How do I create a custom module?

When interacting with the admin, especially when making a custom module, it is important to require `aerocommerce/admin` in the modules' `composer.json` file.

Scaffolding modules

Modules can be created using artisan, a command line interface supplied with Laravel. The easiest way to create a new module is to open the terminal, navigate to the root folder of the project and paste in:

```
php artisan make:module vendor/module-name
```

It is important to stick to module **naming conventions** and include the **vendor** (such as aerocommerce or aerocargo) and the **module name**, with dashes replacing any space in the name.

Running the above command in the terminal results in a couple of brief messages, reporting on the status of the creation and installation process of our module. After all is complete, the module can be accessed in the root directory of Aero under "modules".

Configuring modules

As mentioned above, each module that interacts with the admin needs to require `aerocommerce/admin` in the modules' `composer.json` file.

Composer.json

```
{
    "name": "aerocargo/new-module",
    "description": "",
    "require": {
        "php": "^7.2|^8.0",
        "aerocommerce/core": "^0",
        "aerocommerce/admin": "^0"
```

```
    },
    "autoload": {
        "psr-4": {
            "Aerocargo\\NewModule\\": "src/"
        }
    },
    "extra": {
        "laravel": {
            "providers": [
                "Aerocargo\\NewModule\\ServiceProvider"
            ]
        }
    }
}
```

Service Provider

The module **Service Provider**ensures that our modules are connected to the rest of the platform through the **setup()**method that is automatically scaffolded into the class.

Adding modules to a visible list in the admin

Some modules might not require any interaction with the admin in terms of UI, so they don't need to necessarily be listed in the modules section of the admin. If we wish to give our module an interface and allow users to access the module through the admin interface, we have to add the following code to our **Service Provider's setup()**function:

```
AdminModule::create('new-module')
            ->title('New Aero Module')
            ->summary('A brand new Aero module.')
            ->routes(__DIR__.'/../routes/admin.php')
            ->route('admin.new-module.index');
```

This module should now be listed in the admin.

Loading migrations

In order for the module to be able to detect all the migrations that a module has, it has to have a specified path in the modules' **Service Provider setup()**function.

If the migration folder follows the original anatomy of module structure, all we have to do is paste the following code into the **Service Provider setup()**function:

```
$this->loadMigrationsFrom(__DIR__.'/../database/migrations');
```

You should now be able to call `php artisan migrate` in the root Aero directory to migrate data from the module.

Loading views

Loading views works the same way as loading migrations, all we have to do it to specify the path to our views and also a namespace.

```
$this->loadViewsFrom(__DIR__.'/../resources/views', 'new-module');
```

The second parameter in the above function is the namespace assigned to all of the module views. This is extra useful as we can then use that namespace to render views in the controller:

```
return view('new-module::index');
```

Loading routes

There are two different types of routes that the module has access to. One of them is the **admin routes** which only give access to routes provided the user is an administrator. The other is the **store routes** which affect the store/frontend side of the system.

Admin routes

There are two ways of setting up **admin routes**, depending on whether we choose to use the AdminModule facade and load the module into the admin interface.

If we do choose to add our module to a list in the admin, apply the following chained function to the AdminModule facade in the **setup()**function in the **Service Provider**:

```
AdminModule::create('new-module')
                    ->title('New Aero Module')
                    ->summary('A brand new Aero module.')
                    ->routes(__DIR__.'/../routes/admin.php');
```

This allows us to create the necessary admin routes, for navigating the module in the admin.

If we don't choose to add our module to a list in the admin, we simply add the following piece of code to the **setup()**function in the module **Service Provider**:

```
Router::addAdminRoutes(__DIR__.'/../routes/admin.php');
```

Store routes

In order to add store (frontend) routes to the module, we simply add a piece of code to the **setup()**function of the module **Service Provider**:

```
Router::addStoreRoutes(__DIR__.'/../routes/store.php');
```

Asset Linking

In order to publish any resources from our module to the rest of Aero, we have to specify the path for those resources in a special function. This function is added to the module **Service Provider**:

```
public function assetLinks()
{
        return [
                'aerocargo/new-module' => __DIR__.'/../public',
        ];
}
```

After defining any asset paths to link, you'll need to run the command:

```
php artisan aero:link
```

Revision #1
Created 2026-09-02 13:28:32 UTC by Michael Price
Updated 2026-09-02 13:28:32 UTC by Michael Price