Documentation 3.4

About

Introduction
Demo
Installation
Getting Started
1. Basics
2. CRUD Operations
3. Advanced Features
4. Add-ons, License & Support
Release Notes
Upgrade Guide

Backpack\Base

About
How To

Backpack\CRUD

Basics
Tutorial
Cheat Sheet
Operations
ListEntries
Columns
Buttons
Filters
Create & Update
Fields
Delete
Show
Columns
Reorder
Revisions
How To

Add-ons

Official Add-ons
Community Add-ons

API Cheat Sheet

Edit

Here are all the functions you will be using **inside your EntityCrudController's `setup()` method**, grouped by the operation you will most likely use them for.

## Operations

### ListEntries

#### Columns

```
// Manipulate what columns are shown in the table view.
$this->crud->addColumn($column_definition_array); // add a column, at the end of the stack
$this->crud->addColumns([$column_definition_array, $another_column_definition_array]); // add multiple columns, at the end of the stack
$this->crud->modifyColumn($name, $modifs_array);
$this->crud->removeColumn('column_name'); // remove a column from the stack
$this->crud->removeColumns(['column_name_1', 'column_name_2']); // remove an array of columns from the stack
$this->crud->setColumnDetails('column_name', ['attribute' => 'value']);
$this->crud->setColumnsDetails(['column_1', 'column_2'], ['attribute' => 'value']);

$this->crud->setColumns(); // set the columns you want in the table view, either as array of column names, or multidimensional array with all columns detailed with their types

// ------ REORDER COLUMNS
$this->crud->addColumn()->beforeColumn('name'); // will show this before the given column
$this->crud->addColumn()->afterColumn('name'); // will show this after the given column

$this->crud->addColumn()->makeFirstColumn();
  // will make this column the first one in the list
  // you need to also specify priority 1 in your addColumn statement for details_row or responsive expand buttons to show
```

#### Buttons

```
// possible positions: 'beginning' and 'end'; defaults to 'beginning' for the 'line' stack, 'end' for the others;
$this->crud->addButton($stack, $name, $type, $content, $position); // add a button; possible types are: view, model_function
$this->crud->addButtonFromModelFunction($stack, $name, $model_function_name, $position); // add a button whose HTML is returned by a method in the CRUD model
$this->crud->addButtonFromView($stack, $name, $view, $position); // add a button whose HTML is in a view placed at resources\views\vendor\backpack\crud\buttons
$this->crud->removeButton($name);
$this->crud->removeButtonFromStack($name, $stack);
```

#### Filters

```
// Manipulate what filters are shown in the table view.
//
// Note: check out CRUD > Features > Filters in the docs to see examples of $filter_definition_array
$this->crud->addFilter($filter_definition_array, $values, $filter_logic);
$this->crud->modifyFilter($name, $modifs_array);
$this->crud->removeFilter($name);
$this->crud->removeAllFilters();
$this->crud->filters(); // gets all the filters
```

Details Row

```
// Shows a + sign next to each table row, so that the user can expand that row and reveal details. You are responsible for creting the view with those details.
$this->crud->enableDetailsRow();
// NOTE: you also need to do allow access to the right users: $this->crud->allowAccess('details_row');
// NOTE: you also need to do overwrite the showDetailsRow($id) method in your EntityCrudController to show whatever you'd like in the details row OR overwrite the views/backpack/crud/details_row.blade.php

$this->crud->disableDetailsRow();
```

Export Buttons

```
// Show export to PDF, CSV, XLS and Print buttons on the table view. Please note it will only export the current _page_ of results. So in order to export all entries the user needs to make the current page show "All" entries from the top-left picker.
$this->crud->enableExportButtons();
```

Responsive Table

```
$this->crud->disableResponsiveTable();
$this->crud->enableResponsiveTable();
```

Page Length

```
$this->crud->setDefaultPageLength(10); // number of rows shown in list view
$this->crud->setPageLengthMenu([100, 200, 300]); // page length menu to show in the list view
```

Actions Column

```
// make the actions column (in the table view) hide when not enough space is available, by giving it an unreasonable priority
$this->crud->setActionsColumnPriority(10000);
```

Custom / Advanced Queries

```
// Change what entries are show in the table view.
// This changes all queries on the table view,
// as opposed to filters, who only change it when that filter is applied.
$this->crud->addClause('active'); // apply local scope
$this->crud->addClause('type', 'car'); // apply local dynamic scope
$this->crud->addClause('where', 'name', '=', 'car');
$this->crud->addClause('whereName', 'car');
$this->crud->addClause('whereHas', 'posts', function($query) {
     $query->activePosts();
 });
$this->crud->groupBy();
$this->crud->limit();

$this->crud->orderBy();
// please note it's generally a good idea to use crud->orderBy() inside "if (!$this->request->has('order')) {}"; that way, your custom order is applied ONLY IF the user hasn't forced another order (by clicking a column heading)
```

Show

Use the same Columns API as for the ListEntries operation, but inside your `show()` method.

Create & Update Operations

```
// ------
// FIELDS
// ------
// Manipulate what fields are shown in the create / update forms.
//
// Note: check out CRUD > Features > Field Types in the docs to see examples of $field_definition_array

$this->crud->addField($field_definition_array, 'update/create/both');
$this->crud->addField('db_column_name', 'update/create/both'); // a lazy way to add fields: let the CRUD decide what field type it is and set it automatically, along with the field label
$this->crud->addFields($array_of_fields_definition_arrays, 'update/create/both');
$this->crud->modifyField($name, $modifs_array, 'update/create/both');
$this->crud->removeField('name', 'update/create/both');
$this->crud->removeFields($array_of_names, 'update/create/both');
$this->crud->removeAllFields();

// Note: the last parameter is always the form - create or update; if missing, it's assumed 'both';

// ------ REORDER FIELDS
$this->crud->addField()->beforeField('name'); // will show this before the given field
$this->crud->addField()->afterField('name'); // will show this after the given field
```

Reorder

```
// Show a reorder button in the table view, next to Add
// Provide an interface to reorder & nest elements, provided the parent_id, lft, rgt, depth columns are in the database, and fillable on the model.
$this->crud->enableReorder('label_name', 3);
```

```
// NOTE: the second parameter is the maximum nesting depth; this example will prevent the user from creatin
g trees deeper than 3 levels;
// NOTE: you also need to do allow access to the right users: $this->crud->allowAccess('reorder');

$this->crud->disableReorder();
$this->crud->isReorderEnabled(); // return true/false
```

Revisions

```
// ------------------------
// REVISIONS aka Audit Trail
// ------------------------
// Tracks all changes to an entry and provides an interface to revert to a previous state.
//
// IMPORTANT: You also need to use \Venturecraft\Revisionable\RevisionableTrait;
// Please check out: https://backpackforlaravel.com/docs/crud-operation-revisions
$this->crud->allowAccess('revisions');
```

All Operations

```
// ------
// ACCESS
// ------
// Prevent or allow users from accessing different CRUD operations.

$this->crud->allowAccess('list');
$this->crud->allowAccess(['list', 'create', 'delete']);
$this->crud->denyAccess('list');
$this->crud->denyAccess(['list', 'create', 'delete']);

$this->crud->hasAccess('add'); // returns true/false
$this->crud->hasAccessOrFail('add'); // throws 403 error
$this->crud->hasAccessToAll(['create', 'update']); // returns true/false
$this->crud->hasAccessToAny(['create', 'update']); // returns true/false

// ------------
// EAGER LOADING
// ------------

// eager load a relationship
$this->crud->with('relationship_name');

// -----------
// CUSTOM VIEWS
// -----------

// use a custom view for a CRUD operation
$this->crud->setShowView('your-view');
$this->crud->setEditView('your-view');
$this->crud->setCreateView('your-view');
$this->crud->setListView('your-view');
$this->crud->setReorderView('your-view');
$this->crud->setRevisionsView('your-view');
$this->crud->setRevisionsTimelineView('your-view');
$this->crud->setDetailsRowView('your-view');

// -------
// GETTERS
// -------

$this->crud->getEntry($entry_id);
$this->crud->getEntries();

$this->crud->getFields('create/update/both');

// in your update() method, after calling parent::updateCrud()
$this->crud->getCurrentEntry();

// -------
// ACTIONS
// -------

$this->crud->getActionMethod(); // returns the method on the controller that was called by the route; ex: c
reate(), update(), edit() etc;
$this->crud->actionIs('create'); // checks if the controller method given is the one called by the route

// --------------------------
// CrudPanel Basic Information
// --------------------------
$this->crud->setModel("App\Models\Example");
$this->crud->setRoute("admin/example");
// OR $this->crud->setRouteName("admin.example");
$this->crud->setEntityNameStrings("example", "examples");

// check the FormRequests used in that EntityCrudController for required fields, and add an asterisk to the
m in the create/edit form
$this->crud->setRequiredFields(StoreRequest::class, 'create');
$this->crud->setRequiredFields(UpdateRequest::class, 'edit');
```

Please purchase a commercial license or apply for a non-commercial license code *before* you put your project online, to prevent nagging notification bubbles.