CRÉATION D'UN OUTIL DE SONDAGE EN LIGNE

Auteur : Robustiano LOMBARDO
Chef de projet : Karim BOURAHLA
Experts : M. MALHERBE et M. BERNEY
Version : 3.2
Date de dernière mise à jour : 08.05.2024

Table des matières

1. Presentation du projet 3
2. Analyse / Conception 4
2.1 Base de données 4
2.2 Stratégie de test 5
3. Implémentation 5
3.1 Mise en place de l'environnement 5
3.2 Model 6
3.3 Faker/Seeder 7
3.4 CRUD pour les sondages 7
3.5 Description des tests effectués 10
3.6 Problèmes rencontrer 10
4. Conclusion 10
5. Annexes 10

1. Presentation du projet

Le projet consiste en la programmation d'une application de sondage en ligne en utilisant Laravel et le Kit Jetstream (Inertia.js/Vue.js/Tailwind CSS). Le projet doit donc aussi prendre en compte la conception et l'implémentation d'une base de données MySQL.

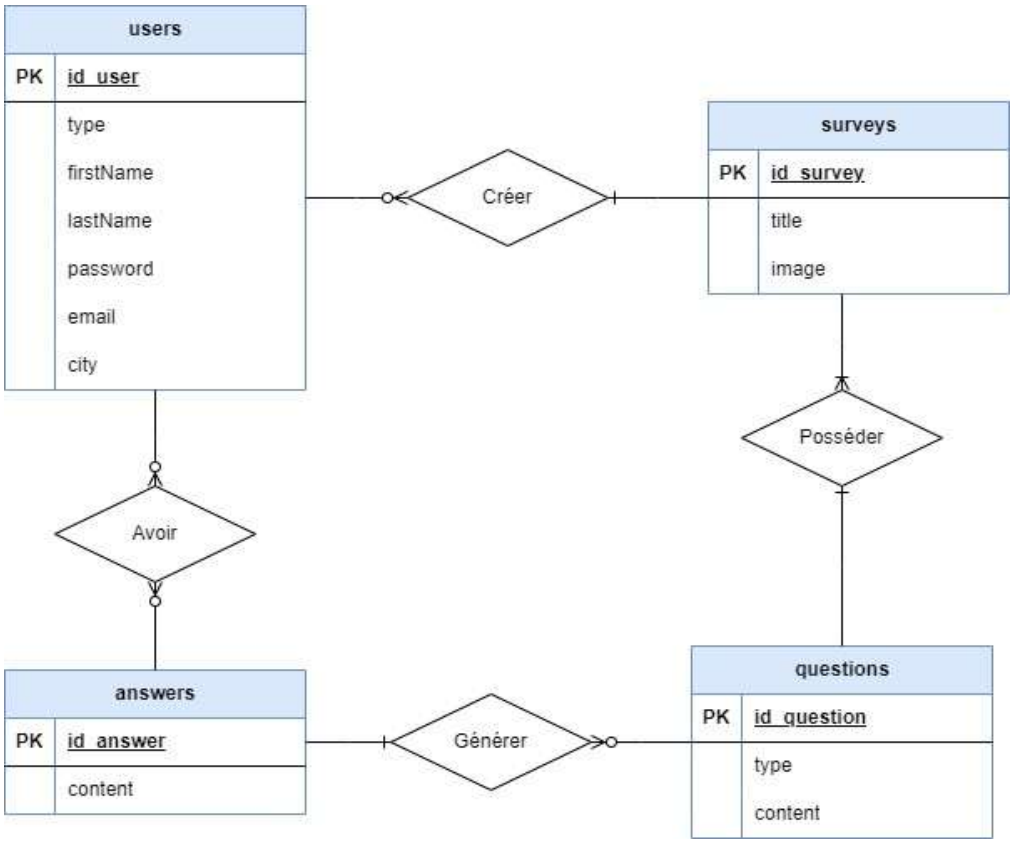
L'application doit inclure des tests avec Pest, une gestion des rôles, des exports vers des fichiers Excel et PDF, la génération et l'affichage d'un graphique et finalement pouvoir appliquer les fonctionnalités CRUD sur tous les modèles.

L'identité visuelle étant secondaire, l'application est donc basé sur les différents rendus disponibles sur le site web https://tailblocks.cc/.

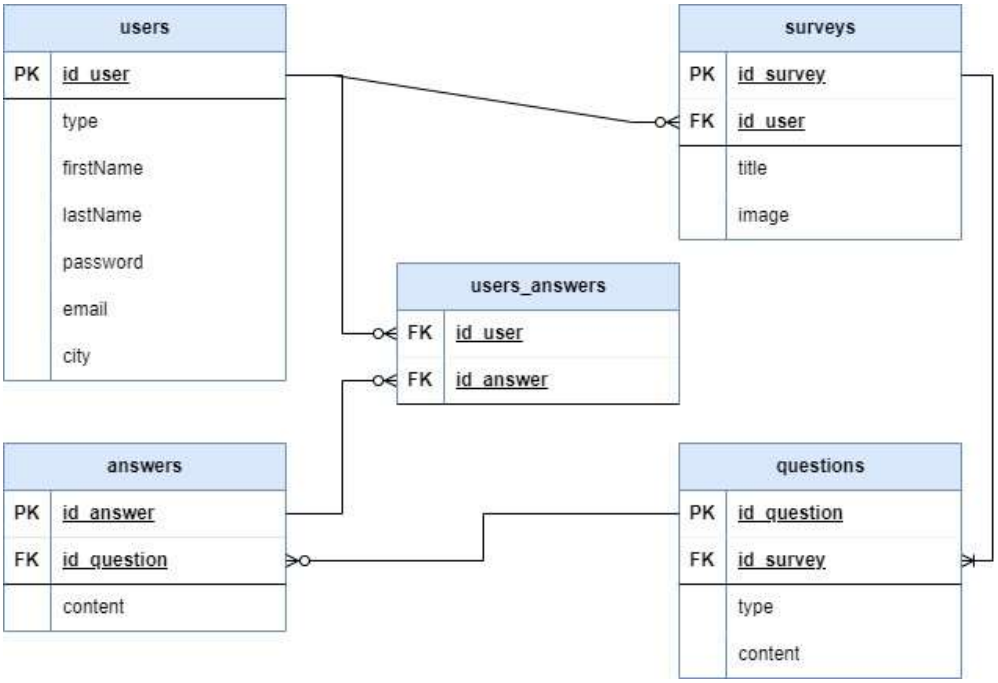
2. Analyse / Conception

2.1 Base de données

MCD :



MLD :



2.2 Stratégie de test

Les fonctionnalités de l'application sont tester grâce à Pest, celui-ci est un framework de tests unitaires. Son utilisation a été demandé par le client pour la réalisation des tests. Les tests unitaires servent à vérifier les fonctionnalités CRUD, la génération des PDFs, la génération des fichiers d'export Excel, la création du graphique et la gestion des droits d'accès pour les différents utilisateurs. L'application s'occupent de générer de fausse données pour que les tests puissent s'exécuter sans problème. Les tests couvrent environ 90% des fonctionnalités, par exemple les tests unitaires ne peuvent pas vérifier le bon affichage du graphique, mais plutôt uniquement les données que celui-ci affiche.

3. Implémentation

3.1 Mise en place de l'environnement

Pour commencer, le projet nécessite d'avoir une version de PHP supérieur ou égal à la 8.2, la dernière version de Composer, Node.js et Xampp installé.

Pour la création du projet, exécuter la commande de création d'un nouveau projet Laravel qui est `laravel new survey` ensuite, voici les paramètres à choisir pour mettre en place l'environnement :

```
Would you like to install a starter kit? [No starter kit]:
 [none     ] No starter kit
 [breeze   ] Laravel Breeze
 [jetstream] Laravel Jetstream
> jetstream


Which Jetstream stack would you like to install? [Livewire]:
 [livewire] Livewire
 [inertia ] Vue with Inertia
> inertia


Would you like any optional features? [None]:
 [none        ] None
 [api         ] API support
 [dark        ] Dark mode
 [verification] Email verification
 [teams       ] Team support
 [ssr         ] Inertia SSR
> none


Which testing framework do you prefer? [Pest]:
 [0] Pest
 [1] PHPUnit
> 0
```

Lorsque de la création du projet, installé dans celui-ci les 4 packages suivants : pour la gestion `composer require spatie/laravel-permission` des rôles et des permissions, pour la `npm install chart.js` génération de graphique, pour la génération de fichier PDF utiliser `composer require barryvdh/laravel-dompdf` et pour finir qui permet `composer require maatwebsite/excel` l'exportation et l'importation de données via un fichier Excel.

Pour faciliter le développement sous cette environnement voici des extensions utiles sur Visual Studio code :

L'environnement est donc désormais propice au bon développement de cette application Laravel avec un kit Jetstream.

3.2 Model

La création des modèles s'effectue grâce à la commande `php artisan make:model Survey -a` le « -a » sers à créer tous les éléments en rapport avec le modèle comme le controller, le seeder et encore d'autres fichiers. La création et la configuration de ces éléments est donc obligatoire pour que des données soient retourner de la base de données.

Jetsteam ayant déjà un système de gestion des utilisateurs, la modification du fichier model et du fichier de migration de la table users pour appliquer les modifications apportées à cette table.

```php
protected $fillable = [
    'firstName',
    'lastName',
    'email',
    'password',
    'city',
];
```

```php
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('firstName');
    $table->string('lastName');
    $table->string('city')->nullable();
    $table->string('email')->unique()->nullable();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password')->nullable();
    $table->rememberToken();
    $table->foreignId('current_team_id')->nullable();
    $table->string('profile_photo_path', 2048)->nullable();
    $table->timestamps();
});
```

La modification des fichiers Vue ou PHP contenant des références au champ « name » est obligatoire, celui-ci n'existant plus la valeur de ce champ retourne une erreur.

3.3 Faker/Seeder

Les fakers et seeders sont utiliser pour la génération de données de tests. Ces fichiers ont été générer au préalable grâce à la commande de création des modèles. Les fichiers fakers servent de modèle pour la génération des données. La création des valeurs de chaque élément de la table choisi ce fait ici, voici un exemple :

```php
return [
    'title' => $this->faker->unique()->lexify('SurTitle-???'),
    'image' => $this->faker->unique()->lexify('SurImage-???'),
    'id_user' => User::inRandomOrder()->first(),
];
```

Pour les fichiers seeders appelle uniquement la classe correspondante à son faker, utilisé `ClassDeVotreChoix::factory(5)->create();` dans le seeder pour générer le nombre de données de prédéfini. Et dans le fichier « DatabaSeeder.php » appelé les tous les autres seeders, comme ceux-ci :

```php
$this->call([
    UserSeeder::class,
    SurveySeeder::class,
    QuestionSeeder::class,
    AnswerSeeder::class,
    UserAnswerSeeder::class,
]);
```

3.4 CRUD pour les sondages

Pour illustrer l'implémentation des fonctionnalités CRUD, celles de la table **surveys** semble être la plus approprié vu le niveau de complexité de celle-ci. Car lorsqu'une modification à lieu sur un de ces éléments, les éléments de la table **questions** et **answers**.

Comme citées précédemment, la fonctionnalité **CREATION** va devoir en plus de créer un élément **surveys**, créer des éléments **questions** et **answers** en lien avec celui-ci. Donc le controller et l'affichage VueJs doivent aussi permettre la création de ces 2 éléments. Dans l'affichage, la création de variable est nécessaire pour récupérer données entrée par l'utilisateur. Toutes les variables ne sont pas ensuite envoyées au controller, certaines sont juste présentes pour rendre le formulaire dynamique.

```javascript
const numberQuestion = ref();

const numberAnswer = reactive([]);

const contentQuestion = reactive([]);

const form = reactive({
    title: null,
    image: null,
    contentQuestion : contentQuestion,
    contentAnswer : [contentAnswer],
    type : typeOfQuestion,
});
```

Par exemple, la variable **numberQuestion** va générer des champs de formulaires supplémentaire permettant la création de plus de question. Alors que **form** va servir à récupérer tous les champs du formulaire pour les transmettre au controller.

Dans le controller, pour la création de ces champs voici comment procéder. Vu que le **survey** doit contenir une image, le stockage de celle-ci dans l'application et ensuite transmettre le chemin et le nom de l'image à la DB pour qu'il soit stocker.

```
if($request->hasFile('image')){
    $path = Storage::disk('public')->putFile('image', new File($request->file('image')));
}
```

Ensuite pour la création des questions et des réponses, voici comment procéder :

```
for($i=1; $i < count($request->contentQuestion);$i++) {
    $question = Question::create([
        'content' => $request->contentQuestion[$i],
        'type' => $request->type[$i],
        'id_survey' => $survey->id,
    ]);
    if(isset($request->contentAnswer[0][$i]))
    for($x=1;$x < count($request->contentAnswer[0][$i]);$x++){
        $answers = Answer::create([
            'content' => $request->contentAnswer[0][$i][$x],
            'id_question' => $question->id,
        ]);
    }
}
```

La boucle va vérifier combien de question l'utilisateur a créée et ensuite les créer dans la DB. Pour les réponses, l'utilisation d'une boucle imbriquée semble être la manière la plus simple pour la création de celle-ci. Cette boucle permet de récupérer directement l'**id** de la question qui vient d'être créée et le transmettre immédiatement à toutes les questions qui lui sont liées.

La fonctionnalité **READ** est la plus simple, celle-ci doit juste retourner le sondage choisi avec les questions et les réponses qui lui son liée. Donc voici à quoi ressemble le code dans le controller :

```
public function show($id)
{
    $survey = Survey::where('id', $id)->get();
    $survey['user'] = User::where('id',$survey[0]->id_user)->get();
    $survey['questions'] = Question::where('id_survey',$id)->get();
    foreach($survey['questions'] as $question){
        $question['answer'] = Answer::where('id_question', $question->id)->get();}
    return Inertia::render('Survey/Survey', ['survey' => $survey]);
```

La fonctionnalité **UPDATE** est la plus complexe, pourquoi est-ce la plus complexe ? Car en premier point, le formulaire doit avoir les champs pré-remplis avec les champs déjà existant dans la base de données et sur un formulaire dynamique ce n'est pas chose facile.

L'utilisation de **onBeforeMount** fourni par VueJs semble être parfaite, car celle-ci permet de définir des valeurs à certaines variables avant même que l'affichage soit chargé. Par exemple voici comment est prédéfini le titre du formulaire :

```
onBeforeMount(() => {
  if(props.survey && props.survey[0].id) {
    form.title = props.survey[0].title;
    numbers.forEach(number => {
        if(number == props.survey.questions.length){
            numberQuestion.number = number;
        }
```

Ensuite après avoir récupérer toutes les valeurs du formulaire (le fonctionnement est le même que pour la fonctionnalité **CREATE**). Le controller doit s'occuper des différentes modifications. Appliquer les modifications aux champs se retrouvant directement dans le formulaire telle que le titre et l'image.

```
$survey = Survey::find($id);
$survey->title = $request->title;
if($request->hasFile('image')){
    $path = Storage::disk('public')->putFile('image', new File($request->file('image')));
    if($survey->image != null)
        Storage::disk('public')->delete($survey->image);
    $survey->image = $path;
}
$survey->update();
```

Ensuite, le controller doit vérifier si les questions que le front vient de lui transmettre existe déjà pour effectuer une modification/suppression/création doit être effectuer pour les questions. La même opération est effectuée pour les réponses.

```
$questions = Question::where('id_survey',$id)->get();
foreach($questions as $question){
    $exist = false;
    for($i = 1; $i < count($request->question['id']); $i++){
        if($request->question['id'][$i] == $question->id){
            $exist = true;
            $question->content = $request->question['content'][$i];
            $question->type = $request->question['type'][$i];
            $question->update();
        }
    }
    if(!$exist){
        $question->delete();
    }
```

La fonctionnalité **DELETE** peut être abordé de plusieurs façon, soit en supprimant tous les éléments 1 par 1 manuellement dans le controller, soit en ayant paramétrer une suppression des relations en cascade. La meilleure option dans cette situation semble être la 2ème.

Cette opération doit être effectuer lors de la création de la DB. Donc dans les fichiers de migration qui vont être impacté par la suppression. Dans ce cas se sont les fichiers **questions_tables.php**, **answers_tables.php** et **user_answers_tables.php**. La ligne permettant relation grâce à une **FK** doit ressembler à ceci :

```
$table->foreignId('id_survey')->constrained(table: 'surveys')->onUpdate('cascade')->onDelete('cascade');
```

Le premier paramètre est le nom du champ, le deuxième est le nom de table pour effectuer la relation. Ensuite ce sont les actions à effectuer en cas de mise à jour ou de suppression de l'éléments en lien.

Finalement, dans la fonction **destroy** ajouter ceci pour supprimer le sondage et l'image qui lui est liée dans l'application.

```
$survey = Survey::find($id);
if($survey->image != null){
    Storage::disk('public')->delete($survey->image);
}
$survey->delete();
```

3.5 Description des tests effectués

On sait depuis longtemps que travailler avec du texte lisible et contenant du sens est source de distractions, et empêche de se concentrer sur la mise en page elle-même. L'avantage du Lorem Ipsum sur un texte générique comme 'Du texte. Du texte. Du texte.' est qu'il possède une distribution de lettres plus ou moins normale,

3.6 Problèmes rencontrer

On sait depuis longtemps que travailler avec du texte lisible et contenant du sens est source de distractions, et empêche de se concentrer sur la mise en page elle-même. L'avantage du Lorem Ipsum sur un texte générique comme 'Du texte. Du texte. Du texte.' est qu'il possède une distribution de lettres plus ou moins normale,

4. Conclusion

5. Annexes

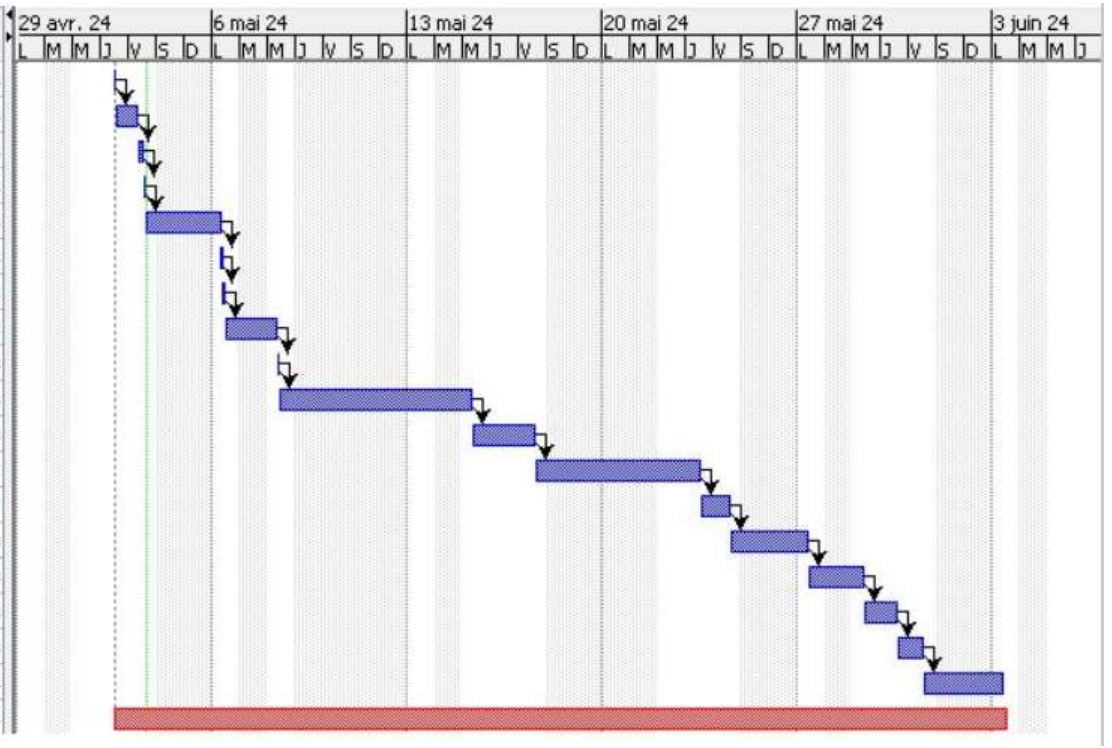
5.1. Résumé du rapport du TPI

On sait depuis longtemps que travailler avec du texte lisible et contenant du sens est source de distractions, et empêche de se concentrer sur la mise en page elle-même. L'avantage du Lorem Ipsum sur un texte générique comme 'Du texte. Du texte. Du texte.' est qu'il possède une distribution de lettres plus ou moins normale,

5.2. **Sources**

5.3. Planification Initial

	Nom	Durée
1	Analyse du cahier des charges	0.2 jours
2	Planification	0.4 jours
3	Modèlisation de la base de données	0.4 jours
4	Création du projet Laravel	0.2 jours
5	Création d'un repot Git	0.2 jours
6	Paramètrage des fichiers de configuration	0.1 jours
7	Création des modèles	0.3 jours
8	Création des Seeders/Fakers	0.4 jours
9	Création des Controllers	0.2 jours
10	Mise en place des Tests	1.5 jours
11	Création des fonctionnalités CRUD	1.8 jours
12	Création des affichages	1 jour
13	Mise en place des CRUD sur le front	1 jour
14	Mise en place des rôles et permissions	0.5 jours
15	Gestion des accès	0.5 jours
16	Mise en place du système de génération de fichier PDF	0.7 jours
17	Mise en place du système de génération de fichier Excel	0.8 jours
18	Génération du graphique	0.5 jours
19	Rapport	11 jours



5.4. Journal de Travail

On sait depuis longtemps que travailler avec du texte lisible et contenant du sens est source de distractions, et empêche de se concentrer sur la mise en page elle-même. L'avantage du Lorem Ipsum sur un texte générique comme 'Du texte. Du texte. Du texte.' est qu'il possède une distribution de lettres plus ou moins normale,