RAPPORT DE STAGE (21/06 AU 25/06)

SOMMAIRE

PREAMBULE

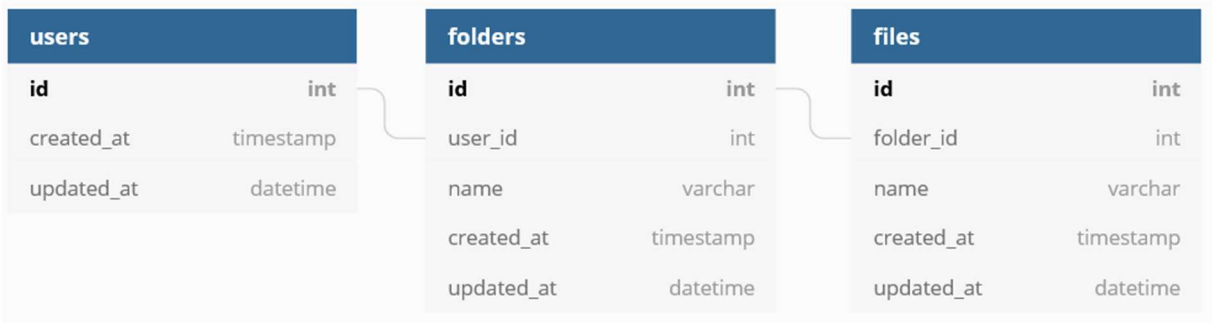
Avant toute chose, il est nécessaire d'indiquer que, pour des raisons de confidentialité, certains travaux ne pourront pas être illustrés (indiqués par un *) ou expliqués dans le détail.

Au sein de ce rapport de stage sont mentionnés des documents PDF, documents qui sont présents dans l'archive.

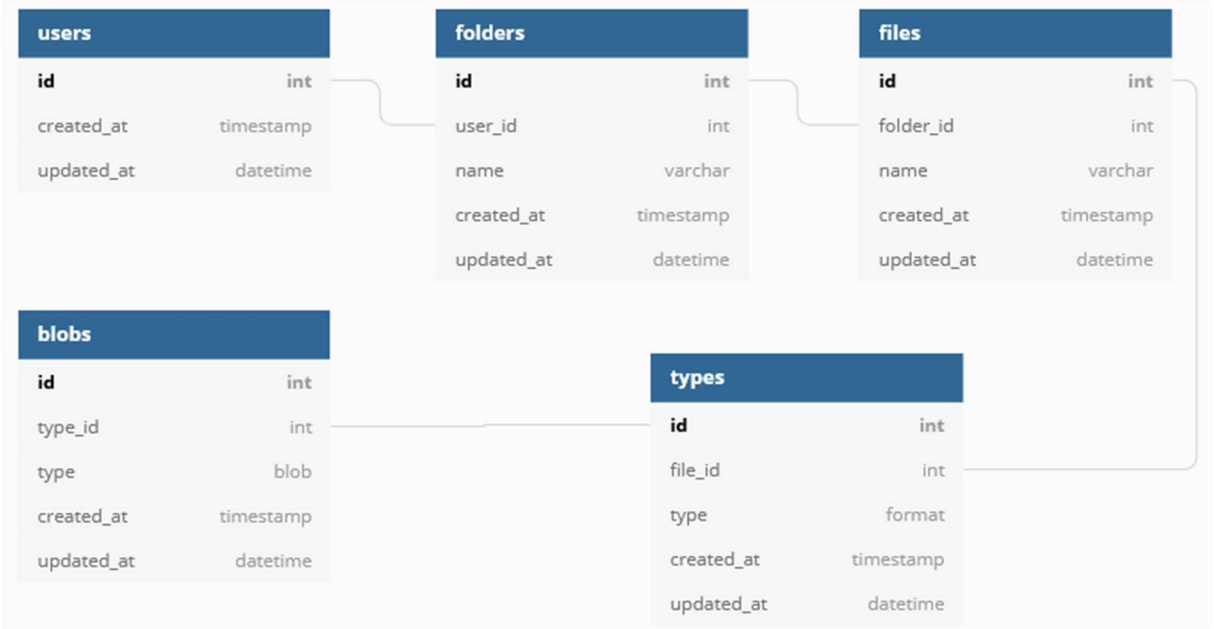
LUNDI 21 JUIN 2021

Activité de la journée : Pour commencer, lors de la réunion du matin, j'ai présenté la conception de la base de données réalisé avec mon binôme à travers l'évolution de différents schémas (faits à l'aide de dbdiagram.io[1]). Pour rappel, cette base de données doit être utilisée pour un Drive et nous voulons y stocker des blobs.
Le raisonnement derrière cette évolution est le suivant :

Au départ, il est nécessaire d'avoir une table utilisateurs (users), un table dossiers (folders) que pourront avoir chaque utilisateur puis une table fichiers (files) pour les fichiers présents dans les dossiers.



Ensuite à cela doit s'ajouter une table blobs qui contiendra le contenu des fichiers. Mais un blob peut très bien être une image, une vidéo ou encore un fichier texte lourd. Une table types est donc nécessaire pour cela.



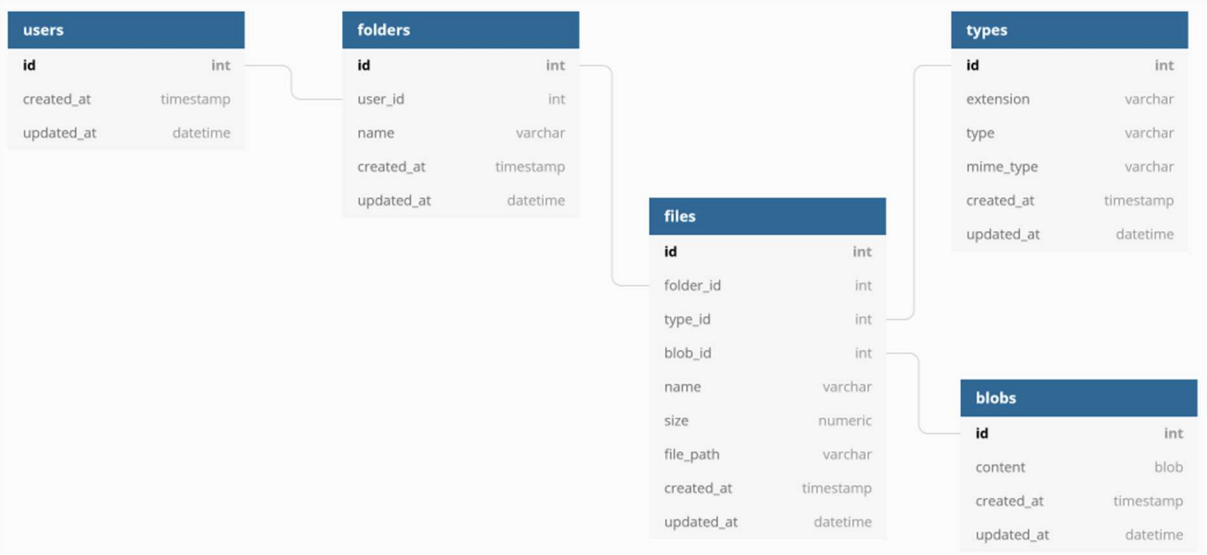
Après concertation avec le maître de stage, il a fallu revoir les relations entre les tables.

[1] https://dbdiagram.io

Les changements se sont effectués au niveau de la table files en particulier qui a été directement reliée à la table types et blobs ce qui a donné lieu à une contrainte d'intégrité référentielle.

Suite à cela, un autre point dans la journée a été fait avec le maître de stage. C'est à ce moment-là qu'il nous a dit que le schéma manquait encore beaucoup d'informations. Il a notamment parlé des métadonnées en insistant sur l'importance de l'extension du fichier et le type MIME (Multipurpose Internet Mail Extensions) et du fait qu'aucune colonne n'existait dans la table blobs pour stocker le contenu du fichier.

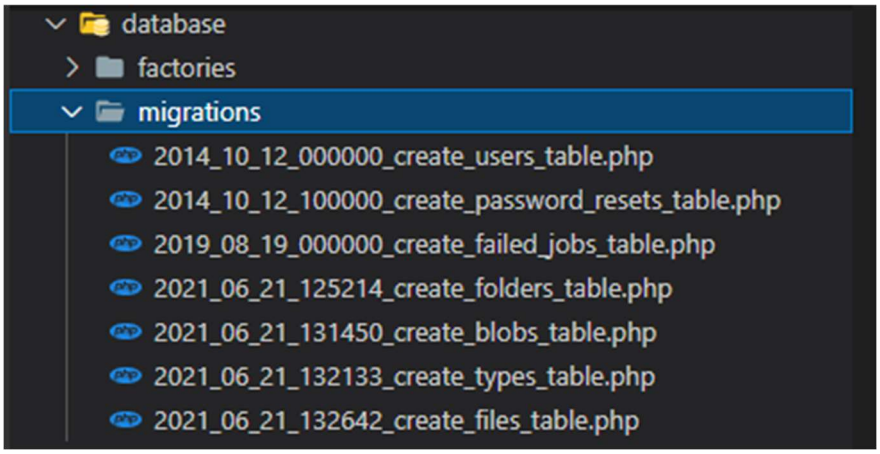
Après quelques ajustements au niveau de la table files (ajout de métadonnées comme la taille du fichier et le chemin), types (ajout de l'extension et du type MIME) et blobs (ajout d'une colonne pour le contenu du fichier) le schéma ressemblait à cela :



MARDI 22 JUIN 2021

Activité de la journée : Suite à la conception du schéma de notre base de données, le maître de stage lors de la réunion nous a demandé de mettre en place la base de données.

Avant toute chose il a fallu commencer par les migrations (toujours à l'aide de Visual Studio Code et du framework Laravel).
A l'aide de la commande « php artisan make:model NomDeLaTable -m » nous avons généré les fichiers des migrations associés à leurs modèles respectifs.



database > migrations > 2014_10_12_000000_create_users_table.php

```php
<?php

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

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->timestamp('email_verified_at')->nullable();
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

database > migrations > 2021_06_21_125214_create_folders_table.php

```php
<?php

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

class CreateFoldersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('folders', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained();
            $table->string('name');
            $table->text('description', 100);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('folders');
    }
}
```

database > migrations > 2021_06_21_131450_create_blobs_table.php

```php
<?php

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

class CreateBlobsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('blobs', function (Blueprint $table) {
            $table->id();
            $table->binary('content');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('blobs');
    }
}
```

database > migrations > 2021_06_21_132133_create_types_table.php

```php
<?php

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

class CreateTypesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('types', function (Blueprint $table) {
            $table->id();
            $table->string('extension')->unique();
            $table->string('type');
            $table->string('mime_type');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('types');
    }
}
```

database > migrations > 2021_06_21_132642_create_files_table.php

```php
<?php

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

class CreateFilesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('files', function (Blueprint $table) {
            $table->id();
            $table->foreignId('folder_id')->constrained();
            $table->foreignId('type_id')->constrained();
            $table->foreignId('blob_id')->constrained();
            $table->string('name')->nullable();
            $table->string('size')->nullable();
            $table->string('file_path')->nullable();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('files');
    }
}
```

Une fois les migrations effectuées, j'ai effectué des recherches concernant l'upload de fichier avec Laravel. Je suis tombé sur un tutoriel [2] qui m'a permis mettre en place une fonction d'upload basique.

Le code du contrôleur :

app > Http > Controllers > FileUpload.php

```php
<?php

namespace App\Http\Controllers;

use App\Models\Blob;
use App\Models\File;
use Illuminate\Http\Request;

class FileUpload extends Controller
{
  public function createForm(){
    return view('file-upload');
  }

  public function fileUpload(Request $req){

        $req->validate([
        'file' => 'required|mimes:csv,txt,xlx,xls,pdf,jpg,png|max:2048'
        ]);

        $fileModel = new File;

        if($req->file()) {
            $fileName = time().'_'.$req->file->getClientOriginalName();
            $filePath = $req->file('file')->storeAs('uploads', $fileName, 'public');

            $fileModel->name = time().'_'.$req->file->getClientOriginalName();
            $fileModel->file_path = '/storage/' . $filePath;
            $fileModel->save();

            return back()
            ->with('success','File has been uploaded.')
            ->with('file', $fileName);
        }
   }
}
```

Les routes :

```php
Route::get('/upload-file', [FileUpload::class, 'createForm']);

Route::post('/upload-file', [FileUpload::class, 'fileUpload'])->name('fileUpload');

```

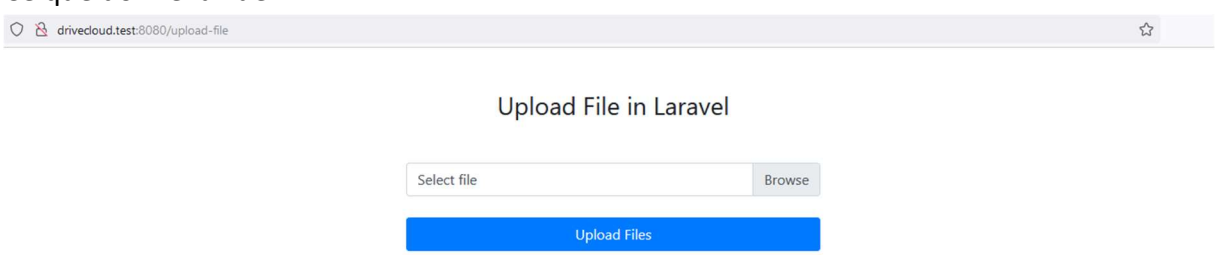
[2] https://www.positronx.io/laravel-file-upload-with-validation/

Le code de la vue :

resources > views > file-upload.blade.php

```
<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">

    <title>Laravel File Upload</title>
    <style>
        .container {
            max-width: 500px;
        }
        dl, ol, ul {
            margin: 0;
            padding: 0;
            list-style: none;
        }
    </style>
</head>

<body>

    <div class="container mt-5">
        <form action="{{route('fileUpload')}}" method="POST" enctype="multipart/form-data">
          <h3 class="text-center mb-5">Upload File in Laravel</h3>
            @csrf
            @if ($message = Session::get('success'))
            <div class="alert alert-success">
                <strong>{{ $message }}</strong>
            </div>
          @endif

          @if (count($errors) > 0)
            <div class="alert alert-danger">
                <ul>
                    @foreach ($errors->all() as $error)
                      <li>{{ $error }}</li>
                    @endforeach
                </ul>
            </div>
          @endif

            <div class="custom-file">
                <input type="file" name="file" class="custom-file-input" id="chooseFile">
                <label class="custom-file-label" for="chooseFile">Select file</label>
            </div>

            <button type="submit" name="submit" class="btn btn-primary btn-block mt-4">
                Upload Files
            </button>
        </form>
    </div>

</body>
</html>
```

Ce que donne la vue :



MERCREDI 23 JUIN 2021

Activité de la journée : Lors de la réunion il a été mis en évidence que la mise en place de cette upload de fichier n'était pas exactement ce que l'on recherchait. Il y avait la problématique d'un fichier qui serait trop lourd à stocker par rapport à la capacité maximale que peut supporter un blob. La tâche qui m'a été confiée à l'issue de cette réunion était alors de trouver une solution pour pouvoir soit :

1- Trouver un type de donnée qui a une capacité de stockage encore plus grande que le blob
2- Recherchez et mettre en place une solution qui permettrait de scinder le fichier en différents « morceaux » (ou fragments) en plusieurs tables avec un référencement par id. (exemple : Un blob de 3 MB serait scindé en 6 fragments de 500 KB avec 1 fragments dans une table à part)

Comme point de départ j'ai commencé à consulter et rassembler des informations concernant les blobs et leur capacité. Cela m'a permis de rédiger une petite documentation :

Recherches_BLOB_Decoupage.pdf

JEUDI 24 JUIN 2021

Activité de la journée : Pendant la réunion du matin j'ai envoyé ma documentation au maître de stage qui m'a dit qu'il la regarderait dans la journée et qu'il ferait le point avec moi.

En attendant sa réponse j'ai poussé mes recherches plus loin en me posant la question de l'organisation du schéma de la base de données au niveau de cette fragmentation du blob. Je me suis dit qu'il avait différentes façons de procéder et que l'une d'entre elles devait être la plus adaptée à la situation.
J'ai donc rédigé un document montrant les différentes façons de procéder :

Conceptions_fragmentation_BLOB.pdf

Voulant tester la méthode que j'estimais la plus optimale (la façon n°3), j'ai donc modifié mon code au niveau de ma migration de ma table blobs pour y ajouter la colonne fragments :

database > migrations > 2021_06_21_131450_create_blobs_table.php

```php
<?php

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

class CreateBlobsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('blobs', function (Blueprint $table) {
            $table->id();
            $table->binary('content');
            $table->foreignId('fragment')->nullable()->constrained('blobs');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('blobs');
    }
}
```

J'ai aussi adapté le code au niveau du contrôleur pour scinder le fichier à upload dans la base de données :

app > Http > Controllers > FileUpload.php

```php

class FileUpload extends Controller
{
  public function createForm(){
    return view('file-upload');
  }

  public function fileUpload(Request $req){

        $req->validate([
        'file' => 'required|mimes:csv,txt,xlx,xls,pdf,jpg,png|max:2048'
        ]);

        if (empty($req->file('file')->getRealPath())) {
          return back()->with('success','No file selected');
        }
        $path = $req->file('file')->getRealPath();
        $size = filesize($path);
        $content = file_get_contents($path);

        if ($size < 500)
        {
        $blob = new Blob();
        $base64 = base64_encode($content);
        $blob->content = $base64;
        $blob->save();
        return response('Success!');
        }
        else
        {
        //$chunkContent = chunk_split($content, 500, ' ');
        $tabContent = str_split ($content, 500000);
        $i =0;
        foreach($tabContent as $chunk){
          $blob = new Blob();
          $base64 = base64_encode($chunk);
          $blob -> content = $base64;
          if ($i != 0) {
            $blob -> fragment = $latestId;
          }
          $blob->save();
          $latestId = $blob->id;
          $i++;

        }
        return response('Success!');
        }
   }
}
```

VENDREDI 25 JUIN 2021

Activité de la journée : Le maître de stage m'a fait un retour positif sur mon code et la façon de procéder. Désormais il ne reste plus qu'à optimiser le code ainsi que trouver le moyen d'upload en même temps les métadonnées du fichier dans les tables files et types.

NOTE DE FIN :

Pour conclure, les différents travaux de cette semaine ont été à mon sens plus ardus du fait qu'ils laissaient place à beaucoup de liberté sans qu'il y ait forcément de mauvais choix. J'ai dû explorer différentes pistes dont beaucoup pouvaient être considérées comme valide (c'est pourquoi mon binôme à partir de mercredi s'est vu assigné une autre tâche que la mienne). Cela m'a permis d'approfondir mes connaissances concernant le BLOB et la conception d'une base de données.

Enfin cela m'a permis de devenir plus autonome même si j'ai du parfois demander des conseils au maître de stage ou bien à Valentine lorsque j'étais vraiment bloqué.