Account Area

Account Area

What are the default pages in the account area?

The following classes are in the **Aero\AccountArea\Http\Responses**namespace.

Class	Route	Description
AccountLoginPage	login	A page where users can log in.
AccountRegisterPage	register	A page where guests can register.
AccountPasswordForgotPage	password.forgot	A page where users can send a password reset email to their email.
AccountPasswordResetPage	password.reset	A page where users can reset their password.
AccountOrdersPage	account.orders	A page that lists all of a user's orders and lets them filter/search.
AccountOrderViewPage	account.view-order	A page that shows more information about an order.
AccountAddressesPage	account.addresses	A page that lists all of the users addresses.
AccountAddressNewPage	account.new-address	A page where users can create a new address,
AccountAddressEditPage	account.edit-address	A page where users can edit one of their addresses.
AccountDetailsPage	account.details	A page where users can update their account details and change their password.
AccountInvoicePage	account.view-invoice	A simple invoice for an order - currently not linked to from anywhere.
AccountOverviewPage	account	The first page a user sees after they log in - currently redirects to the orders page.

What are the default sets in the account area?

The following classes are in the **Aero\AccountArea\Http\Responses**namespace.

Class	Method	Route
AccountLoginSet	Post	login
AccountLogoutSet	Post	logout
AccountRegisterSet	Post	register
AccountPasswordForgotSet	Post	password.forgot
AccountPasswordResetSet	Post	password.reset
AccountAddressNewSet	Post	account.new-address
AccountAddressEditSet	Put	account.edit-address
AccountAddressMakeDefaultSet	Put	account.make-default-address
AccountDetailsSet	Put	account.details
AccountAddressDelete	Delete	account.delete-address

What are the default forms in the account area?

The following classes are in the **Aero\AccountArea\Http\Forms**namespace.

Class	Description
AccountAddressEditForm	The edit address form.
AccountAddressNewForm	The new address form.
AccountDetailsForm	The account details form.
AccountLoginForm	The login form.
AccountPasswordForgotForm	The forgotten password form.
AccountPasswordResetForm	The password reset form.
AccountRegisterForm	The register form.

How do I inject into a slot in the account area?

You can inject into account area slots using the **Aero\AccountArea\AccountAreaSlot::inject** method. This method accepts two parameters, the key of the slot you want to inject into and then the view or a closure that defines the view to be injected. When passing a closure it enables you to add additional view data to use inside of your injected view.

```
<?php

namespace Acme\MyModule;

use Aero\AccountArea\AccountAreaSlot;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       $this->loadViewsFrom(__DIR__.'/../resources/views', 'my-module');

       AccountAreaSlot::inject('orders.order.card.header', 'my-module::account-area-slot');

       AccountAreaSlot::inject('orders.order.card.header', function ($data) {
           $data['custom_data'] = 'custom value';

           return view('my-module::account-area-slot', $data);
       });
   }
}
```

How do I configure the account area?

ENV Variables

Variable	Default	Description
AERO_ACCOUNT_AREA_THEME	aerocommerce/account-area	Defines the namespace that should be registered as having the views and resources for the account area. This should only be changed under unique circumstances.
AERO_ACCOUNT_AREA_LAYOUT	layouts.main	Defines the layout that the account area views use. If the layout provided doesn't exist, account-area::layouts.blank will be used.

Config File

If you would like to change any of the config values you can publish the config file by running:

```
php artisan vendor:publish --provider="Aero\AccountArea\ServiceProvider"
```

Once the file is published you will be able to edit it through the config php file created at **/config/aero/account-area.php**.

Date Format

This defines the format used when displaying any dates in the account area.

Order Status State Classes

This defines the map used to find the colour for the order status state.

The colours are defined using CSS classes:

Class	Colour
success	Green
warning	Orange
error	Red

Order Status Radio Options

This defines the order statuses shown in the radio options on the orders page and what order status states they map to.

Order Years

This defines the number of previous years to display in the years dropdown on the orders page.

Page Sections

This defines the sections that make up each page by default. You can add/edit/remove sections through code, it is not necessary to adjust these values usually.

Slots

This defines the views that are injected into slots by default.

Links

This defines the links that should be in the navbar by default. You can add/edit/remove links through code, it is not necessary to adjust these values usually.

How do I create a custom page in the account area?

To create and register a custom account area page you need to use the **Aero\AccountArea\AccountArea::registerPage()**helper method. This method expects you to pass a class that extends **Aero\AccountArea\AccountAreaPage**.

```
<?php

namespace Acme\MyModule;

use Acme\MyModule\AccountArea\Pages\CustomPage;
use Aero\AccountArea\AccountArea;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AccountArea::registerPage(CustomPage::class);
   }
}
```

Your account area page class must implement the **title**, **route**, and **routeName**methods that all should return a string. It's common practice to use Laravel Localization for the title and route so that they can change depending on the language used. You should not use this for the routeName as the routeName is used in code to refer to the route and therefore should never change.

You should also define a protected static steps and middleware array and a protected sections array. The static steps array can be used to define any extra response steps required for your page. The middleware array can be used to define any middleware required for your page (the most useful being 'account' which means users need to be logged in to view your page). The sections array defines the views that make up your page.

```php
<?php

namespace Acme\MyModule\AccountArea\Pages;

use Aero\AccountArea\AccountAreaPage;

class CustomPage extends AccountAreaPage
{
   protected static $steps = [
       Steps\AttachWishlist::class,
   ];

   protected static $middleware = [
       'account',
   ];

   protected $sections = [
       'navbar' => 'account-area::sections.navbar',
       'page-header' => 'account-area::partials.page-header',
       'wishlist' => 'wishlist::account-area-wishlist',
   ];

   static function title(): string
   {
       return __('My Custom Page');
   }

   static function route(): string
   {
       return __('custom-page');
   }

   static function routeName(): string
   {
       return 'acme.my-module.custom-page';
   }
}
```

Adding your Page to the Navbar

To add your page to the navbar you need to make sure your page class uses the **Aero\AccountArea\Traits\HasAccountAreaLink**trait and then add the link to the navbar using the **addLink**, **addLinkBefore**, or **addLinkAfter**methods from within the pipeline.

The **Aero\AccountArea\Traits\HasAccountAreaLink**trait makes you implement a **toLink**method that returns an array of data used to render your link. The array should have an icon and text key.

```
public static function toLink(): array
{
   return [
       'icon' => 'my-module::icon',
       'text' => 'My Page',
   ];
}
```

```
<?php

namespace Acme\MyModule;

use Acme\MyModule\AccountArea\Pages\CustomPage;
use Aero\AccountArea\AccountAreaPage;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AccountAreaPage::extend(function ($page) {
           $page->addLink(CustomPage::class);
       });
   }
}
```

How do I extend a form in the account area?

Forms can be extended using a pipeline from your modules service providers setup method or your projects app service providers boot method. You can read more about pipelines in the article "

Introduction to pipelines

Adding a Field

To add fields to a form you can use the **addSection**, **addSectionBefore**, or **addSectionAfter** methods from within the pipeline.

addSection Method

This method accepts a key and a view to be injected. A field added with this method will be added as the first field.

addSectionBefore Method

This method accepts the same parameters as the **addSection**method but additionally accepts a third parameter, the key of the field this new field should be added before.

addSectionAfter Method

This method accepts the same parameters as the **addSection**method but additionally accepts a third parameter, the key of the field this new field should be added after.

Finding the Current Field Keys

If you do not know the keys of the current fields you can use this code snippet to dump the current keys when visiting the page with the form.

```php
<?php

namespace Acme\MyModule;

use Aero\AccountArea\Http\Forms\AccountAddressForm;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AccountAddressForm::extend(function ($form) {
           dd($form->getSections()->keys()->toArray());
       });
   }
}
```

```php
<?php

namespace Acme\MyModule;

use Aero\AccountArea\Http\Forms\AccountAddressForm;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AccountAddressForm::extend(function ($form) {
           $form->addSection('my-section', 'my-view');
       });
   }
}
```

Removing a Field

You can use the **removeSection**method within the pipeline to remove a field from a form. The methods expect one parameter, the key of the field that you would like to remove.

```
<?php

namespace Acme\MyModule;

use Aero\AccountArea\Http\Forms\AccountAddressForm;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AccountAddressForm::extend(function ($form) {
           $form->removeSection('line2');
       });
   }
}
```

How do I create a custom set in the account area?

Sets are used for POST/PUT/DELETE requests. To create and register a custom account area set you need to use the **Aero\AccountArea\AccountArea::registerPost()**, **Aero\AccountArea\AccountArea::registerPut()**, or **Aero\AccountArea\AccountArea::registerDelete()**, helper method. This method expects you to pass a class that extends **Aero\AccountArea\AccountAreaSet**.

```
<?php


namespace Acme\MyModule;


use Acme\MyModule\AccountArea\Pages\CustomSet;
use Aero\AccountArea\AccountArea;
use Aero\Common\Providers\ModuleServiceProvider;


class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AccountArea::registerPost(CustomSet::class);
   }
}
```

Your account area set class must implement the **route**and **routeName**methods that all should return a string. You should also define a protected static steps and middleware array. The steps array can be used to define any extra response steps required for your set. The middleware array can be used to define any middleware required for your set (the most useful being ‘account’ which means the user needs to be logged in).

```
<?php


namespace Acme\MyModule\AccountArea\Pages;
```

```
use Aero\AccountArea\AccountAreaSet;

class CustomSet extends AccountAreaSet
{
   protected static $steps = [
       Steps\DeleteWishlist::class,
   ];

   protected static $middleware = [
       'account',
   ];

   static function route(): string
   {
       return 'wishlist/delete';
   }

   static function routeName(): string
   {
       return 'acme.my-module.wishlist.delete';
   }
}
```

How do I extend a page or set in the account area?

Account Area Pages/Sets are Response Builders (link) which means that they can be extended using a pipeline. You can use all of the usual Response Builder stuff and also for pages you can add/remove/update sections using **addSection**, **addSectionAfter**, **addSectionBefore**, and **removeSection**.

addSection Method

This method accepts a key string and a view string. A section added with this method will be added as the first section.

addSectionBefore Method

This method accepts the same parameters as the **addSection**method but additionally accepts a third parameter, the key of the section this new section should be added before.

addSectionAfter Method

This method accepts the same parameters as the **addSection**method but additionally accepts a third parameter, the key of the section this new section should be added after.

removeSection Method

This method accepts one parameter, the key of the section to remove.

Finding the Current Section Keys

If you do not know the keys of the current sections you can use this code snippet to dump the current keys out when visiting the page.

```
<?php

namespace Acme\MyModule;

use Aero\AccountArea\Http\Responses\AccountOrdersPage;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AccountOrdersPage::extend(function (AccountOrdersPage $page) {
           dd($page->getSections()->keys());
       });
   }
}
```

```
<?php

namespace Acme\MyModule;

use Aero\AccountArea\Http\Responses\AccountOrdersPage;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       $this->loadViewsFrom(__DIR__.'/../resources/views', 'my-module');

       AccountOrdersPage::extend(function (AccountOrdersPage $page) {
           $page->addSection('my-section', 'my-module::view');

           $page->addSectionAfter('my-other-section', 'my-module::view', 'navbar');
       });
   }
```

```
}
```

How do I add a custom CSS file to the account area?

You can add a custom CSS file to the account area by creating a file named `account-area.css` in your public directory.

Example contents where we're overriding the default colours:

```
:root {
    --bpa-color-primary: deepskyblue;
    --bpa-color-success: forestgreen;
    --bpa-color-warning: darkorange;
    --bpa-color-error: red;
    --bpa-color-helpbox: aliceblue;
    --bpa-color-content-background: ghostwhite;
}
```

How do I create a new form for the account area?

To create a form you need to create a class that extends the `AccountAreaForm` class.

The `$sections` array lists the views used to render the form.

The `method` method defines the method that the form will use (this should be `post`, `get`, `put`, or `delete`).

The `route` method should return the URL for the form. A `$data` variable is accessible in case the route requires parameters that are available in the data.

```
<?php

namespace Acme\MyModule;

use Aero\AccountArea\AccountAreaForm;

class MyForm extends AccountAreaForm
{
    protected $sections = [
        'email' => 'account-area::partials.email-input',
        'password' => 'account-area::partials.password-input',
        'forgot-password' => 'account-area::partials.login-forgot-password-link',
        'submit' => 'account-area::partials.login-button',
    ];

    protected function method(): string
    {
        return 'post';
    }

    protected function route($data): string
    {
        return route('login');
```

```
    }
}
```

Registering a form

Any form you wish to use in the account area should be registered in your service provider.

```
<?php

namespace Acme\MyModule;

use Aero\AccountArea\AccountArea;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup(): void
    {
        AccountArea::registerForm(MyForm::class);
    }
}
```

What are the available slots in the account area?

The following slots can be injected into.

- `orders.order.card.header` - `orders.order.card.dispatched` - `orders.order.card.not-dispatched` - `orders.order.card.footer` - `addresses.address.card.footer`