Modules

General

General

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/acme",
    "description": "",
    "require": {
        "php": "^7.2|^8.0",
        "aerocommerce/core": "^0",
        "aerocommerce/admin": "^0"
    },
```

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

```
\Aero\Admin\AdminModule::create('acme')
    ->title('Acme')
    ->summary('A brand new Aero module.')
    ->routes(__DIR__.'/../routes/admin.php')
    ->route('admin.acme');
```

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', 'acme');
```

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('acme::index');
```

Loading routes

There are two different types of routes that the module has access to:

- Admin routes, which only give access to routes provided the user is an administrator.
- Store routes, which affect the store/frontend side of the system.

Admin Routes:

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

> See "Adding modules to a visible list in the admin" to see how to add routes via the `AdminModule` instead.

Store Routes:

```
\Illuminate\Routing\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(): array
{
    return [
        'aerocargo/acme' => __DIR__.'/../public',
    ];
```

```
}
```

> After defining asset links, you will need to run `php artisan aero:link`.

Vue

How do I register a custom Vue component for my module?

To register a Vue component for our module, we will need to create a JavaScript file for our components and in order to initialise Vue.

Link assets

In order to link all the assets, which is a necessary step of registering a Vue component, we need to add a function to the module Service Provider. This is the code we need in the service provider:

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

> Replace `aerocargo/acme` with your module vendor & name.

> After defining asset links, you will need to run `php artisan aero:link`.

Initialise components

```
// resources/js/components.js

import NewComponent from './components/NewComponent;

window.Acme = {
    install(Vue) {
```

```
        Vue.component('new-component', NewComponent);
    },
}
```

> The above code gives us access to the `<new-component></new-component>` tag, provided we've loaded the components into a view.

Loading components into a view

```
@push('scripts')
    <script src="{{ asset(mix(components.js', 'modules/aerocargo/acme')) }}"></script>
    <script>
        window.AeroAdmin.vue.use(window.Acme);
    </script>
@endpush
```

Enable Vue dev tools

In order to enable Vue devtools for our module, we simply add the following line of code into the file where we initialize our components:

```
import NewComponent from './components/NewComponent;

window.Acme = {
    install(Vue) {
        Vue.config.devtools = true;

        Vue.component('new-component', NewComponent);
    },
}
```