Station Documentation

Release 0.2

Ben Hirsch

Jul 13, 2018

Contents

Repository on Github

Station helps Laravel developers to auto-generate, configure, and deploy an admin interface with advanced scaffolding, migrations and models for their Laravel software. It includes an artisan build command and a simple but extensible UI.

If you are tired of creating CRUD-capable interfaces from scratch, Station may be perfect for you. In a few minutes you can have a simple, but advanced back-end admin installed and ready for use by you, your customers, and your customers' users.

Station shines when used as a CMS however it is *not* your typical template-based CMS system. It does not include front-end templates or layouts. Instead it is intended to be used as a database, content, and user management system. It takes away the heavy-lifting involved in creating a back-end for your web site or application. But, it leaves the front-end a blank canvas for your creativity!

CHAPTER 1

Features

- A password-protected section of your domain under `/station/...` or the virtual directory name of your choice.
- Authenticated user system utilizes Laravel's native user model.
- A simple set of configuration text files defines your entire database, permissions and navigation schema.
- A build command at `php artisan station:build` will generate + run migrations and create models.
- Define multiple user `groups` where users can only access areas configured for their own group.
- A bootstrap-based UI containing necessary behaviors such as drag and drop reorderables, image upload w/ crop tools, nested sortables, and more.
- User management functions such as password reminders and password reset.

CHAPTER 2

Requirements

- All of the basic requirements of the Laravel library
- GD (compiled with PHP, if you want to take advantage of the image resizing features)

CHAPTER 3

Documentation Contents

3.1 Installation

1. Environment

Make sure Laravel is already installed and running. You should already have a valid database connection. This install process works best on a fresh Laravel installation.

2. Install Using Composer

Station should be installed via Composer by requiring the `lifeboy/station` package in your project's `composer.json`.

```
{
    "require": {
        "lifeboy/station": "dev-master"
    }
}
```

Then run a composer update

```
composer update
```

This assumes you have a working dev or production environment with Laravel 5 and a database already installed and configured.

3. Register Station Within Laravel

To use Station, you must register the provider when bootstrapping your Laravel application.

Find the `providers` key in `app/config/app.php` and register the Station Service Provider.

```
'providers' => array(
    // ... add below ...
    Lifeboy\Station\StationServiceProvider::class,
```

(continues on next page)

(continued from previous page)

```
    Collective\Html\HtmlServiceProvider::class,
),
```

Also update the `aliases` key in `app/config/app.php`

```
'aliases' => [
    // ...
    'Form' => Collective\Html\FormFacade::class,
    'Html' => Collective\Html\HtmlFacade::class,
],
```

In `app/Http/Kernel.php`, update the `$routeMiddleware` class variable:

```
protected $routeMiddleware = [
    // ...
    'station.session' => \Lifeboy\Station\Filters\Session::class
];
```

4. Publish Station's Assets to Laravel

```
php artisan vendor:publish
```

Note: We sometimes see a "path not found" warning on this step. You can safely ignore this.

5. Run Default Migrations

```
php artisan migrate
```

6. Set the Administrator Email

In `config/packages/lifeboy/station/_app.php` set the `root_admin_email` to your email address.

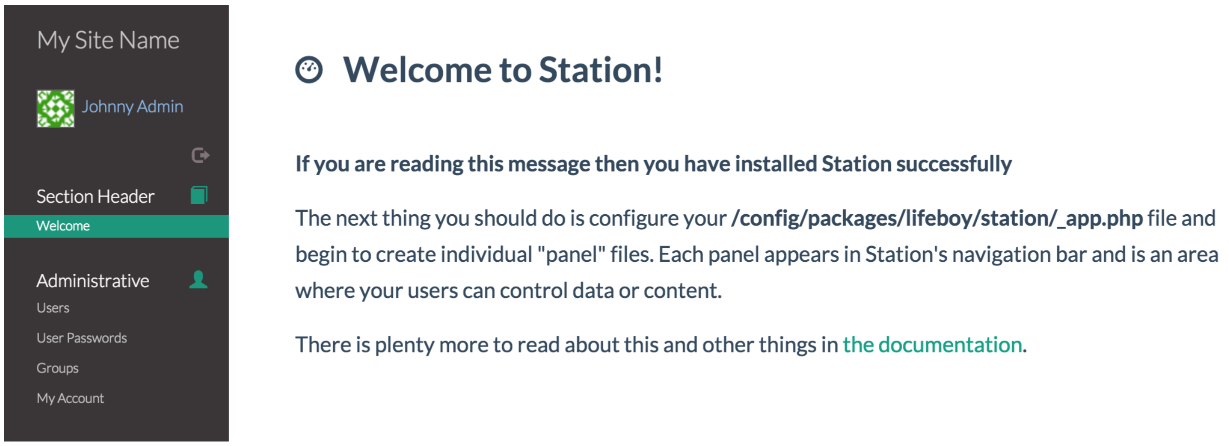
7. Run Station's Build Command

This will generate new migrations, run the new migrations, generate models, and seed the database.

```
php artisan station:build
```

8. Test Installation

You should now be able to browse to your app at: `http://{host}/station/`. You can login using user/password: `admin/admin`. Upon login, you will see a welcome screen:



Welcome to Station!

If you are reading this message then you have installed Station successfully

The next thing you should do is configure your **/config/packages/lifeboy/station/_app.php** file and begin to create individual "panel" files. Each panel appears in Station's navigation bar and is an area where your users can control data or content.

There is plenty more to read about this and other things in the documentation.

9. Configure Station and Your Panels!

That's it! Next you will configure Station and tailor it to the needs of your users.

Note: You will be able to configure and test Station now. However, to run Station in production, please make sure that your email system has been configured and enabled. (See *Configuring Emailers* for more info).

3.2 Basic Principles & Configuration

Station has three levels of configuration: **Panel Level**, **Element Level** and **Application Level**.

3.2.1 Panel Level Configuration

Each individual section that appears in the navigation bar is referred to as a "panel". By default, a panel contains all of the functionality to create, remove, update and delete records from a database table or foreign table. Panels can be customized in numerous ways. Panels can also be overriden entirely so that you may develop your own functionality and circumvent default panel behaviors.

To learn all of the specifics of configuring panels, refer to *Working With Panels*.

3.2.2 Element Level Configuration

Each panel can have any number of elements associated with it. Typically each element refers to a specific element in your database but that is not always the case. A panel can have "virtual" elements as well as no elements at all.

To learn all of the specifics of configuring panel elements, refer to *Working With Panels*.

3.2.3 Application Level Configuration

All of Station's top level configuration occurs in the `config/packages/lifeboy/station/_app.php` file. The boilerplate version of this file was copied to your app during the installation process.

Following is a list of all of the available configuration options:

name

This is the name of your site or application

root_admin_email

This value will be seeded into the `users` table when the admin user is created during the installation process.

root_uri_segment

This value is what Station will use as its pseudo-directory "slug". By default this is set to `station` which means that all of the URLs for Station will be found at `http://my-site.com/station/...`. You could use `admin` which would result in all URLs looking like `http://my-site.com/admin/...`

panel_for_user_create

This is the name of the panel which is used for creating new users. You may never need to use this as your app will likely have it's own onboarding process. However this panel is shipped with Station in case you want a simple way for new users to create an account.

user_group_upon_register

This is the group which a new user will be automatically set to upon creation.

css_override_file

Optionally set this to the URL or path of a .css file which will be loaded on every Station URL. This is excellent for adding branding specifics to your Station installation.

media_options

AWS: Station currently only supports file uploads to AWS/S3. Enter your bucket, key and secret key here to establish connectivity. If your app does not require uploads then you do not need to configure this option.

Sizes: This is where you set the default image sizes for all uploaded images. If any of your panel elements utilize image uploads and *do not* specify a set of image sizes, these sizes will be used. This is a handy way to set standard sizes for all of your images site-wide, if needed. For more information how to configure this option, refer to the *sizes* configuration documentation.

user_groups

This associative array defines the user groups for your app but also, just as importantly, defines the **entire navigation structure** of Station. Please refer to the boilerplate sample of `config/packages/lifeboy/station/_app.php`, included in the installation, to visualize the structure of this document.

user_groups.[group name]: This associative array's key is the group name. This is only for internal use. You can use any name you want. When you run `php artisan station:build` your groups will be seeded to the `groups` table in your database and presented as options in your `users` panel.

user_groups.[group name].starting_panel: This is the panel key that a user belonging to this group will be redirected to upon log in. The syntax for this is `panel_name.ACTION`, where action is L (list view), C (create view), or U (update view).

user_groups.[group name].panels This nested associative array defines the navigation that a user from this group will see in their navigation bar. It also defines the sections, panel titles, and permissions that a user of this group has regarding these panels. The format is as follows:

```php
<?php

'panels' => [

        'demo_section'    => ['name' => 'Section Header',   'is_header' =>␣
↪TRUE, 'icon' => 'glyphicon glyphicon-book'],
        'posts'           => ['name' => 'Posts',            'permissions' =>
↪'CRUDL'],

        // ... more sections headers and panels go here ...
]
```

In the above example, **demo_section** is the key name for a section header. The actual name is irrelevant. Just make sure all of your section header keys have unique names because this is PHP array and you cannot duplicate your key names! **is_header** indicates that this item is only a header title and not an actual panel. The **icon** option allows you to use bootstrap glyphicon names to accompany your section headers.

The **posts** key references an actual panel, not a section header. This key must match the name of a file in the `config/packages/lifeboy/station` directory where the *Working With Panels* is defined. The **name** option is the actual title of the panel as it will appear in the naviagtion.

The **permissions** option sets the permissions that a user from this group has on this panel. You can enter any combination of the letters C.R.U.D. and L:

```
C = Create
R = Read
U = Update
D = Delete
L = List
```

For example, if you only specify the letter `L` for permissions then the user will only be able to list the records in this panel. Specifying all of the letters gives the user full permissions on this panel.

html_append_file

This option allows you to specify an HTML or PHP blade file to append to every Station view. This is ideal for analytics.

html_prepend_content_file

Like `html_append_file` you can specify an HTML or PHP blade file to prepend to the content area of every panel in Station. This is ideal for onboarding progress timelines or system-wide, universal alerts.

strict_domains

This forces all requests within Station to return a 404 unless one of the domains specified in this array is the domain indicated in the request.

3.2.4 Configuration Variables

The %user_id% variable can be used in any value of the application or panel config files. The user's ID will be replaced. This allows you to create panels which display only user-specific data. See *Working With Panels* for more examples of where and how this can be used. See below on how this configuration variable can be used in the application level configuration:

3.2.5 Custom Configuration Variables

You can create your own custom configuration variables custom_user_vars which are accessible in any panel configuration file and the application configuration file. You can also create custom_view_vars which are available in any Station views. Just add them to the top-level of your config/packages/lifeboy/station/_app.php file.

```
<?php

'custom_user_vars' => [

        'user_company_ids' => '\CompanyRepository::id_list_for_user(%user_id%)',
        'user_store_ids' => '\StoreRepository::id_list_for_user(%user_id%)',
],

'custom_view_vars' => [

        'onboarding_progress_html' => '\UserRepository::onboarding_progress_html_for(
↪%user_id%)',
],
```

In this example we are utilizing a CompanyRepository class, which is part of our Laravel app. This class is returning a set of IDs based on the current user's ID. Those IDs are now stored in %user_company_ids%, which can be used in any panel configuration file.

Similarly, with custom_view_vars we are creating the variable $onboarding_progress_html which is now accessible in any Station view. In this example we're generating a snippet of HTML which is being inserted into the file that we specified as our html_prepend_content_file. That snippet of HTML contains information about onboarding specific to the user who is logged in.

You can create as many of these custom variables as you wish.

3.3 Working With Panels

Panels are the heart and soul of Station. Out of the box, panels contains all of the functionality necessary for a user to manipulate a specific type of data in your database. There are numerous options for panels and elements giving you the flexibility to craft back-end tools to fit virtually any scenario or database structure.

Typically each panel maps to a database table one-to-one. However, a panel can also contain one or more **subpanels** as well as references to other tables for lookup capabilities.

A panel can also be complete overriden so you can build your own functionality from scratch, but within the user-authenticated comfort of Station.

3.3.1 The Build Command

Each time you add a new panel or change a panel's configuration you need to run `php artisan station:build`.

This artisan command will analyze your panels and elements and then do the following:

- generate migrations for any missing database fields, including database pivot tables
- run the new migrations
- generate / refresh models with foreign relationships for each panel (see "*Working with Station Models*" for more info)
- seed the database (only when new user groups have been added)

If you attempt to navigate to a newly created panel in the Station navigation, you will likely encounter errors. You must run the build command first.

3.3.2 Accessing Panel Configuration

In your Laravel app you may wish to access a panel's configuration array. To accomplish this, just add `use Lifeboy\Station\Config\StationConfig as StationConfig` to the top of any class. Then,

```
$my_panel_config = StationConfig::panel('my_panel_name');
```

3.3.3 Panel Creation Workflow

As discussed in "*Application Level Configuration*", when you add new array elements to the `user_groups` associative array in your `config/packages/lifeboy/station/_app.php` file, you are in-effect registering new panels with Station.

Station is able to determine the behaviors of each panel given the values of the configuration parameters found in each panel's configuration file. For example, if you have a panel named `posts` in your `config/packages/lifeboy/station/_app.php` file, then you need a corresponding `posts.php` file in the `config/packages/lifeboy/station/` directory to define the parameters of that panel.

For example, refer to the `config/packages/lifeboy/station/users.php` file which was packaged with your installed copy of Station, you will notice that there are two top-level array keys, `panel_options` and `elements`:

```
'panel_options' => [

        // define your panel-level options here...
],

'elements'      => [

        // define your element-level options here...
],
```

Panel options control the overall configuration of the panel (think of this as database table-level options). Elements define how your users interact with specific database fields.

Without further ado, what follows is an exhaustive list of each and every panel and element option possible along with specific examples for each. Dive in!

3.4 Panel Options

The `panel_options` key of the array defined in any of the files at `/config/packages/lifeboy/station` represents the panel-level options which are available to users of Station.

Generally, each panel is mapped to a specific database table. However, this is not always the case. Some panels have an *override* defined. See below for the full documentation on configuring panels.

Note: All options marked with a * are required

3.4.1 table *

Quite simply, this is the name of the database table to which this panel corresponds. Note that you can have many panels which use the same table.

```
'panel_options'   => [

   'table' => 'posts',
   ...
],
```

3.4.2 single_item_name *

This is the singular version of the type of data which is being dealt with in this panel.

```
'panel_options'   => [

   'table'              => 'posts',
   'single_item_name'   => 'Post',
   ...
],
```

3.4.3 allow_bulk_delete

Setting this option to true will allow users to select and delete multiple items in list views (including filtered list views)

3.4.4 default_order_by

This option allows you to set the order in which records for this panel will be displayed. You can choose one or more database fields. You will use traditional SQL syntax.

```
'panel_options'    => [

   'table' => 'posts',
   'default_order_by' => 'title ASC, date DESC',
   ...
],
```

3.4.5 has_timestamps

If this option is set to true then the migrations and models generated for this panel/table will create and utilize Laravel Eloquent's `created_at` and `updated_at` timestamp fields.

```
'panel_options'    => [

   'table' => 'posts',
   'has_timestamps' => TRUE,
   ...
],
```

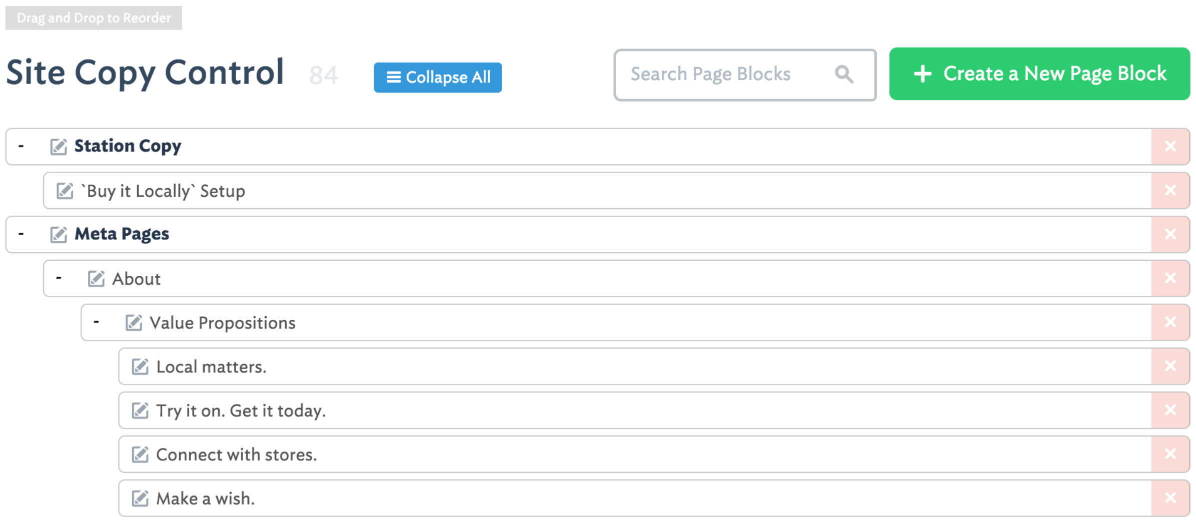
3.4.6 js_include

This option allows you to specify a javascript file to include on all pages of this panel. It is great for sewing in your own functionality. jQuery is available on all pages as well.

```
'panel_options'    => [

   'table' => 'posts',
   'js_include' => '/js/my-own.js',
   ...
],
```

3.4.7 nestable_by

This is a very powerful feature which will allow your users to reorder and hierarchically "nest" the records of this table. When enabled your users can drag and drop records to reorder them arbitrarily as well as "nest" them into a tree-like model.



```
'panel_options'    => [

   'table' => 'pages',
   'nestable_by' => ['position', 'parent_id', 'depth'],
   ...
],
```

The three array elements are (1) the field name which contains the overall sort-order (2) the field name which contains the ID of the parent of the record. Records on the top-level have a `parent_id` of `0` and (3) the depth level of the record.

Note: You do not need to also create ''position'', ''parent_id'', and ''depth'' elements in your panel's :ref:'element-options' configuration. Station will manage these for you.

3.4.8 no_build

If this option is set to true then the `php artisan station:build` command will simply skip this panel entirely when it attempts to create models and migrations. This is useful for panels where you want to use the `override` option and you have no need for a data model to be available.

3.4.9 no_data_alert

This option, defined by an array, can be used to configure a special message to users of a panel which has no data. This can be useful for when you want to assist users on creating a type of data for the first time.

```
'panel_options'    => [
   'table'              => 'posts',
   'single_item_name'   => 'Blog Post',
   'no_data_alert'      => [

      'header'     => 'You have no blog posts yet',
      'body'       => 'Go ahead and create your first blog post now!'
   ]
],
```

3.4.10 no_data_force_create

When this option is set to true it will redirect a user who is trying to access a panel's (initial) list view to the panel's create view instead.

```
'panel_options'   => [

   'table' => 'posts',
   'no_data_force_create' => TRUE,
   ...
],
```

3.4.11 override

This option allows you to completely override the functionality of a specific panel using a controller and method from your Laravel app. For an example of this, look at the `welcome` panel which shipped with Station.

```
'panel_options'   => [

   'table' => 'posts',
   'override' => ['L' => 'MyControllerName@method_name'],
   ...
],
```

The `L` above means that this will override the (initial) list view of your panel. However you can override the `U` (update) function instead and just leave the list view as-is using `'override' => ['U' => 'MyControllerName@method_name'],`. When using the update override, the record your user is attempting to modify will be passed as data to your controller method automatically.

3.4.12 preview_url

This option allows you to specify a array template for generating the url for a button which will become visible in the update view of every record in this panel.

```
'panel_options'   => [

   'table' => 'posts',
   'preview_url' => ['http://www.domain.com/post/', 'posts.id', '/preview'],
   ...
],
```

The elements of this array will concatenate to form the preview URL. When one of the array's elements is in the format `table_name.field_name` it will be replaced by the actual record's value. So the example above might produce `http://www.domain.com/post/9999/preview` and a button which looks like the one below will appear on your panel's update pages:

Edit This Post
Preview
Back To List

3.4.13 reorderable_by

This option allows you to specify a field name to use as your table's "position" field. This is a field which is used to store an arbitrary, user-defined sorting-order for the records in the table. When enabled, your users will be able to drag and drop records to reorder them within the list view of this panel. Each time a user reorders the records, all of the values for the field you specify will be re-written from 0 through X.

```
'panel_options'    => [

   'table' => 'categories',
   'reorderable_by' => 'position',
   'default_order_by' => 'position',
   ...
],
```

Note: You do not need to also create a ``position`` element in your panel's :ref:`element-options` configuration. Station will manage this for you.

3.4.14 where

This option allows you to append a SQL `where` clause onto the standard query which retrieves the data for this panel.

```
'panel_options'    => [

   'table' => 'posts',
   'where' => 'title LIKE "%robot%"',
   ...
],
```

This is also a good opportunity to pass in *Configuration Variables* or *Custom Configuration Variables* if those are relevant to your app.

```
'panel_options'    => [

   'table' => 'employees',
   'where' => 'company_id IN (%user_company_ids%)',
],
```

3.5 Element Options

The `elements` key of the array defined in any of the files at `/config/packages/lifeboy/station` represents the list of elements which are available to users of Station.

Generally, each element is mapped to a specific database field. However, this is not always the case. Some elements are "virtual". See below for the full documentation on configuring elements.

Note: *All options marked with a * are required!*

Important: You do not need to create an `id` element in your panels. Station assumes that any panel which is mapped to a *table ** has an `id` field and it will auto-generate this field for you as your table's primary key and index.

3.5.1 (key name) *

The key of each element (unless the element `type` is `virtual`) is the name of the database field of this panel's *table* *.

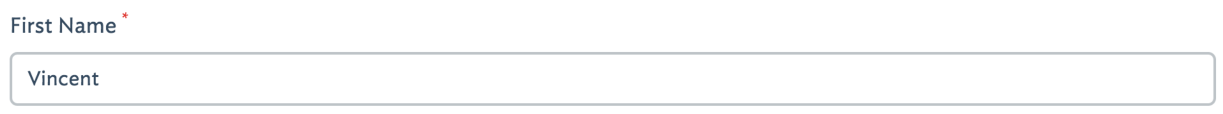
```
<?php

'first_name'   => [

   'label' => 'First Name',
   'type' => 'text',
   ...
],
```

In this example, `first_name` is the name of the database field.

3.5.2 label *

This is the user-facing "name" of the field. This name will appear in a number of places. (1) As the input label in the create and update form. (2) on the top of a list view column (if this element has been given list permissions) and (3) in validation error messages, when a user has not fulfilled validation requirements for this element.

```
<?php

'first_name'   => [

   'label' => 'First Name',
   'type' => 'text',
   ...
],
```



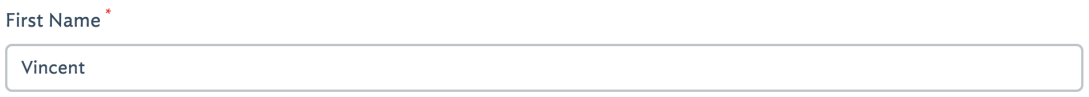
3.5.3 type *

This is the "type" of element and there are numerous options. Your choice of element type is based primarily on which kind of browser input you wish to use, however it also influences the field type used in the auto-generated database migrations:

```
<?php

'is_active'   => [

   'label' => 'Activated?',
   'type' => 'boolean',
   ...
],
```

text

- This will generate a VARCHAR(255) database field.
- The input for user manipulation is a simple `<input type="text">`

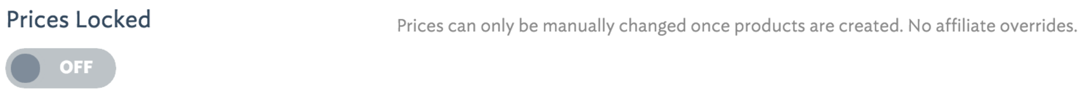


integer

- This will generate a INT(12) database field.
- The input for user manipulation is a simple `<input type="text">`

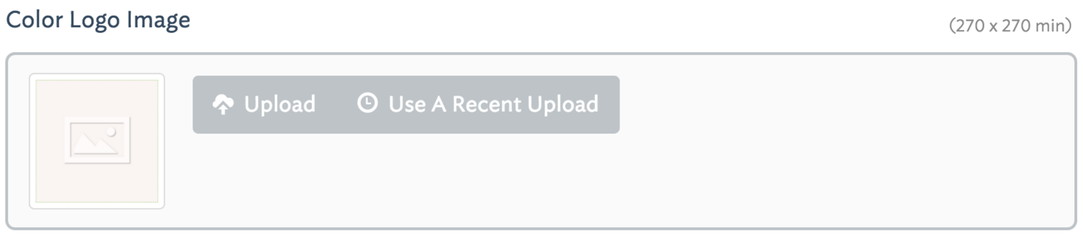
boolean

- This will generate a TINYINT(1) database field.
- The input for user manipulation is a toggle "on/off" switch (which masks a `<input type="checkbox">`).


Prices Locked
OFF
Prices can only be manually changed once products are created. No affiliate overrides.

image

- This will generate a VARCHAR(255) database field.
- The input for user manipulation is a special image uploader w/ crop tool mechanism.
- See the *sizes* option for more details on configuration, sizing and cropping.



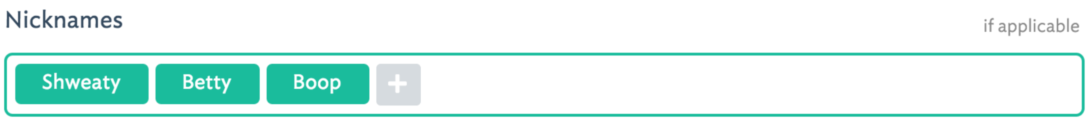
file

- This will generate a VARCHAR(255) database field.
- The input for user manipulation is a special file uploader modal dialog.
- You may also supply a `allowed_types` array containing the file-extensions this will allow.
- You may also supply a `directory` string containing the remote directory that uploaded files will save to.

- The uploaded file name will be saved to the database field.

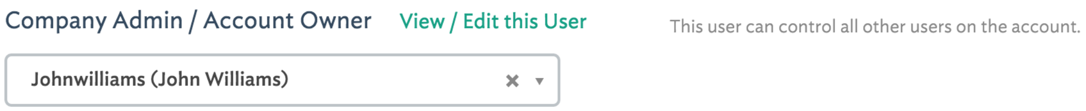
tags

- This will generate a VARCHAR(255) database field.
- The input for user manipulation is a special tagging interface.
- The field data is written to the database as comma delimited values.



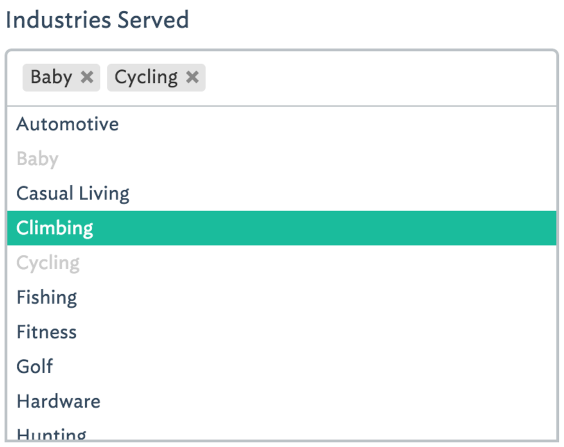
select

- This can be used in conjunction with another table or with static data (see the *data* option).
- The input for user-manipulation uses the wonderful Chosen library which contains a dropdown with search bar


Company Admin / Account Owner
View / Edit this User
This user can control all other users on the account.
Johnwilliams (John Williams)

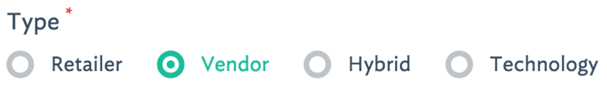
multiselect

- This can only be used when a relationship with another table has been defined (see the *data* option).
- Data will be written to the database via a pivot table which is auto-generated via *The Build Command*.
- The input for user-manipulation uses the wonderful Chosen library which contains a taggable drop-down with search bar



radio

- This will generate a VARCHAR(255) database field.
- This can be used in conjunction with another table or with static data (see the *data* option).
- The input for user-manipulation uses enhanced radio buttons (masking standard `<input type="radio">` inputs).



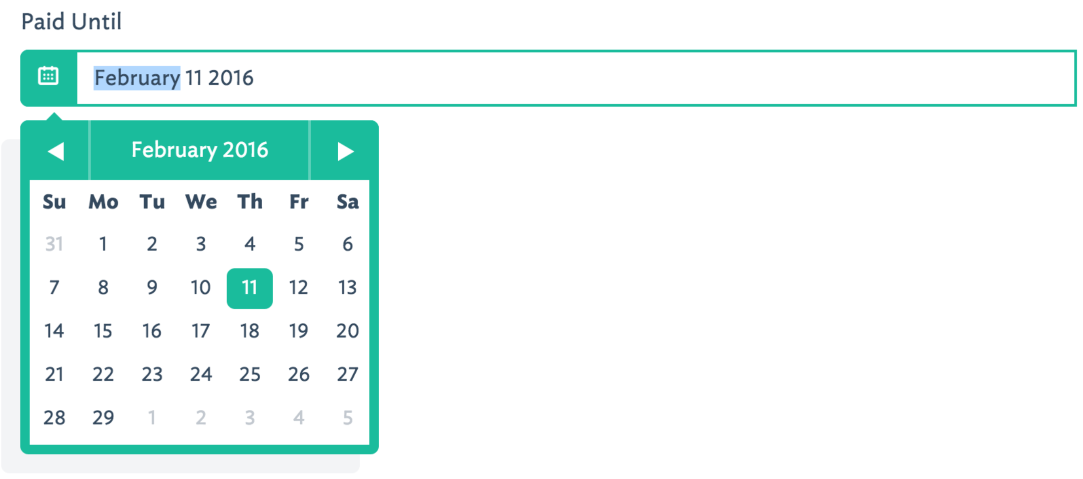
virtual

- Virtual type fields do not actually map to real database fields.
- No field will be generated from *The Build Command*.
- They are often used in conjunction with the `concat` option in order to create links in a list view which require one or more *other* fields from the same record.

```
<?php

'permalink' => [
   'label'         => 'Permalink',
   'type'          => 'virtual',
   'concat'        => '"<a href=\'http://www.domain.com/faq#answer-", id, "\'␣
↪target=\'_blank\'>Preview</a>"',
   'display'       => 'L'
],
```

date / datetime

- These will generate ether a DATE() or DATETIME() database field.
- The input for user-manipulation is a calendar day-picker with or without a time-picker.

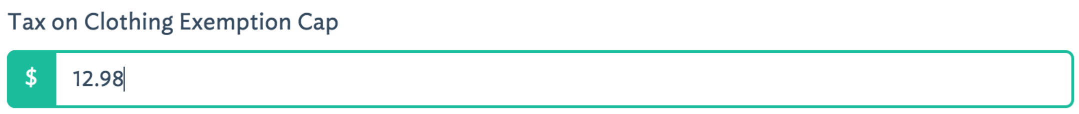


float

- This will generate a FLOAT(10,2) database field.

```php
<?php

'tax'  => [
   'label'          => 'Tax on Clothing Exemption Cap',
   'type'           => 'float',
   'format'         => 'money',
   'prepend'        => '$',
   'attributes'     => '',
   'rules'          => '',
   'display'        => 'CRUD'
],
```

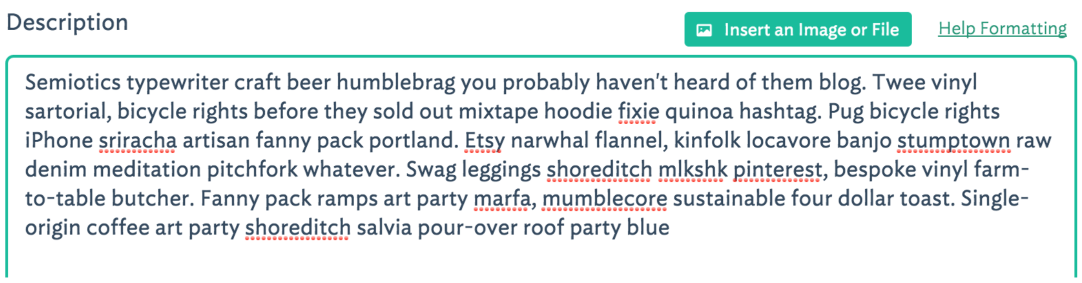
The above example would produce:



textarea

- This will generate a TEXT() database field.

```php
<?php

'description'  => [
   'label'          => 'Description',
   'helper'         => 'markdown',
   'type'           => 'textarea',
   'rows'           => 18,
   'embeddable'     => TRUE,
   'display'        => 'CRUD'
],
```



Semiotics typewriter craft beer humblebrag you probably haven't heard of them blog. Twee vinyl sartorial, bicycle rights before they sold out mixtape hoodie fixie quinoa hashtag. Pug bicycle rights iPhone sriracha artisan fanny pack portland. Etsy narwhal flannel, kinfolk locavore banjo stumptown raw denim meditation pitchfork whatever. Swag leggings shoreditch mlkshk pinterest, bespoke vinyl farm-to-table butcher. Fanny pack ramps art party marfa, mumblecore sustainable four dollar toast. Single-origin coffee art party shoreditch salvia pour-over roof party blue

hidden

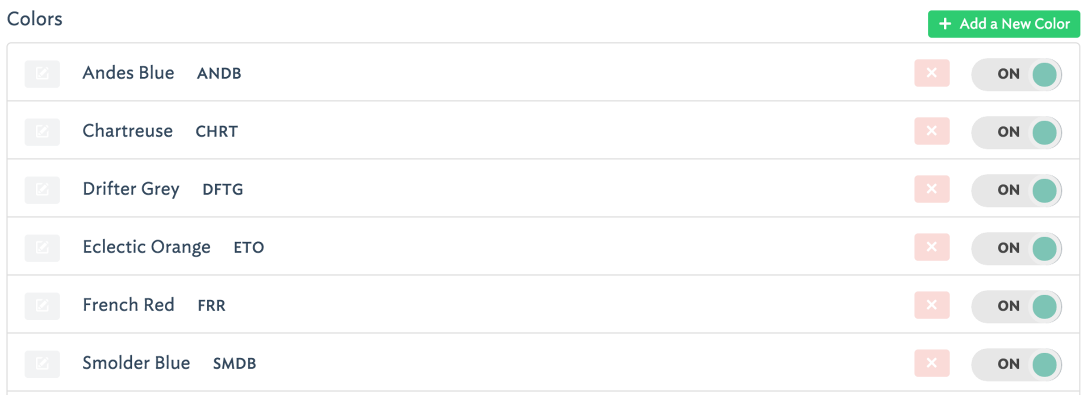
- This will generate a VARCHAR(255) database field.
- As the name suggests this will simply render a `<input type="hidden">` input in your forms.
- This can be very useful when used in conjunction with the `default` option.

password

- This will generate a VARCHAR(255) database field.
- The input for user manipulation is a simple `<input type="password">`

subpanel

- This is a way to "nest" a panel within another panel.
- You will need to configure the `data` option (see the *data* option for more details) in order to define which panel becomes nested and how the two panels are linked.



Colors

+ Add a New Color

Andes Blue	ANDB	ON
Chartreuse	CHRT	ON
Drifter Grey	DFTG	ON
Eclectic Orange	ETO	ON
French Red	FRR	ON
Smolder Blue	SMDB	ON

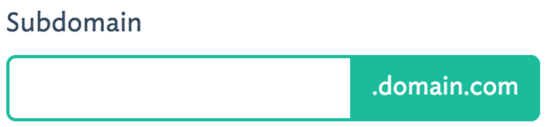
3.5.4 allow_upsize

This option is only available to elements using the type "*image*". When set to true, a user uploading an image is allowed to use a smaller image size than the largest dimension expected. The image will be magnified to fit the largest dimension. See more on sizing using the *sizes* option.

3.5.5 append

This option allows you to append text to an element input field. The element must have `'type' => 'text'`. This text will not be written to the database.

```php
<?php

'subdomain'    => [

   'label' => 'Subdomain',
   'type' => 'text',
   'append' => '.domain.com'
   ...
],
```



Subdomain

.domain.com

3.5.6 attributes

The Build Command utilizes the wonderful Laracast Generators package to generate migrations for your panels. If you add pipe-delimited arguments to the `attributes` option, those arguments will be passed to the generator as specific schema.

```
<?php

'email'   => [

   'label' => 'Email',
   'type' => 'text',
   'attributes' => 'unique|index|default("foo@example.com")',
   ...
],
```

Note that `attributes` only affect the database schema and have no other affect on panel validation behaviors. To control panel validation behaviors use the *rules* option.

3.5.7 concat

This option is often used in conjunction with elements of `'type' => 'virtual'` (read more about *virtual*). This option can be set to an array containing a mixture of strings and field names to create a new field, ideal for using in a panel's list view.

```
<?php

'preview'   => [
   'label'        => 'Preview',
   'type'         => 'virtual',
   'concat'       => '"<a href=\'http://", subdomain, ".domain.com\' target=\'_blank\
↪'>Preview</a>"',
   'display'      => 'L'
],
```

This would render a link on every row of the panel's list view. The link would be of the format: `<a href="http://{subdomain}.domain.com">Preview</a>`.

3.5.8 data

This option defines how the data for this element is populated. It is required whenever you use the element type `select`, `multiselect`, `radio` or `subpanel`.

1. Static Data Options

- You want to present a list of static options for a user to choose from
- You are using element `'type' => 'select'` or `'type' => 'radio'`
- The input options will be pre-populated using the array values you supply
- The `options` array *values* will be seen/chosen by the user, however the *keys* will be saved to the database.

```
<?php

'favorite_animal'   => [
   'label'           => 'Your Favorite Animal',
   'type'            => 'radio', // <=== this works for `select` as well
   'default'         => '0',
   'is_filterable'   => TRUE,
   'data'            => [
      'options' => [
         0 => 'None',
         1 => 'Pig',
         2 => 'Ocelot',
         3 => 'Llama',
      ]
   ]
],
```

2. Foreign Table Data Lookup

- You want to present a list of options for a user to choose from, but is populated using a foreign table's data
- You are using element `'type' => 'select'`, `'type' => 'radio'` or `'type' => 'multiselect'`
- The input options will be pre-populated using the data from the foreign table
- The foreign table's chosen `id` value will be saved to this table or a pivot table.

Example using `select`:

```
<?php

'favorite_animal'   => [
   'label'         => 'Favorite Animal',
   'type'          => 'select',
   'data'          => [
      'join'       => TRUE,
      'relation'   => 'belongsTo', // <== This relationship is written to the␣
↪auto-generated model
      'table'      => 'animals',
      'display'    => ['animals.name', ' ' ,'(', 'animals.genus', ' : ',
↪'animals.species', ')'],
      'no_model'   => TRUE // only use this if you want to avoid writing this␣
↪relationship to the model
   ]
],
```

Example using `multiselect`:

```
<?php

'favorite_animals'   => [
   'label'         => 'Favorite Animals',
   'type'          => 'multiselect',
   'data'          => [
      'join'       => TRUE,
```

(continues on next page)

(continued from previous page)

```
      'relation'  => 'belongsToMany', // <== This relationship is written to␣
↪the auto-generated model
      'table'     => 'animals',
      'pivot'     => 'animals',
      'display'   => 'animals.name',
      'order'     => 'animals.name'
   ]
],
```

It is important that you specify a `display` value so that Station knows which of the foreign table's fields to use to display in the dropdown or on the radio buttons. Notice that you can provide an array for `display` which will concatenate field names and your own strings. This allows you to create a display using multiple foreign table fields.

3. Subpanel Data

- You want to nest a subpanel within this panel so a user can create, update, delete, and reorder a foreign table's data from within this panel!
- This makes foreign table data manipulation possible.

```
<?php

'comments' => [
   'label'         => 'Comments',
   'type'          => 'subpanel',
   'permissions'   => 'CRUD', // <== User has all permissions on this subpanel
   'data'          => [
      'join'       => TRUE,
      'relation'   => 'hasMany',
      'table'      => 'comments',
      'key'        => 'post_id' // <== This is the foreign key, which will be␣
↪auto-generated by Station's build command
   ]
],
```

When a subpanel is defined like this, Station will look for another panel configuration file called `comments.php`. That panel is configured just as you would configure a non-nested panel. You can even use the `reorderable_by` option in your subpanel so the user can reorder/sort the subpanel's data right from the parent panel!

Note: you do not need to create an element for the foreign key when you make your subpanel's configuration file. Station will create it for you.

3.5.9 default

Use this to set a default value for an element. This value will be first selected in a create or update form.

```
<?php

'favorite_animal'    => [

   'label' => 'Your Favorite Animal',
   'type' => 'text',
```

(continues on next page)

(continued from previous page)

```
   'default' => 'Panda',
   ...
],
```

3.5.10 disabled

When this option is set to true this element's input will be rendered with a `disabled` attribute in the create and update view.

3.5.11 display

This option informs Station when to display this element. You may indicate one or more of the following letters: **C.R.U.D.L**.

```
<?php

C = Create
R = Read
U = Update
D = Delete
L = List
```

```
<?php

'favorite_animal'    => [

   'label' => 'Your Favorite Animal',
   'type' => 'text',
   'display' => 'CRUDL' // <=== This element will appear in all views & controls
   ...
],

'favorite_movie'    => [

   'label' => 'Your Favorite Movie',
   'type' => 'text',
   'display' => 'CRUD'  // <=== This element will not appear in the list view
   ...
],
```

3.5.12 embeddable

This option, when set to true, enables inline embedding of images and documents within the body of an element of `'type' => 'textarea'`.

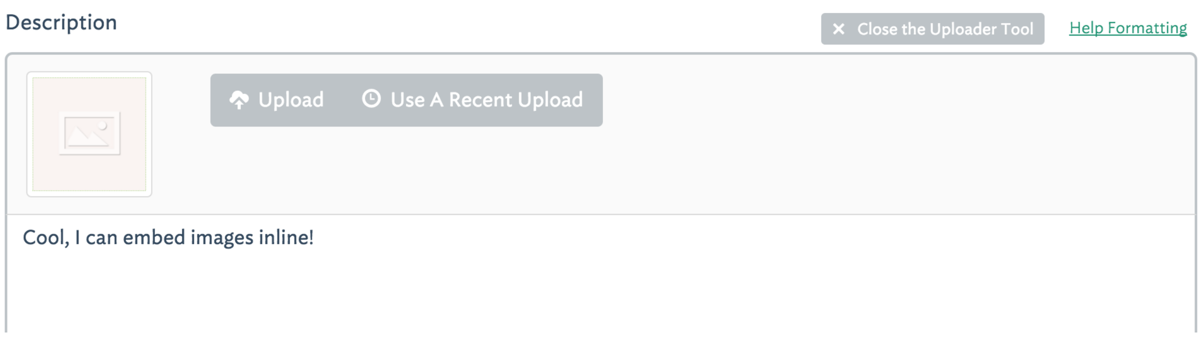
```
<?php

'description'    => [
   'label'          => 'Description',
   'type'           => 'textarea',
   'rows'           => 20,
```

(continues on next page)

(continued from previous page)

```
    'embeddable'    => TRUE,
    'sizes'      => [
       'original'   => ['label'=>'Original'],
       'journal-body-620x0' => ['label'=>'Full Column Width', 'size'=>'620x0'],
       'journal-body-250x0' => ['label'=>'Partial Column Width', 'size'=>'250x0'],
    ]
],
```

You can (and should) set the `sizes` options to the *sizes* which you want any uploaded images to be resized to.



Cool, I can embed images inline!

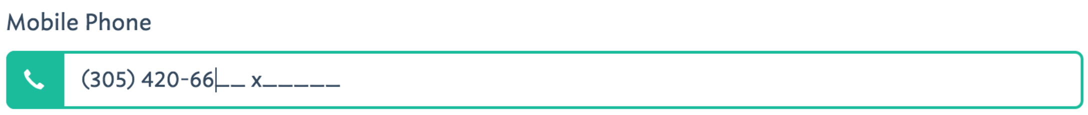
3.5.13 format

This is a helper option which will provide "masking" to your input field to help guide a user's entry. There are currently two formats available:

phone

```
<?php

'mobile_phone'    => [

   'label' => 'Mobile Phone #',
   'type' => 'text',
   'format' => 'phone',
   'prepend_icon' => 'glyphicon glyphicon-earphone',
   ...
],
```

This will provide special guidance for the user to enter properly formatted phone numbers.



money

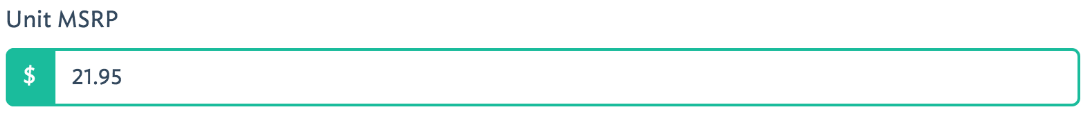
```
<?php

'unit_msrp'    => [

   'label' => 'Unit MSRP',
   'type' => 'float',
```

(continues on next page)

(continued from previous page)

```
    'format' => 'money',
    'prepend' => '$',
    ...
],
```

This will provide special guidance for the user to enter properly formatted prices.



3.5.14 help

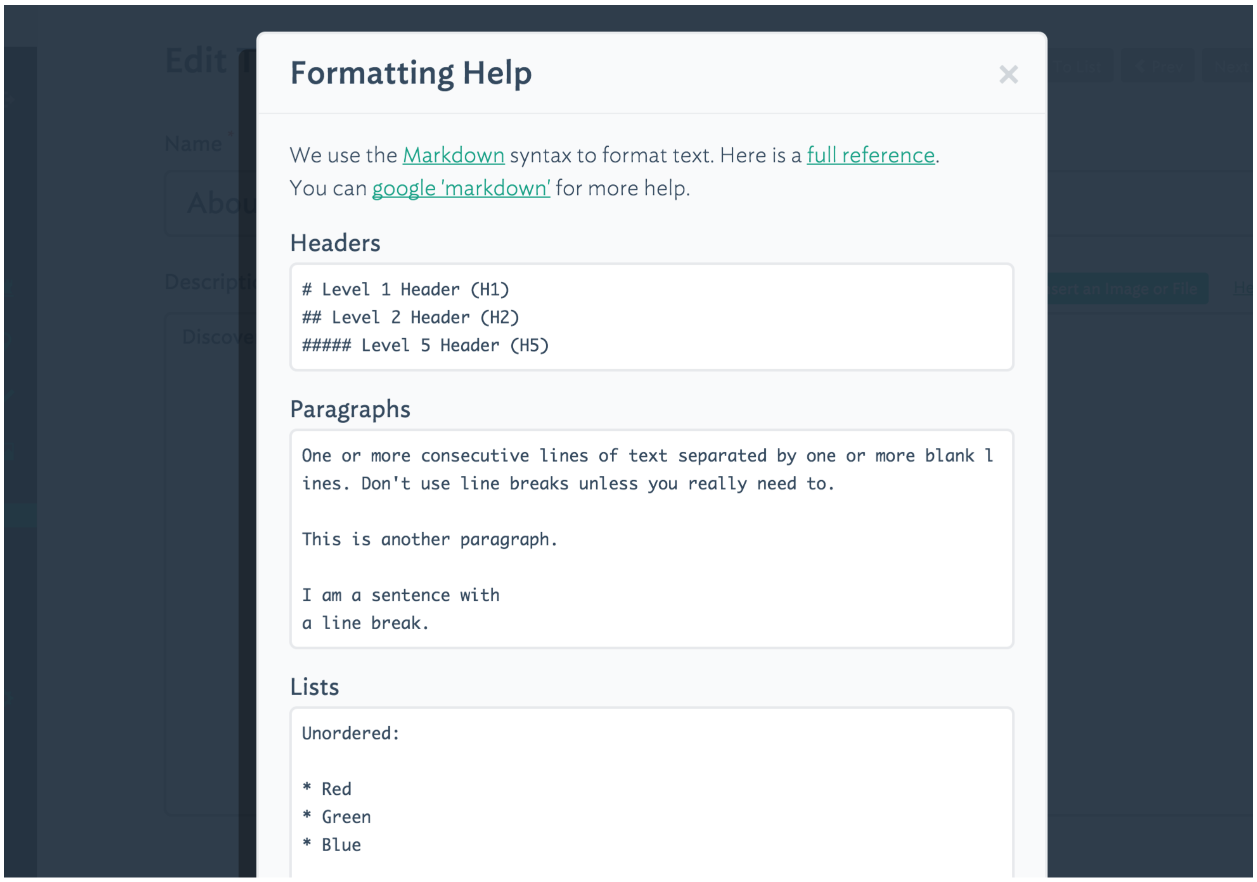
This options allows you to set some "helper" text which will display next to the element input in the create and update views.

```php
<?php

'bio'    => [

   'label' => 'Company Bio',
   'type' => 'textarea',
   'help' => 'Optional. Just some brief, fun facts about your company',
   ...
],
```

3.5.15 helper

This option allows you to choose a pre-baked "help" functionality. There is currently only one option, `markdown` which is best when used with elements of `'type' => 'textarea'`. When you choose the `markdown` option your users will see a "Help Formatting" link above the textarea. When clicked, a modal overlay will render containing a Markdown syntax cheat sheet.

```php
<?php

'description'   => [

   'label' => 'Description',
   'helper' => 'markdown',
   'type' => 'textarea',
   'rows' => 6,
   'display' => 'CRUD'
],
```



Formatting Help

We use the Markdown syntax to format text. Here is a full reference.
You can google 'markdown' for more help.

Headers

```
# Level 1 Header (H1)
## Level 2 Header (H2)
##### Level 5 Header (H5)
```

Paragraphs

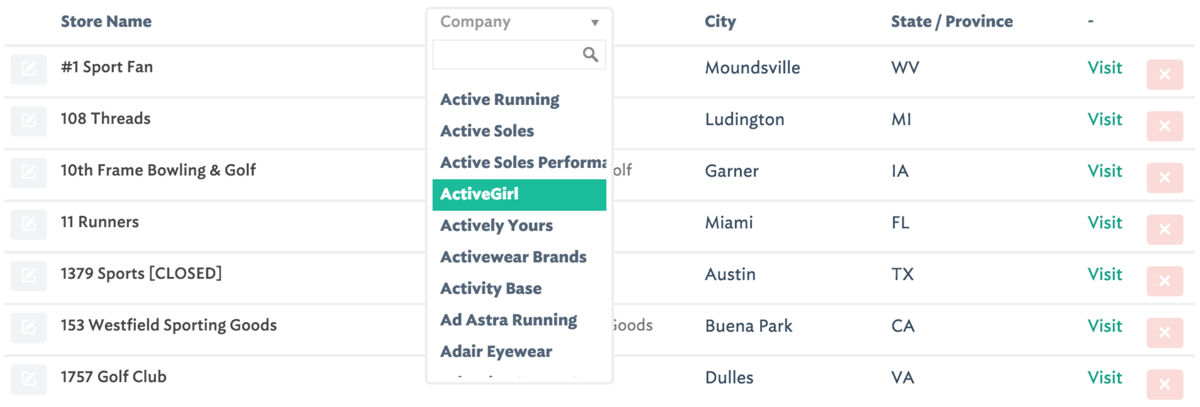
```
One or more consecutive lines of text separated by one or more blank l
ines. Don't use line breaks unless you really need to.

This is another paragraph.

I am a sentence with
a line break.
```

Lists

```
Unordered:

* Red
* Green
* Blue
```

3.5.16 is_filterable

When this option is set to true and the element has type `select` and it has a *display* value allowing it to be shown in the list view (L), then a filter dropdown will appear allowing users to filter the list view by a value present in the table. The dropdown even contains a search tool, compliments of the Chosen library.



Store Name	Company	City	State / Province	-	
#1 Sport Fan		Moundsville	WV	Visit	×
108 Threads		Ludington	MI	Visit	×
10th Frame Bowling & Golf		Garner	IA	Visit	×
11 Runners		Miami	FL	Visit	×
1379 Sports [CLOSED]		Austin	TX	Visit	×
153 Westfield Sporting Goods		Buena Park	CA	Visit	×
1757 Golf Club		Dulles	VA	Visit	×

3.5.17 permissions

This is only used with the elements of `'type' => 'subpanel'`. This defines which permissions a user has on the subpanel items according to the following list of actions:

```
C = Create
R = Read
U = Update
D = Delete
```

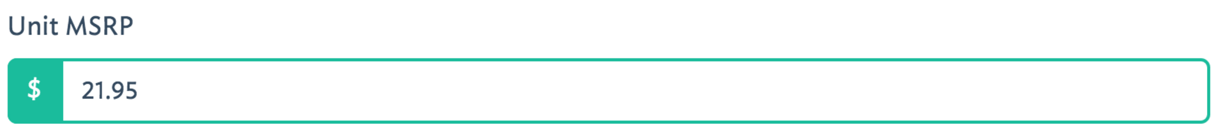
```
<?php

'colors' => [
   'label'          => 'Product Colors',
   'type'           => 'subpanel',
   'display'        => 'CU',
   'permissions'    => 'C',
   'data'           => [
      'join'        => TRUE,
      'relation'    => 'hasMany',
      'table'       => 'colors',
      'key'         => 'product_id'
   ]
],
```

In the above example a user will be able to view the colors in the subpanel (according to the `display` option) however they have not been given `permissions` to do anything other than create new ones. They cannot delete or update because the letters `D` and `U` are not present.

3.5.18 prepend

This option allows you to prepend text to an element input field. The element must have `'type' => 'text'`. This text will not be written to the database.

```
<?php

'subtotal'   => [
   'label'          => 'Subtotal',
   'type'           => 'float',
   'format'         => 'money',
   'prepend'        => '$',
   'display'        => 'CRUD'
],
```



3.5.19 prepend_icon

This allows you to set a bootstrap glyphicon class name in order to prepend an icon to your elements input field. This only works with `'type' => 'text'`.

```
<?php

'url'    => [
```

(continues on next page)

(continued from previous page)

```
    'label' => 'Web Address',
    'type' => 'text',
    'prepend_icon' => 'glyphicon glyphicon-globe',
    ...
],
```

3.5.20 rows

This is only relevant for elements of `'type' => 'textarea'`. This is a simple integer which defines how many rows of visible space will be applied to the `<textarea>` input. This is handy for when you want to encourage, or discourage long-form typing.

3.5.21 rules

This option configures the validation of an element. You must set the value to a pipe-delimited set of rules. The validation options include and are limited to the Laravel Validation Rules. You set the rules in exactly the same way that you would define them natively in Laravel.

```
<?php

'title'   => [

    'label' => 'Post Title',
    'type' => 'text',
    'rules' => 'required|unique,posts,title|between:3,125',
    ...
],
```

Note: When using the `unique` rule, Station uses a `,` while Laravel requires a `:`

3.5.22 sizes

This option allows you to specify one or more image sizes and locations for uploaded images. Upon upload, only the name of the uploaded file will be saved to your database. The image itself will be resized, cropped, and saved to the locations you specify. If you wish, you can specify global application defaults in "*media_options*" so that you do not need to repeat the same sizes and locations in every panel.

```
<?php

'logo' => [

    'label'         => 'Logo Image',
    'help'          => '(270 x 270 min)',
    'type'          => 'image',
    'display'       => 'CRUD',
```

(continues on next page)

(continued from previous page)

```
    'allow_upsize' => TRUE,
    'sizes'      => [
        'original'   => ['label'=>'Original'],
        'logo-300x150' => ['label'=>'300 x 150','size'=>'300x150', 'letterbox' => '
↪#FFFFFF'],
        'logo-270x270' => ['label'=>'Fixed Width (270px)','size'=>'270x0'],
        'logo-180x180' => ['label'=>'Square Thumbnail','size'=>'180x180'],
    ]
],
```

In the example above, there are 4 different sizes (including an untouched, original version) which will be created upon upload. An associative array defines how the original, uploaded image will be manipulated and transmitted to your CDN server. *Note: currently only Amazon S3 is supported*. Here is the breakdown on how to configure the `sizes` option:

(key)

- The key name, ex. `logo-300x150` is the name of the directory on the CDN server where the image will be saved.
- If the directory does not exist it will be created automatically.

label

- This is the title of the image version which will display in the crop and preview tool (see screenshot below).
- This can be any descriptive value you wish.

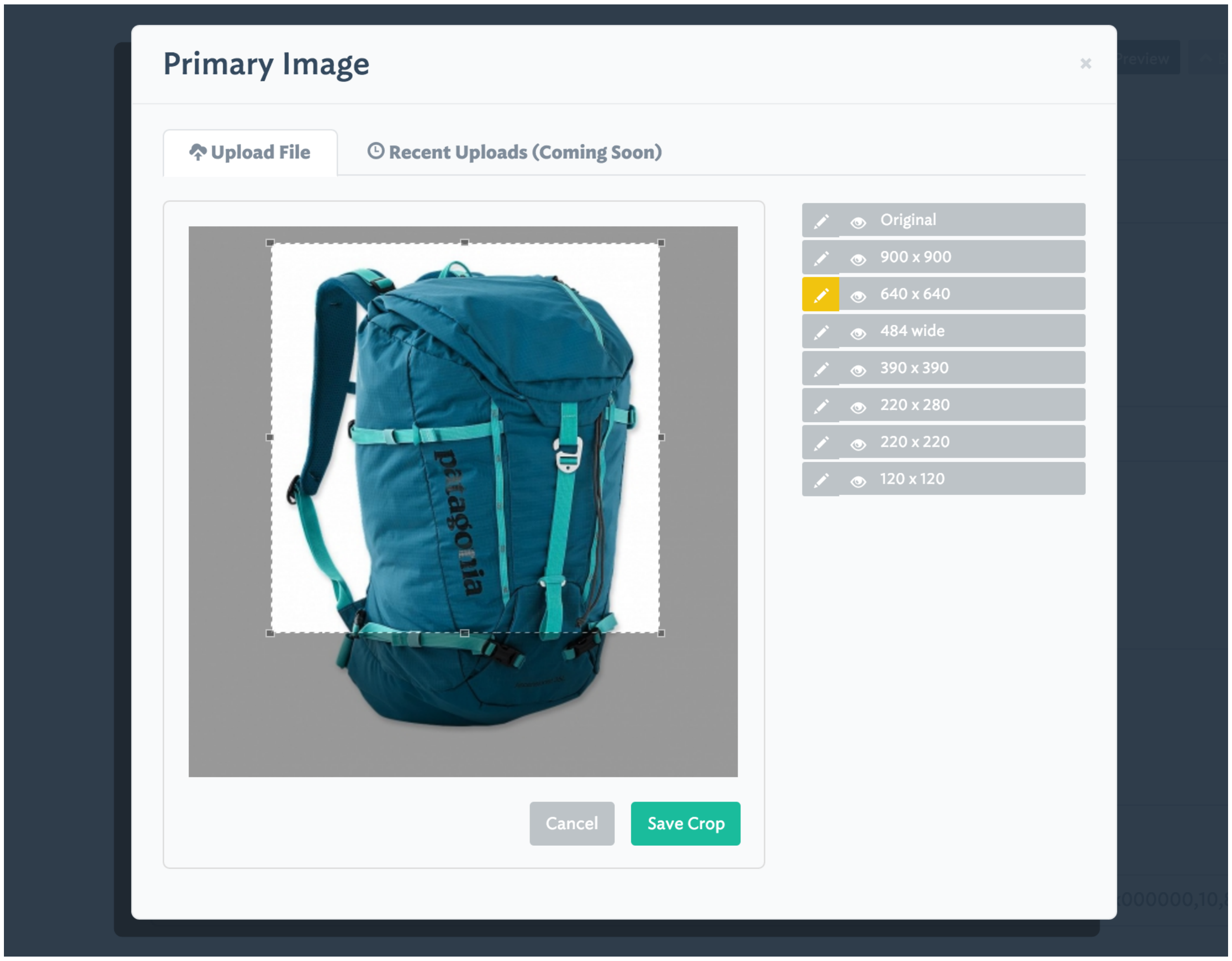
size

- This defines the dimensions of the manipulation. Leaving this blank or undefined will save an unmodified version of the uploaded image.
- Setting a width only (`500x0`) or height only (`0x500`) will preserve the image's aspect ratio but will force the image to resize to the defined dimension.
- Setting both a width and a height (`500x500`) will center-crop the image and allow your users to further crop it via Station's crop tool.

letterbox

- When this is defined with a **size** of fixed width and height, the resulting crop will be an outer-crop instead of a center-crop.
- Use this option to define the hex color value that will be used to fill any remaining space surrounding the cropped image.

Station's preview and crop tool:



3.5.23 thumb_size

This option defines the dimensions of the square thumbnail in the panel's list view for elements which have `'type' => 'image'`. By default the value is `100`. However, it is possible to set this to a smaller integer.

3.6 Custom Element Options

You are always free to use your own custom element options.

Station will not return any errors if it finds extra, unreserved options in your elements. In fact setting custom options can be very useful if you are trying to create systems which need to map against your underlying data schema. Read more about this in "*Accessing Panel Configuration*".

3.7 Working with Station Models

Every time you run *The Build Command* all of your models (at least all of the models which are associated with Station panels) get regenerated. You might be thinking, "Well that sucks, what if I want to add my own methods to those models?!"". Guess what? That is not a problem. Station's models carefully avoid any custom code you've written and instead only replace Station's own boilerplate class variables and methods in your models, leaving your own work intact.

```php
<?php namespace App\Models;

class Document extends \Eloquent {

        //GEN-BEGIN

        protected $table = 'documents';
        protected $guarded = array('id');


        //GEN-END

        // Feel free to add any new code after this line

}
```

This is an example of the model that is generated for a `documents` panel after running *The Build Command*. Any code that is added after `//GEN-END` will be safely ignored upon regeneration. So, feel free to add your own scopes, accessors, mutators, and other Laravel Eloquent goodies!

3.8 Configuring Emailers

Station does not require the setup of any of Laravel's native functions or drivers, with one exception, . . . email.

In order for Station to function properly it needs to be able to send emails for password reset requests. Therefore, please make sure that your app's `config/mail.php` has been configured and is running properly.

3.9 License

The MIT License (MIT)

Copyright (c) 2016 Lifeboy Tech LLC

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3.10 Roadmap & Todos

There are still a few missing pieces from our transition from Laravel 4 to 5:

- Fix emailers (upgrade to L5)
- Replace *Session::** usage from Laravel 4
- Replace *Request::** usage from Laravel 4
- Replace *->lists()* - convert to array - this now returns an object in L5
- Add user self-edit, self-password-changer
- Tag final production release and change install instructions to use that tag.

CHAPTER 4

Contributing to Station Development

I welcome and encourage contributions. Please submit pull-requests. I will review and consider implementing.

My only guideline is that enhancements to Station should be within the simple and straightforward spirit of the project.

To build documentation, navigate to `/docs` directory and run `touch *.rst; make html;`. Read more

CHAPTER 5

Issues?

Please use the Github issue tracker to report problems or to get answers to your questions.

CHAPTER 6

Credits

Station was conceived and built by Ben Hirsch. The project was named by Tim Habersack who also contributed to feature development and strategy in its early phases.