Installing Backpack for Laravel

Installing the Backpack admin panel is straightforward, but there are a few extra steps you'll likely want to take.

Pro tip!

Add the following to your shell configuration file: `alias sail='bash vendor/bin/sail'`

Set Up Admin Users

The first thing you'll want to do is to create the concept of user roles in your project. Let's start by creating a migration for this in order to add a `role` field to your `users` table:

```
sail artisan make:migration add_role_to_users_table
```

Open the new migration file and add the new field:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
  public function up()
  {
    Schema::table('users', function (Blueprint $table) {
      $table->string('role');
    });
  }

  public function down()
```

```
  {
    Schema::table('users', function (Blueprint $table) {
      $table->dropColumn('role');
    });
  }
};
```

⚠ Notes

- You may be tempted to make `role` an `enum` field. This is generally not advisable, so please make it a normal `string` field instead. Also, do not provide a default value, as this is typically better handled in our application layer.
- A common alternative implementation is to set a `boolean` `is_admin` field. This is fine, but offers less flexibility for potential features down the line.

Next, let's create a *PHP enum* to help keep track of user roles:

```
<?php
/**
 * app/Enums/UserRole.php
 */

namespace App\Enums;

enum UserRole: string
{
  case User = 'USER';
  case Admin = 'ADMIN';
}
```

Next, let's change the `User` model to support this new field.

```
<?php

namespace App\Models;

// ...

class User extends Authenticatable
{
  // ...
```

```
  protected $attributes = ['role' => UserRole::User];

  protected $fillable = ['name', 'email', 'password', 'role'];

  protected $casts = [
    // ...
    'role' => UserRole::class,
  ];
}
```

The `$attributes` property allows you to specify a default value for fields on a model. Here, we are making users non-admins by default.

The `$casts` property allows us to specify how to cast values from the database to PHP classes. In this case, we're turning plain strings (`USER` and `ADMIN`) into PHP enums.

The `$fillable` property allows us to specify which fields can be "mass assigned".

Download and Install Backpack

Now that you have admin users set up, download Backpack from the Composer. Run the following in your project's root folder:

```
# This assumes you have `sail` aliased to `vendor/bin/sail`
sail composer require backpack/crud
```

After that, run the installation command and follow the prompts:

```
sail artisan backpack:install
```

- When asked if you'd like to create a new admin user, say `no`.
- When asked if you'd like to install premium functionality, choose `Backpack Pro`. The credentials can be found in Keybase (go to "Files" under the `veganhacktivists.admin` team.

Now, you've officially got Backpack installed! You're almost done.

Keeping the Repository Clean

You may have noticed that Backpack dumps a huge number of files into `public/packages`. These don't really need to be checked in to the repository, so let's fix that by running the following command:

```
cd public && ln -s ../vendor/backpack/crud/src/public/packages packages && cd -
```

This creates a symlink from `public/packages` to the package's own copy of the directory.

Thanks to Joaquín for sharing this tip!

Configuring and Customizing Backpack

By default, Backpack has a login routes, which requires you to log in to access the admin panel even if you're already logged in as an admin user. Let's change that to make things more convenient by editing `config/backpack/base.php`:

```
<?php

return [
  // ...
  'setup_auth_routes' => false, // (change from `true`)

  'guard' => null, // (change from 'backpack')
];
```

Now, you will only need to log in once. However, we now need to define a custom logout route for Backpack. We do this in `routes/backpack/custom.php`:

```
<?php

// ...
use Laravel\Fortify\Http\Controllers\AuthenticatedSessionController;

Route::group(
  [/* Leave this alone */],
  function () {
    // ...
    Route::get('logout', [AuthenticatedSessionController::class, 'destroy']);
```

```
  }
); // this should be the absolute last line of this file
```

Thanks to Jeremy for sharing this tip!

⚠ Notes

- This snippet assumes that you have Laravel Fortify installed. If you are using Laravel Jetsream (which is recommended), this is included.
- Typically, we want session management to handled inside of a `POST` request, but Backpack expects it to be a `GET`, hence why we do it that way in this instance.
- Because this new route was created in `routes/backpack/custom.php`, the URI will automatically be prepended with `/admin/`, so the full path to the logout route will be `/admin/logout`.

Logging out from the admin panel should now work! Now, we need to change one more file.

Right now, all users are considered admins, so we need to fix that. Also, if you try to access `/admin` while being logged out, it will redirect you to `/admin/login`, which no longer exists. In order to change this behavior, open `app/Http/Middleware/CheckIfAdmin.php`:

```
<?php

// ...

class CheckIfAdmin
{
  private function checkIfUserIsAdmin($user)
  {
    // USE OUR `role` FIELD
    return $user->role === UserRole::Admin;
  }

  private function respondToUnauthorizedRequest($request)
  {
    if ($request->ajax() || $request->wantsJson()) {
      return response(trans('backpack::base.unauthorized'), 401);
    } else {
      // REPLACE THIS
      // return redirect()->guest(backpack_url('login'));
      return redirect()->guest(route('login'));
    }
```

```
    }

    // ...
}
```

You're officially done setting up Backpack!

Bonus: Creating An Admin User

Now that Backpack is set up, you'll need a way to access the admin panel. First, sign up for an account using the project's registration form. After that, let's set the user's role using Tinker:

```
sail artisan tinker
```

When the prompt is ready for your input, enter the following, making sure to retrieve the correct user:

```
use App\Enums\UserRole;
$user = User::first();
$user->role = UserRole::Admin;
$user->save();
```

Exit Tinker (Ctrl + D), and you should now be able to visit `/admin` using your admin account.

Bonus: Displaying Users in Backpack

Now that everything's set up, let's set up Backpack to allow us to perform CRUD operations on our users.

First, let's create a `CrudController` for our users:

```
sail artisan backpack:crud user
```

When prompted about validation rules, the default option (`request`) is fine. After the command runs, you will have a new file to configure the new "Users" section in the admin panel: `app/Http/Controllers/Admin/UserCrudController.php`.

⚠ Note

If the command failed and complained about an undefined array key, then you likely modified the last line in your `routes/backpack/custom.php` file. Make sure that the very last is `}); // this should be the absolute last line of this file`. It's silly, but that's how they add the route to your routes file automatically.

If you visit `/admin/user`, you will see a list of your users. **Whoa**, is that a password hash we see in the table? To remove it (and to add our `role` field), let's open the new controller (`app/Http/Controllers/Admin/UserCrudController.php`):

```
<?php

namespace App\Http\Controllers\Admin;

// ...
class UserCrudController extends CrudController
{
  // ...
  use App\Enums\UserRole;

  public function setup()
  {
      // ...
  }
  protected function setupListOperation()
  {
    // Pro tip: Replace `CRUD::` with `$this->crud->` in order to allow
    //          for better intellisense
    $this->crud->column('name');
    $this->crud->column('email');
    CRUD::column('password'); // remove this line
    $this->crud->column('role')->type('enum');
  }

  protected function setupShowOperation()
  {
    $this->crud->column('name');
    $this->crud->column('email');
    $this->crud
```

```
      ->column('role')
      ->type('enum');
    $this->crud->column('created_at')->type('datetime');
    $this->crud->column('updated_at')->type('datetime');
  }

  protected function setupCreateOperation()
  {
    $this->crud->setValidation(UserRequest::class);

    $this->crud->field('name');
    $this->crud->field('email');
    $this->crud->field('password');

    $this->crud
      ->field('role')
      ->type('enum');
  }
  // ...
}
```

Bonus: Fixing Update/Create Functionality

By default, Backpack requires that you set a pre-hashed password in the "Password" field in order to create or update a user. This is not desirable behavior, so let's fix that.

First, let's open `app/Http/Requests/UserRequests.php` and make passwords optional when updating a user:

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
use Laravel\Fortify\Rules\Password;
```

```
class UserRequest extends FormRequest
{
  // ...

  public function rules()
  {
    return [
      'password' => [
        $this->method() === 'POST' ? 'required' : 'optional',
        'string',
        new Password(),
      ],
    ];
  }
  // ...
}
```

Next, we will update `app/Http/Controllers/Admin/UserCrudController.php` to hash passwords before storing them, and to omit them from the update if they haven't been provided:

```
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Requests\UserRequest;
use Backpack\CRUD\app\Http\Controllers\CrudController;
use Illuminate\Support\Facades\Hash;

class UserCrudController extends CrudController
{
  // ...
  use \Backpack\CRUD\app\Http\Controllers\Operations\CreateOperation {
    store as traitStore;
  }

  use \Backpack\CRUD\app\Http\Controllers\Operations\UpdateOperation {
    update as traitUpdate;
  }

  // ...
```

```
public function store()
{
  $this->crud->hasAccessOrFail('create');
  $this->crud->setRequest($this->crud->validateRequest());

  $request = $this->crud->getRequest();

  // Encrypt password if specified.
  if ($request->input('password')) {
    $request->request->set(
      'password',
      Hash::make($request->input('password'))
    );
  } else {
    $request->request->remove('password');
  }

  $this->crud->setRequest($request);
  $this->crud->unsetValidation(); // Validation has already been run

  return $this->traitStore();
}

public function update()
{
  $this->crud->hasAccessOrFail('update');
  $this->crud->setRequest($this->crud->validateRequest());

  $request = $this->crud->getRequest();

  // Encrypt password if specified.
  if ($request->input('password')) {
    $request->request->set(
      'password',
      Hash::make($request->input('password'))
    );
  } else {
    $request->request->remove('password');
  }
```

```
    $this->crud->setRequest($request);
    $this->crud->unsetValidation(); // Validation has already been run

    return $this->traitUpdate();
  }
}
```

⚠ Note

In order for a CRUD form to work, all fields in the form must be defined in the `$fillable` property on your model.

Revision #15
Created 13 November 2022 17:36:09
Updated 9 December 2022 15:35:11