos dentro de la base de datos snake case.