<https://spatie.be/docs/laravel-backup/v9/installation-and-setup>

composer require spatie/laravel-backup

php artisan vendor:publish --provider="Spatie\Backup\BackupServiceProvider" --tag=backup-config

config/backup.php (alter)

    'backup' => [

        /\*

         \* The name of this application. You can use this name to monitor

         \* the backups.

         \*/

        'name' => 'app',

        'source' => [

            'files' => [

                /\*

                 \* The list of directories and files that will be included in the backup.

                 \*/

                'include' => [

                    base\_path(),

                ],

….………………………………..

config/backup.php (alter)

      /\*

             \* The disk names on which the backups will be stored.

             \*/

            'disks' => [

                'public\_backup',

            ],

config/filesystem.php (add)

        'public\_backup' => [

            'driver' => 'local',

            'root' => public\_path('backup'),

            'url' => env('APP\_URL') . '/backup',

            'visibility' => 'public',

        ],

In Controller:

<?php

namespace App\Http\Controllers;

use App\Enum\MasterPermissionEnum;

use App\Events\ResetPasswordConfirmationEvent;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Artisan;

use Illuminate\Support\Facades\Storage;

class BackupController extends Controller

{

    public function generate()

    {

        $this->authorize(MasterPermissionEnum::BACKUP\_GENERATE->value);

        dispatch(function () {

            Artisan::call('backup:run', ['--only-db' => true]);

        });

        event(new ResetPasswordConfirmationEvent);

    }

    public function index()

    {

        $this->authorize(MasterPermissionEnum::BACKUP\_VIEW->value);

        // Just use the relative path under the disk root

        $files = Storage::disk('public\_backup')->files('app');

        // Return mapped backup info

        $backups = collect($files)->map(function ($file) {

            return [

                'path' => $file,

                'url' => Storage::disk('public\_backup')->url($file),

                'size' => $this->formatBytes(Storage::disk('public\_backup')->size($file)),

                'created\_at' => date('Y-m-d H:i:s', Storage::disk('public\_backup')->lastModified($file)),

            ];

        });

        return inertia('backup/index', [

            'backups' => $backups,

        ]);

    }

    public function destroy(Request $request)

    {

        $this->authorize(MasterPermissionEnum::BACKUP\_DELETE->value);

        $disk = Storage::disk('public\_backup');

        // Path is URL-encoded when sent from frontend

        $decodedPath = urldecode($request->path);

        if ($disk->exists($decodedPath)) {

            $disk->delete($decodedPath);

        }

        event(new ResetPasswordConfirmationEvent);

    }

    protected function formatBytes($bytes, $precision = 2)

    {

        $units = ['B', 'KB', 'MB', 'GB', 'TB'];

        $bytes = max($bytes, 0);

        $pow = floor(($bytes ? log($bytes) : 0) / log(1024));

        $pow = min($pow, count($units) - 1);

        $bytes /= pow(1024, $pow);

        return round($bytes, $precision) . ' ' . $units[$pow];

    }

}