

CRÉATION D'UN OUTIL DE SONDAGE EN LIGNE



Auteur : Robustiano LOMBARDO
Chef de projet : Karim BOURAHLA
Experts : M. MALHERBE et M. BERNEY
Version : 6.0
Date de dernière mise à jour : 03.06.2024

Table des matières

1. Presentation du projet	1
2. Analyse / Conception.....	2
2.1 Base de données.....	2
2.2 Stratégie de test.....	3
3. Implémentation	3
3.1 Mise en place de l'environnement	3
3.2 Model	4
3.3 Faker/Seeder.....	5
3.4 CRUD pour les sondages	5
3.5 Graphique.....	9
3.6 Rôles et permissions.....	11
3.7 Export PDF	14
3.8 Export Excel.....	15
3.9 Répondre aux sondages.....	18
3.10 Affichage des réponses	20
3.11 Description des tests effectués.....	21
3.12 Erreurs restantes	23
4. Conclusion	24
4.1 Bilan personnel	24
4.2 Comparaison planification initial et journal de travail.....	24
4.3 Résumé du rapport	25
5. Soucres - Webographie	26
6. Glossaire.....	27
7. Annexes	28

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 basée sur les différents rendus disponibles sur le site web <https://tailblocks.cc/>.

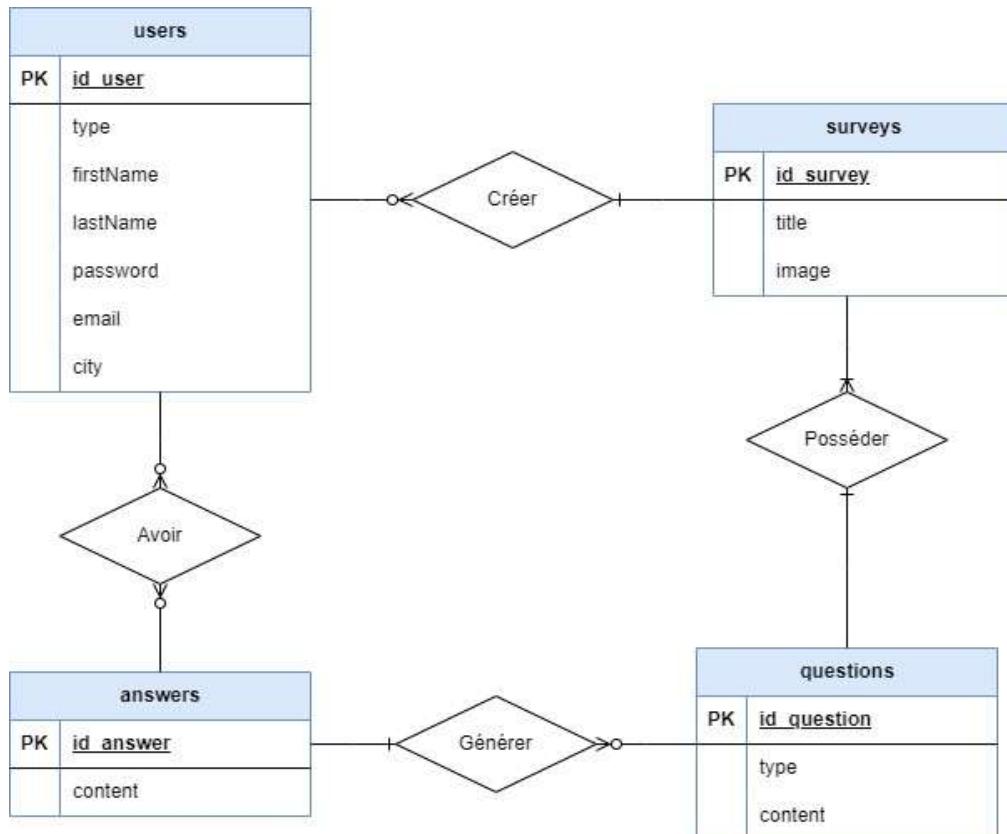
Le client évalue les points suivants :

1. Les liens vers les sondages sont uniques et sécurisés.
2. L'administrateur de la plateforme peut exécuter un CRUD de tous les modèles.
3. L'administrateur peut exporter la liste des créateurs au format Excel avec choix des attributs.
4. L'administrateur peut générer un rapport pdf avec la liste des sondages et choix des attributs.
5. L'implémentation d'une stratégie permettant la sécurisation de la base de données.
6. La mise en place de test PEST.
7. La mise en place de la méthode de suivi des versions avec GitFlow.

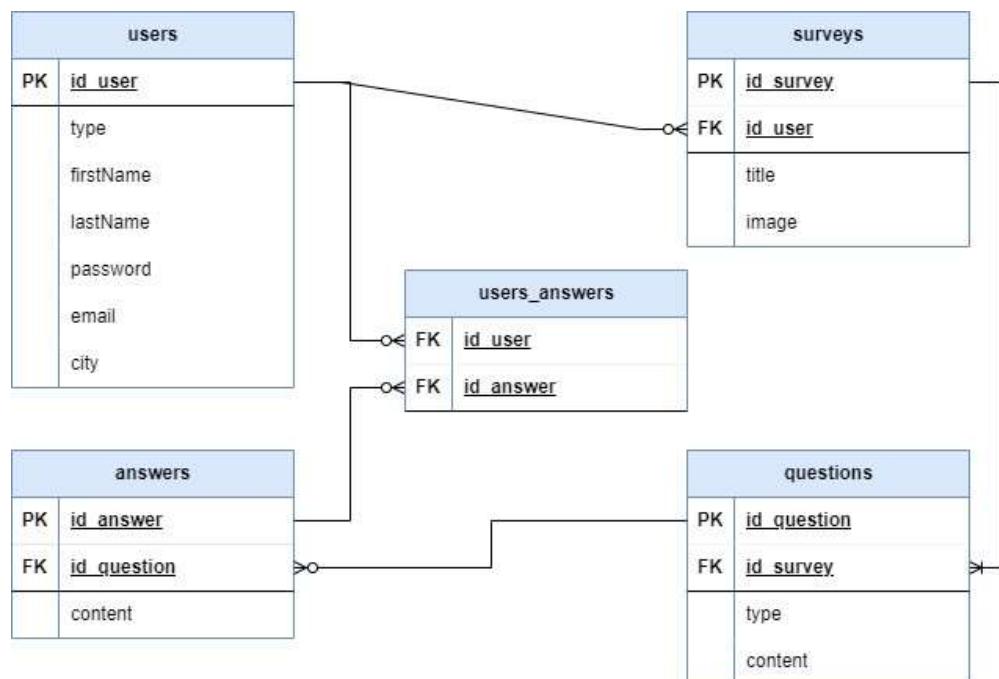
2. Analyse / Conception

2.1 Base de données

MCD :



MLD :



2.2 Stratégie de test

Les fonctionnalités de l'application sont testées grâce à Pest, celui-ci est un framework de tests unitaires. Son utilisation a été demandée par le client pour la réalisation des tests. Les tests unitaires servent à vérifier les fonctionnalités CRUD, la génération des PDF, 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'occupe de générer de fausses 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 supérieure 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 cet 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 » sert 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 retournées de la base de données.

JetStream 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.

```
protected $fillable = [
    'firstName',
    'lastName',
    'email',
    'password',
    'city',
];
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 utilisés pour la génération de données de tests. Ces fichiers ont été générés 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 se fait ici, voici un exemple :

```
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 prédéfini. Et dans le fichier « **DatabaseSeeder.php** » appelle tous les autres seeders, comme ceux-ci :

```
$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, celle de la table **surveys** semble être la plus appropriée vu le niveau de complexité de celle-ci. Car lorsqu'une modification a lieu sur un de ces éléments, les éléments de la table **questions** et **answers**.

Comme citée précédemment, la fonctionnalité **CREATION** doit permettre la création d'élément **surveys**. Dans l'affichage, la création de variables est nécessaire pour récupérer les données entrées par l'utilisateur.

```
const form = reactive({
    title: null,
    image: null,
});
```

Ici, la variable **form** contient un élément pour chaque **input** du formulaire, ajouter en autant que nécessaire, le but et récupérer toutes les données.

Ensuite, le formulaire est envoyé au **controller** en utilisant la méthode **POST**.

```
function submit (){
    router.post('/surveys', form);
};
```

En plus d'une génération standard de données, une image doit être stockée dans l'application et le chemin de celle-ci doit être transmis au **survey** permettant plus tard d'afficher l'image qui lui est attribuée.

```
public function store(StoreSurveyRequest $request)
{
    if($request->hasFile('image')){
        $path = Storage::disk('public')->putFile('image', new File($request->file('image')));
    }
    $survey = Survey::create(array_merge($request->validated(), [
        'title' => $request->title,
        'image' => $path ?? null,
        'id_user' => Auth::id(),
    ]));
    return redirect('/surveys/'.$survey->id);
}
```

Tout d'abord, une vérification est effectuée sur la présence de l'image fournie par l'utilisateur. Ensuite, si celle-ci existe, l'application va la stocker et fournir le chemin d'accès à cette image. Puis une validation des informations envoyées par l'utilisateur est effectuée. Les conditions de validation se trouvent dans le fichier **StoreQuestionRequest.php** et sont gérées avec la fonction **rules**.

```
public function rules(): array
{
    return [
        'title' => ['required', 'min:5', 'max:100'],
        'image' => ['required'],
    ];
}
```

En spécifiant les éléments du formulaire et les conditions que ceux-ci doivent respecter. Si le formulaire passe la validation, le **survey** peut être créé, sinon un message d'erreur est retourné.

Attention, la création d'un lien symbolique est obligatoire pour permettre à l'application d'enregistrer les images fournies par l'utilisateur. Ce lien se crée grâce à la commande `php artisan storage:link` cette commande va donc créer un lien entre le répertoire public et le répertoire où sont stocker les images.

La fonctionnalité **READ** est la plus simple, celle-ci doit juste retourner le sondage choisi avec les questions et les réponses qui lui sont liées. 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** ressemble beaucoup à la fonction **CREATE** du moins pour le frontend. Car le **controller** doit ici récupérer l'élément qui doit être modifié et lui appliquer ces modifications.

Pour le front, 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(() => {
  form.title = props.survey[0].title;
});
```

Un point très important est que les méthodes **PUT** et **PATCH** ne fonctionnent pas pour l'envoi d'image vers le **controller**. Donc, la seule méthode permettant d'envoyer des données restantes est la méthode **POST**.

```
function submit (){
  router.post('/surveys/' +props.survey[0].id, form);
};
```

Ensuite, après avoir récupéré 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 dans le formulaire.

```
public function update(UpdateSurveyRequest $request, $id)
{
  $survey = Survey::find($id);
  $request->validated();
  $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();

  return redirect('/surveys/'.$survey->id);
}
```

Le **controller** va vérifier si l'utilisateur a bien rempli tous les champs en procédant de la même manière que pour la fonctionnalité **CREATE** mais dans le fichier **UpdateSurveyRequest.php**.

```
public function rules(): array
{
  return [
    'title' => ['required', 'min:5', 'max:100'],
  ];
}
```

Ici, uniquement le champ **title** est vérifié car le champ **image** peut se retrouver au pire vide. L'image existante n'est pas supprimée tant que l'utilisateur ne décide pas de mettre une autre image dans le formulaire.

La fonctionnalité **DELETE** peut être abordée de plusieurs façons, soit en supprimant tous les éléments 1 par 1 manuellement dans le **controller**, soit en ayant paramétré une suppression des relations en cascade. La meilleure option dans cette situation semble être la deuxième.

Cette opération doit être effectuée lors de la création de la DB. Donc, dans les fichiers de migration qui vont être impactés par la suppression. Dans ce cas, ce sont les fichiers **questions_tables.php**, **answers_tables.php** et **user_answers_tables.php**. La ligne permettant la 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.

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();
```

De plus d'une suppression de classique, cette fonctionnalité doit avoir une double validation et lorsque la suppression a été effectuée, la position du scroll doit être préservée.

Pour la validation de suppression, en utilisant le kit de démarrage JetStream tous les éléments nécessaires sont déjà fournis. Donc, pour éviter d'effectuer le travail en double, utilisez ces composants. Les composants utiles sont « **ConfirmationModal**, **DangerButton** et **SecondaryButton** ».

```
<ConfirmationModal :show="showConfirmDeleteQuestionModal" @close="closeModal">
  <template #title>
    <h2 class="text-lg font-semibold text-slate-800">Supprimer ce sondage ?</h2>
  </template>
  <template #content>
    <p class="text-lg font-semibold text-slate-600">{{ CurrentElement.title }}</p>
  </template>
  <template #footer>
    <div class="mt-6 flex space-x-4">
      <DangerButton @click="deleteQuestion(CurrentElement.id)">Supprimer</DangerButton>
      <SecondaryButton @click="closeModal">Annuler</SecondaryButton>
    </div>
  </template>
</ConfirmationModal>
<Pagination :meta="questions.meta"/>
```

Pour pouvoir utiliser correctement ces composants, ceux-ci ont besoin de variables ou de fonction à leur déclenchement. Et pour récupérer les informations de l'objet à supprimer, c'est transmis lorsque l'utilisateur appuie sur le bouton de suppression.

```
<button @click="openModal(item)" type="button">
  <Trash></Trash>
</button>
```

Finalement, pour préserver le scroll de la page. Celui-ci s'applique à l'envoi de la requête de suppression sur le front. En plus d'envoyer l'id de l'élément à supprimer, rajouter la variable **preserveScroll** en paramètre.

```
const deleteQuestion = (id) =>{
  router.delete('/questions/' + id, {preserveScroll: true});
  closeModal();
};
```

3.5 Graphique

Pour l'implémentation du graphique, une méthode assez simple pour générer des graphiques de qualité et de manière dynamique est d'utiliser la librairie **Chart.js**. Son installation a déjà été présentée dans **3.1 Mise en place de l'environnement**.

Le graphique s'affiche sur la **Homepage** de l'application. Donc, pour y afficher un graphique, importer sur la page ces éléments :

```
import Chart from 'chart.js/auto';
import { onBeforeMount, onMounted } from 'vue';
```

Le premier permet d'utiliser la classe **Chart** qui sert à la génération et à l'affichage du graphique. **onBeforeMount** permet de charger des données avant la génération de l'affichage et **onMounted** va afficher le graphique.

```
const dataAnswers = {
  labels: [],
  datasets: [
    {
      type: 'bar',
      label: 'Nombre de personne ayant répondu au sondage',
      data: [],
      borderWidth: 1
    }
]
```

Ensuite, la création d'une variable permettant par la suite de transmettre au graphique les données à afficher.

```
onBeforeMount(() => {
  if(props.surveys.data[0] != null){
    props.surveys.data.forEach(element => {
      dataAnswers.labels.push(element.title);
      dataAnswers.datasets[0].data.push(element.nbAnswer);
    });
  }
})
```

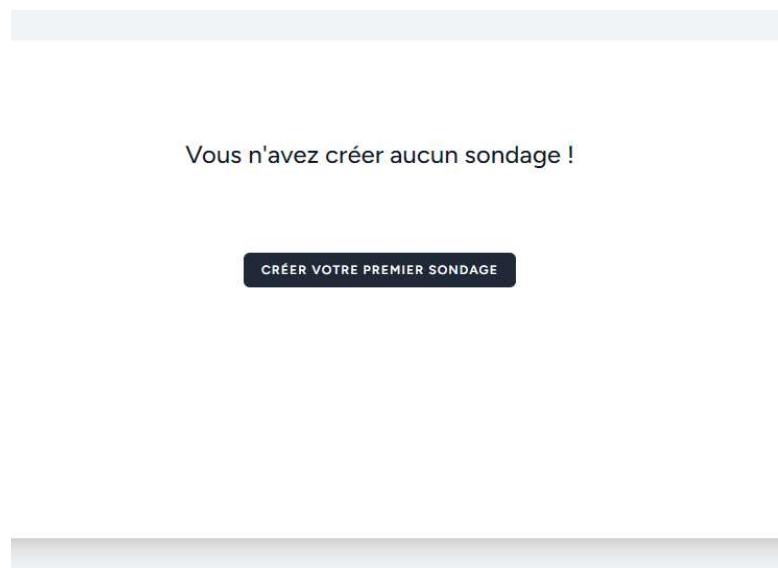
Pour permettre au graphique d'avoir accès aux données, la méthode **onBeforeMount** se charge de transmettre les données à la variable.

```
onMounted(() => {
  if(props.surveys.data[0] != null){
    const ctx = document.getElementById('myChart');
    const myChart = new Chart(ctx, {
      data: dataAnswers,
      options: {
        scales: {
          y: {
            beginAtZero: true
          }
        }
      });
  }
})
```

Ici, le graphique est créé avec les données qui viennent d'être récupérer. Cette méthode récupérées aussi un élément HTML avec grâce à l'id **myChart**. Cet élément doit être de type **canvas** pour pouvoir afficher le graphe.

```
<div v-if="surveys.data[0] == null">
  <h2 class="text-center text-2xl font-medium title-font mb-4 text-gray-900">
    Vous n'avez créer aucun sondage !
  </h2>
  <div class="mt-20 text-center">
    <PrimaryButton @click="createSurvey">Créer votre premier sondage</PrimaryButton>
  </div>
</div>
<canvas id="myChart"></canvas>
```

Voici à quoi ressemble la partie html permettant l'affichage. De plus, si l'utilisateur n'a pas encore créé de sondage, le graphe ne s'affiche pas, mais à la place, du texte lui est affiché. Celui-ci lui dit qu'aucun sondage n'a été créé par l'utilisateur et un bouton est aussi affiché lui proposant de créer son premier sondage.



Jusqu'ici, ce n'était que la partie front. Pour le backend, une information à savoir est qu'une des **goods practices** de l'utilisation de **Chart.js** est la création de son propre **controller**. Donc la création de **ChartController.php** est nécessaire. A l'intérieur de celui-ci, une seule fonction est nécessaire, car l'application n'utilise qu'un seul graphique.

```
public function homepage()
{
    $surveys = Survey::where('id_user',Auth::id())->get();
    for($x = 0; $x < count($surveys);$x++){
        $surveys[$x]->user = count(User::where('id_survey',$surveys[$x]->id)->get());
    }
    $surveys = ChartResource::collection($surveys);
    return Inertia::render('HomePage', ['surveys' => $surveys]);
}
```

Cette fonction doit retourner uniquement les sondages créés par l'utilisateur et le nombre de personnes ayant répondu au sondage. Dès que ces informations sont récupérées, celles-ci sont transmises au fichier **ChartResource.php** qui a été créé au préalable. Ce fichier sert à séparer les données inutiles dans certains cas pour éviter de surcharger le front avec des informations que celui-ci n'utilise pas, comme ici : la date de création, la date de dernière modification ou encore l'image dans ce cas.

```
public function toArray(Request $request): array
{
    return [
        'id' => $this->id,
        'title' => $this->title,
        'nbAnswer' => $this->user,
    ];
}
```

Dans ce fichier, la fonction **toArray** est celle qui va filtrer les données et retourner uniquement les valeurs choisies.

3.6 Rôles et permissions

L'application doit pouvoir gérer les accès de certains utilisateurs grâce au rôle et aux permissions qui leur sont accordés. Donc, comme pour les graphiques, la librairie **Laravel-permission** permettant cette gestion est déjà installée.

Tout d'abord, récupérer les fichiers permettant la configuration et la création de nouveaux rôles et de permission grâce à la commande :

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

Créer ensuite un fichier **Seeder** permettant la création des 2 rôles que va contenir l'application.

```
public function run(): void
{
    $role = Role::create(['name' => 'user']);
    $permissions = Permission::create(['name' => 'user']);
    $role->syncPermissions($permissions);

    $role = Role::create(['name' => 'admin']);
    $permissions = Permission::create(['name' => 'admin']);
    $role->syncPermissions($permissions);
}
```

Ensuite, pour que les utilisateurs puissent se voir attribuer un rôle, la modification du modèle de ceux-ci est nécessaire. Ajouter donc `use HasRoles;` cet élément permet de faire comprendre à la librairie que le modèle peut recevoir un rôle. N'oubliez pas d'attribuer les rôles aux utilisateurs de tests dans le fichier **Seeder** des utilisateurs.

```
public function run(): void
{
    $users = User::factory(9)->create();
    foreach ($users as $user){
        $user->assignRole('user');
    }
    $users = User::factory(1)->create([
        'firstName'=>'admin',
        'lastName'=>'admin',
        'email'=>'admin@admin.ch'
    ]);
    foreach ($users as $user){
        $user->assignRole('admin');
    }
}
```

La prochaine étape est la vérification des accès permettant de restreindre les accès d'un certain rôle. Celle-ci peut s'effectuer à plusieurs endroits dans l'application. En premier, c'est dans le fichier de **route web.php**. Ce fichier permet d'attribuer les différentes requêtes **http** à une fonction d'un **controller** permettant par exemple l'affichage ou la création de données.

La vérification ici se fait aux travers du **middleware**, en ajoutant par exemple cet élément qui va vérifier si l'utilisateur qui souhaite accéder à l'**url « /survey/create »** possède le rôle « **user** ». Si celui-ci dispose bel et bien de ce rôle, l'accès lui est autorisé, sinon l'application lui renvoie un statut 403 lui expliquant que l'utilisateur ne dispose pas du rôle nécessaire pour accéder à cette page.

```
Route::middleware(['auth:sanctum', config('jetstream.auth_session'), 'verified', 'role:user'],
]->group(function () {
    Route::get('/surveys/create', [SurveyController::class, 'create'])->name('surveys.create');
```

Mais pour pouvoir vérifier le rôle de l'utilisateur, la modification du **middleware** est nécessaire. Celle-ci s'effectue en ajoutant l'alias « **role** » dans le fichier **app.php**.

```
->withMiddleware(function (Middleware $middleware) {
    $middleware->web(append: [
        \App\Http\Middleware\HandleInertiaRequests::class,
        \Illuminate\Http\Middleware\AddLinkHeadersForPreloadedAssets::class,
    ]);
    $middleware->alias([
        'role' => \Spatie\Permission\Middleware\RoleMiddleware::class,
    ]);
})
```

Pour utiliser les rôles dans le **controller** c'est beaucoup plus simple. En ajoutant la référence `use Illuminate\Support\Facades\Auth;` dans le fichier désiré. Cela permet à l'application de récupérer les informations de l'utilisateur qui est actuellement connecté donc de savoir son rôle. Ensuite, vérifier son rôle grâce à l'objet « **Auth** ». Voici un exemple dans le fichier **SurveyController.php**.

```
public function index()
{
    if(Auth::user()->can('admin'))
    {
        $surveys = SurveyResource::collection(Survey::paginate(12));
        return Inertia::render('Admin/SurveyDashboard', ['surveys' => $surveys]);
    }else{
        $surveys = SurveyResource::collection(Survey::where('id_user',Auth::id())->paginate(12));
        return Inertia::render('Survey/SurveyDashboard', ['surveys' => $surveys]);
    }
}
```

Le premier cas retourne toutes les données de la table **surveys** vu que l'utilisateur actuellement connecté est un administrateur. Dans l'autre cas, les données retournées sont uniquement celles que l'utilisateur a créées.

Le dernier endroit où utiliser ces fichiers est dans le **frontend** avec **Vuejs**. Ici, c'est assez simple de vérifier le rôle d'un utilisateur. En rajoutant un simple **if** qui a comme condition que l'utilisateur actuel doit avoir le rôle cité. Presque tous les affichages disposent de cette information grâce à « `$page.props.auth.user` » qui contient les informations de l'utilisateur. Voici comment c'est appliqué dans le fichier **AppLayout.vue** qui sert de **Header** :

```
<div class="ms-3 relative" v-if="$page.props.auth.user.roles[0].name == 'admin'">
```

Voici l'affichage à quoi ressemble le **Header** d'un utilisateur ayant le rôle « **admin** » :



Et voici celui d'un utilisateur ne disposant pas de ce rôle :



3.7 Export PDF

La librairie permettant les exports PDF a aussi été installée au début de l’application. L’export se déroule en deux étapes, la première est de permettre à l’utilisateur de sélectionner quels champs vont être exporté, et en second vient la génération du fichier PDF et son affichage.

La création d’un **controller** pour la gestion des exports PDF est conseillée. Donc créer ce fichier dans celui-ci deux méthodes, une pour chaque étape de l’export.

```
/**  
 * Show the page for select all element the user want to export  
 */  
public function index()  
{  
    return Inertia::render('Admin/ExportPDF');  
}  
  
/**  
 * Export the element selected by the user.  
 */  
public function export(ExportSurveyRequest $request)  
{  
    $surveys = SurveyResource::collection(Survey::all());  
  
    $pdf = PDF::loadView('pdf/pdf', ['data' => $surveys, 'title'=> $request->input('title'), 'image'=>$request->input('image')]);  
    return $pdf->stream('Survey.pdf');  
}
```

L’objet **PDF** dans la fonction « export » sert à générer le document dans un premier temps et ensuite à l’afficher. Pour pouvoir afficher le PDF, la création d’un modèle de PDF est nécessaire. Dans le dossier « ressources/views » créer un nouveau dossier « pdf » et à l’intérieur de celui-ci, créer un fichier se nommant « pdf.blade.php » qui sert donc de modèle.

```
<body>  
    <div style="text-align:center">  
        <h1>Exportation des Sondages</h1>  
    </div>  
    @foreach($data as $survey)  
        <div>  
            @if($title)  
                <h3 style="text-align:center">Titre : {{$survey['title']}}</h3>  
            @endif  
            @if($image)  
                <img src="" alt="Image">  
                <!---->  
            @endif  
        </div>  
    @endforeach  
</body>
```

Le fichier se comporte plus ou moins comme un fichier php standard. Dans celui-ci, une simple boucle traversant tout le tableau contenant les sondages et affiche les différents champs que l’utilisateur a choisis. Les images ne sont pour le moment pas exportées, un problème fait que toutes les requêtes retournent des time-outs dès que la ligne commenter est active.

Le dernier point est de créer une vue qui permet d'afficher les champs que l'utilisateur souhaite exporter. Pour ce faire, un simple petit formulaire contenant les différents champs possibles à l'exportation. La seule spécificité est que le formulaire doit être envoyé au travers d'un GET et via un tag « **a** ».

```
<form @submit.prevent="submit" class="p-2 w-full text-center mt-4">
  <label for="title" class="leading-7 text-lg text-gray-600">Quel champs voulez-vous exporter ?</label>
  <div class="flex flex-wrap mt-6">
    <div class="p-2 w-1/2">
      <input type="checkbox" v-model="form.title" inputId="Checkbox" name="Checkbox" value="Titre" />
      <label :for="'Checkbox'" class="ml-2">Titre</label>
    </div>
    <div class="p-2 w-1/2">
      <input type="checkbox" v-model="form.image" inputId="Checkbox" name="Checkbox" value="Image" />
      <label :for="'Checkbox'" class="ml-2">Image</label>
    </div>
    <div class="mt-8 w-1/7 mx-auto">
      <a :href="'pdf/export?title='+form.title+'&&image='+form.image" class="flex mx-auto text-white">
        </div>
    </div>
  </div>
</form>
```

Et voici l'affichage qui est retourné à l'utilisateur :

Export des sondages au format PDF

Quel champs voulez-vous exporter ?

Titre Image

Exporter

3.8 Export Excel

La librairie permettant les exports Excel a été installée au début de l'application. La librairie propose d'exporter mais aussi d'importer des fichiers **XLSX** et **CSV**. Le client ne souhaitant pour le moment avoir uniquement la fonctionnalité d'export, aucun besoin de se pencher sur l'import.

Comme pour l'exportation PDF, l'utilisateur doit pouvoir choisir quels champs doivent être exportés. Donc, commencez par créer un **controller** permettant de gérer les différentes fonctionnalités.

```
/**
 * Show the page for select all element the user want to export
 */
public function index()
{
  return Inertia::render('Admin/ExportExcel');
}

/**
 * Export the element selected by the user.
 */
public function export(ExportUserRequest $request)
{
  $firstName = $request->input('firstName');
  $lastName = $request->input('lastName');
  $email = $request->input('email');
  $users = User::all();
  $users = ExcelResource::collection($users);
  return Excel::download(new UserExport($users,$firstName,$lastName,$email), 'users'.time().'.xlsx');
}
```

Comme pour les PDFs, le **controller** ne contient que deux fonctions. La première qui sert à afficher la vue avec les différents champs exportables. Et la seconde qui génère le fichier **XLSX**. La génération se fait en plusieurs étapes, tout d'abord la récupération des champs que l'utilisateur souhaite exporter. Ensuite, un tri est effectué grâce à un fichier **resource** se nommant **ExcelResource.php**. Qui retourne uniquement les champs choisis par l'utilisateur.

```
public function toArray(Request $request): array
{
    return [
        'Nom' => $this->when($request->lastName == 'true', $this->lastName),
        'Prénom' => $this->when($request->firstName == 'true', $this->firstName),
        'Email' => $this->when($request->email == 'true', $this->email),
    ];
}
```

Ensuite, le **controller** envoie les données au fichier **UserExport.php** et lui donne en paramètres les données et les champs à exporter. Celui-ci retourne ensuite une **collection** formatée pour générer le fichier **XLSX**.

```
public function __construct($data, $firstName, $lastName, $email)
{
    $this->data = $data;
    $this->firstName = $firstName;
    $this->lastName = $lastName;
    $this->email = $email;
}

/**
 * @return \Illuminate\Support\Collection
 */
public function collection()
{
    return $this->data;
}
```

Le fichier peut dès à présent être généré mais le tableau dans le fichier **XLSX** ne contient pas d'en-têtes, donc celui-ci n'est pas très compréhensible. Pour rajouter des en-têtes et retourner les données sous forme de **collection**, la **class** doit implémenter ces deux interfaces.

```
class UserExport implements FromCollection, WithHeadings
```

Ensuite, ajouter la fonction « **headings** » qui sert à la création des en-têtes. Ici, une vérification sur les champs sélectionnés par l'utilisateur est effectuée pour éviter de créer un décalage en ajoutant un en-tête qui ne contient pas de données.

```
public function headings(): array{
    $heading = [];
    if($this->lastName == 'true'){
        array_push($heading, 'Nom');
    }
    if($this->firstName == 'true'){
        array_push($heading, 'Prénom');
    }
    if($this->email == 'true'){
        array_push($heading, 'Email');
    }
    return $heading;
}
```

Le document est maintenant généré et remplis parfaitement les attentes du client.

Nom	Prénom	Email
Abdul Kassulke	Chelsie Metz	xpredovic@example.net
Ms. Makenzie Steuber	Celestino Russel	yshields@example.org

Mais une dernière partie doit être effectuée. C'est l'affichage des champs qui va être retrouvé dans le document Excel. Pour ce faire, la méthode est la même que pour l'exportation PDF. La création d'un affichage permettant à l'utilisateur de choisir les champs à exporter.

Exporter la liste des utilisateurs au format Excel

Quel champs voulez-vous exporter ?

Nom

Prénom

Email

Exporter

Le formulaire et la méthode d'envoi sont identiques à celui de l'exportation PDF avec juste un champ qui a été ajouté.

```
<div class="mt-8 w-1/7 mx-auto">
|   <a :href="'excel/export?firstName='+form.firstName+'&&lastName='+form.lastName+'&&email='+form.email"
|</div>
```

3.9 Répondre aux sondages

L'application doit permettre à des utilisateurs non authentifiés de répondre au sondage via un URL sécurisé. Le plus simple pour cela est d'utiliser un **uuid** comme **id** pour la table **survey**, ceci permet à l'application de retrouver l'id du sondage très simplement et sécurise le lien, car ici l'utilisateur ne peut pas accéder à un autre sondage juste en changeant le numéro dans l'url.

localhost:8000/surveys/2 => localhost:8000/surveys/9c2cad3e-5821-46d8-a2e8-35f8dbb3eafb

Pour permettre la génération d'un **uuid**, cela se déroule dans le fichier de migration de la table **survey**. Ici, la clé primaire va être attribuée à un élément **uuid**, qui va donc la générer automatiquement à chaque nouvel entrer dans la table. Les autres tables comprenant une **FK** qui est reliée à cette table doivent faire référence à un **uuid**.

Pour l'ID :

```
$table->uuid('id')->primary(); $table->foreignUuid('id_survey')
```

Pour la FK :

Ensuite l'application a besoin d'un formulaire pour pouvoir accueillir les réponses des utilisateurs. Donc la création d'une vue permettant l'affichage et l'envoie du formulaire. Celui-ci ressemble aux formulaires des méthodes **CREATE**, mais s'en la barre de navigation.

Rerum aut labore eve



Vos informations

Nom	Prénom	Ville
<input type="text"/>	<input type="text"/>	<input type="text"/>

Questions

Question Text

Le formulaire est ensuite envoyé au **controller** pour la création des réponses et l'enregistrement des informations de l'utilisateur.

```
public function store(StoreAnswerRequest $request, $id)
{
    $user = User::create(array_merge($request->validated(), [
        'firstName' => $request->firstName,
        'lastName' => $request->lastName,
        'city' => $request->city,
        'id_survey' => $id,
    ]));
    foreach($request->questions as $question){
        if($question['type'] == "Text"){
            if($question['userAnswers'] != null){
                $answer = Answer::create([
                    'content' => $question['userAnswers'],
                    'id_question' => $question['id'],
                ]);
                UserAnswer::create([
                    'id_user' => $user->id,
                    'id_answer' => $answer->id,
                ]);
            }
        }
    }
}
```

La fonction **store** va créer un utilisateur et générer les réponses fournies par celui-ci. À la suite de ça, l'utilisateur se voit remercier d'avoir répondu au sondage sur une page annexe.



Merci d'avoir répondu au sondage : test tpi

3.10 Affichage des réponses

Comme validé avec le client, la page de réponse affiche uniquement le nombre de personnes ayant choisi cette réponse pour les questions de type **Select** et **CheckBox**. Et les réponses de type **Text**, affichent le contenu de celle-ci dans un conteneur avec une pagination.

Voici une question de type Text ?

Ans-jlg
Ans-rai
Ans-wex
Ans-nfx

« Previous 1 Next »

Voici une question de type Select ?

Ans-oxt 0	Ans-frk 0	Ans-qzo 2	Ans-spw 0
--------------	--------------	--------------	--------------

Voici une question de type Check Box ?

Ans-pbo 2	Ans-qpp 2	Ans-fjv 2	Ans-kkh 0
--------------	--------------	--------------	--------------

Le **controller** dispose d'une fonction « **getAnswer** » qui lui permet de retourner toutes les réponses triées par questions. Si la question est de type **Text**, le controller retourne le contenu de toutes les réponses que celui-ci a trouvées avec une pagination. Si la question n'est pas de ce type, le contenu est retourné en plus du nombre de fois que les utilisateurs ont choisi cette réponse.

```
public function answer($id)
{
    $survey = Survey::where('id', $id)->get();
    $survey['user'] = User::where('id', $survey[0]->id_user)->get();
    if(Auth::user()->can('admin') || $survey[0]->id_user == Auth::id())
    {
        $survey['questions'] = Question::where('id_survey', $id)->get();
        foreach($survey['questions'] as $question){
            if($question->type == "Text"){
                $question['answer'] = AnswerResource::collection(Answer::where('id_question', $question->id)->paginate(4));
            }
            else{
                $question['answer'] = Answer::where('id_question', $question->id)->get();
                foreach($question['answer'] as $answer)
                {
                    $answer['count'] = count(UserAnswer::where('id_answer', $answer->id)->get());
                }
            }
        }
    }
}
```

3.11 Description des tests effectués

Les tests effectués sont des tests unitaires. Les tests ne valident que les méthodes concernant les utilisateurs.

1. Test sur la connexion des utilisateurs à l'application web :

1.1. L'utilisateur peut accéder à la page de connexion :

Ce test permet de savoir si un utilisateur lambda peut accéder à la page de connexion.

```
public function test_login_screen_can_be_rendered(): void
{
    $response = $this->get('/login');

    $response->assertStatus(200);
}
```

1.2. L'utilisateur peut se connecter :

```
public function test_users_can_authenticate_using_the_login_screen(): void
{
    $user = User::factory()->create();

    $response = $this->post('/login', [
        'email' => $user->email,
        'password' => 'password',
    ]);

    $this->assertAuthenticated();
    $response->assertRedirect(route('home', absolute: false));
}
```

1.3. L'utilisateur ne peut pas se connecter avec de faux identifiant :

```
public function test_users_can_not_authenticate_with_invalid_password(): void
{
    $user = User::factory()->create();

    $this->post('/login', [
        'email' => $user->email,
        'password' => 'wrong-password',
    ]);

    $this->assertGuest();
}
```

Les résultats attendus sont que l'utilisateur puisse accéder à la page de connexion. Que l'utilisateur puisse se connecter avec ces identifiants et que celui-ci accède ensuite à la homepage. Et que l'utilisateur ne puisse pas se connecter avec de faux identifiants. Voici les résultats obtenus suite à l'exécution de ces tests :

PASS	Tests\Feature\AuthenticationTest
✓	login screen can be rendered
✓	users can authenticate using the login screen
✓	users can not authenticate with invalid password

2. Création d'un compte utilisateur :

2.1. L'utilisateur peut accéder à la page d'enregistrement :

```
public function test_registration_screen_can_be_rendered(): void
{
    $response = $this->get('/register');

    $response->assertStatus(200);
}
```

2.2. L'utilisateur peut créer un compte :

```
public function test_new_users_can_register(): void
{
    if (! Features::enabled(Features::registration())) {
        $this->markTestSkipped('Registration support is not enabled.');
    }

    $response = $this->post('/register', [
        'lastName' => 'Test User',
        'firstName' => 'Test User',
        'email' => 'test@example.com',
        'password' => 'password',
        'password_confirmation' => 'password',
        'terms' => Jetstream::hasTermsAndPrivacyPolicyFeature(),
    ]);

    $this->assertAuthenticated();
    $response->assertRedirect(route('home', absolute: false));
}
```

Les résultats attendus sont que l'utilisateur puisse accéder à la page pour créer un compte. Que l'utilisateur puisse se créer un compte depuis l'interface. Voici les résultats obtenus suite à l'exécution des tests :

```
PASS Tests\Feature\RegistrationTest
✓ registration screen can be rendered
✓ new users can register
```

3. Modification de mot passe :

3.1. L'utilisateur peut modifier son mot de passe :

```
public function test_password_can_be_updated(): void
{
    $this->actingAs($user = User::factory()->create());

    $this->put('/user/password', [
        'current_password' => 'password',
        'password' => 'new-password',
        'password_confirmation' => 'new-password',
    ]);

    $this->assertTrue(Hash::check('new-password', $user->fresh()->password));
}
```

3.2. L'utilisateur peut se connecter avec son nouveau mot de passe :

```
public function test_current_password_must_be_correct(): void
{
    $this->actingAs($user = User::factory()->create());

    $response = $this->put('/user/password', [
        'current_password' => 'wrong-password',
        'password' => 'new-password',
        'password_confirmation' => 'new-password',
    ]);

    $response->assertSessionHasErrors();

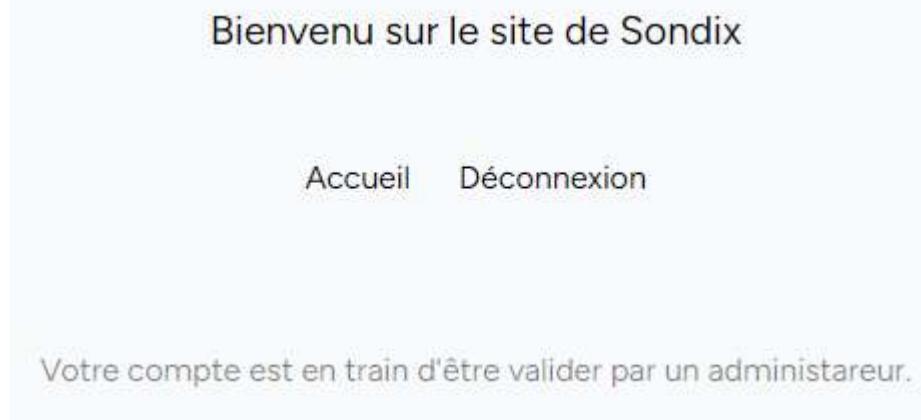
    $this->assertTrue(Hash::check('password', $user->fresh()->password));
}
```

Les résultats attendus sont que la modification du mot de passe a bien été effectuée. Et que l'utilisateur puisse se connecter avec son nouveau mot de passe. Voici les résultats obtenus suite à l'exécution des tests :

PASS Tests\Feature\UpdatePasswordTest
✓ password can be updated
✓ current password must be correct

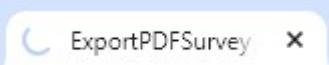
3.12 Erreurs restantes

- Lorsqu'un utilisateur se connecte, celui-ci arrive forcément sur la page d'accueil avec un message disant que son compte doit être vérifié par un administrateur. Une correction possible est de réussir à récupérer le rôle de l'utilisateur sur la page d'accueil.



- Si les images sont exportées au format PDF, celles-ci font crash violemment l'application, ne permettant plus d'accéder à aucune page du site.

Chargement jusqu'à prendre un timeout :



Pour le moment, aucune correction n'a été trouvée, l'erreur est venue de la méthode d'affichage des images dans le fichier **pdf.blade.php**.

4. Conclusion

4.1 Bilan personnel

Le projet avait pour objectif la création d'une application de gestion de sondages en ligne utilisant le Framework Laravel avec le kit de démarrage JetStream. La réalisation de ce projet m'a permis de développer de nouvelles compétences et d'approfondir mes connaissances dans plusieurs domaines techniques.

Avant de commencer ce projet, je n'avais qu'une expérience limitée avec Laravel, ayant appris à l'utiliser seulement deux à trois mois auparavant. Cette contrainte initiale m'a imposé de consacrer du temps supplémentaire à la compréhension des fonctionnalités de base de Laravel, en particulier pour les opérations CRUD (Create, Read, Update, Delete). Malgré ce défi, j'ai réussi à acquérir une maîtrise satisfaisante de ce Framework.

La gestion du temps a été un aspect critique de ce projet. La création des maquettes, qui n'était pas prévue dans la planification initiale, a remplacé la mise en place des tests, ce qui a entraîné un retard dans certaines phases du développement. Cependant, j'ai réussi à rattraper ce retard lors de la création des fonctionnalités d'exportation en PDF et Excel. En fin de compte, la documentation et la réalisation du projet ont été complétées dans les délais impartis, bien que certains ajustements aient été nécessaires.

Ce projet m'a offert la possibilité d'approfondir mes connaissances avec Laravel et la manière de gérer mon temps. Même si le projet n'est pas entièrement fini et qu'il contient quelques erreurs, je suis satisfait du résultat de mon projet pour le temps dont je disposais.

En conclusion, ce projet a été une expérience enrichissante qui m'a permis de surmonter des défis techniques et de gérer efficacement mon temps. Les compétences acquises et les leçons apprises me seront bénéfiques pour mes projets futurs et pour mon développement professionnel.

Je tiens à remercier Monsieur Bourahla pour le temps qu'il m'a consacré et pour avoir répondu à toutes mes questions de façon rapide et très précise. Mais aussi Monsieur Malherbe et Monsieur Berney pour leurs soutiens durant le suivi du TPI.

4.2 Comparaison planification initial et journal de travail

La planification est assez proche de la réalité. Mais dans la réalisation, il manquait la tâche de création de maquettes. Cette opération a malheureusement remplacé celle de la mise en place des tests. De plus, la partie de création des fonctionnalités CURD a été plus longue que prévu. Mais le temps perdu a su être rattraper sur la création des fonctionnalités permettant l'export PDF et Excel. Ce qui a permis à l'application de pouvoir, même s'ils ne sont pas tous présentss des tests unitaires.

La documentation a pris une grande partie de la réalisation du projet celle-ci a pris un peu plus de 22 heures à être réalisée sur les 88 heures prévues. La planification et l'analyse ont pris environ 15 heures dans la durée du projet. Et le reste du temps a été attribué à la réalisation du projet.

4.3 Résumé du rapport

Présentation du projet

Le projet vise à développer une application de gestion de sondages en ligne en utilisant le Framework Laravel avec le kit de démarrage JetStream, incluant Inertia.js, Vue.js et Tailwind CSS. L'objectif principal est de créer une plateforme permettant la gestion complète des sondages, depuis la conception de la base de données jusqu'à l'exportation des résultats sous forme de fichiers Excel et PDF.

Implémentation

La partie Implémentation contient toute la partie pratique du projet. Dans celle-ci, on y retrouve comment mettre en place l'environnement de travail. On utilise Vs code, Xampp, NodeJs et Composer. Il contient aussi les paramètres à sélectionner pour la création du projet Laravel. Ensuite, une explication sur l'utilité des **Model** et comment ils ont été créés. La création des fichiers **Fakers** permettant la génération de données a été mise en place et expliquée.

Ensuite, la création des fonctionnalités **CRUD**. Il a fallu implémenter les différentes fonctionnalités, celles-ci permettant de créer, lire, mettre à jour et supprimer un élément d'une table. De plus, il y est expliqué les quelques requête HTTP que l'on y retrouve et, par exemple, pourquoi utiliser une requête **Post** pour la mise à jour des données et non une **Put** ou une **Patch**.

Après une explication sur l'implémentation des graphiques et comment les utiliser. Comment les rôles et les permissions ont été gérés et les différences entre l'interface d'un utilisateur standard et un administrateur. Les administrateurs peuvent donc, exporter la liste des sondages au format PDF et la liste des utilisateurs au format Excel. Une explication est effectuée sur l'exportation PDF et de même pour l'exportation Excel.

Une explication est faite sur la manière permettant de sécuriser l'accès des sondages et comment ils sont affichés. Et après vient l'affichage des réponses. L'affichage vu par l'utilisateur est présenté et expliqué. Ensuite, une présentation des tests effectués avec les réponses attendues.

Finalement, une brève présentation des erreurs encore présentes sur l'application comme l'affichage des images sur les fichiers PDF et une solutions possibles est proposée.

Conclusion

Le projet a permis de créer une application de gestion de sondages en ligne fonctionnelle, répondant aux exigences du client en termes de sécurité, d'exportation de données et de gestion des accès utilisateurs. Malgré les défis rencontrés, tels que la mise en place des maquettes et la gestion des tests unitaires, le projet est un travail satisfaisant.

5. Soucres – Webographie

Documentation officiel Laravel :

- Site global sur les différentes fonctionnalités :

<https://laravel.com/docs/11.x>

- Mise en place du système d'exportation Excel :

<https://laravel-excel.com/>

Documentation non-officiel Laravel :

- Mise en place de l'exportation PDF :

<https://laraveldaily.com/post/laravel-dompdf-generate-simple-invoice-pdf-with-images-css>

- Supprimer un lien symbolique :

<https://stackoverflow.com/questions/48510683/laravel-how-to-revert-local-storage-symlink-and-refresh>

- Mise en place des rôles :

<https://spatie.be/docs/laravel-permission/v6/introduction>

- Implémentation du graphique :

<https://www.chartjs.org/>

- Pour la vérification des rôles dans le middleware :

<https://www.honeybadger.io/blog/laravel-permissions-roles/>

Autres sources :

- Documentation officiel de Vuejs :

<https://vuejs.org/>

- Documentation officiel d'Inertia :

<https://inertiajs.com/>

- Documentation officiel de Tailwindcss :

<https://tailwindcss.com/>

- Pour le design du site :

<https://tailblocks.cc/>

- Pour ajouter des éléments dans un tableau php :

<https://www.php.net/manual/en/function.array-push.php>

- Pour la réalisation des maquettes :

<https://www.figma.com/design/H1qNDwyQJ5LUQ61ruCoF9L/Survey?node-id=0-1&t=EsGy8KxiqJyeP0sH-0>

- Correction orthographe :

<https://quillbot.com/fr/correcteur-orthographe>

- Chat GPT comme aide pour essayer de corriger le bug avec les images sur les exports PDF :

<https://chatgpt.com/>

Images :

- Image sur la page de garde :

<https://www.interstis.fr/blog/outil-de-sondage-comment-faire-un-sondage-simple>

6. Glossaire

Controller

Elément informatique permettant de gérer les connexions entre le client et l'application web.

CRUD

Est l'acronyme de Create, Read, Update et Delete. Désigne les quatre opérations de base pour la persistance des données.

DB ou Database

C'est une base de données. Sert à stocker les données du sites web ou alors celles fournies par les utilisateurs.

FK ou Foreign Key

C'est un élément retrouver dans une table permettant de faire référence à une données précise ce situant dans une autre table.

GitFlow

Est un modèle de gestion de branche sur Git. Permet de diviser le travail par rapport des développeurs.

ID

C'est l'identifiant d'une donnée enregistrer dans la base de données.

Librairie

Une librairie désigne un entrepôt de code utilisable par des programmeurs.

Lien symbolique

C'est fichier informatique qui pointe vers un autre fichier informatique.

Pagination

C'est une méthode qui sert à diviser les éléments reçus pour éviter de surcharger le site.

Requête http

C'est une demande effectuer par le navigateur vers le serveur de l'application pour que celui-ci effectue une certaine action.

Table

Celui-ci se retrouve dans les base de données, cet élément qui contient un ensemble de données défini par le développeur.

Test Unitaire

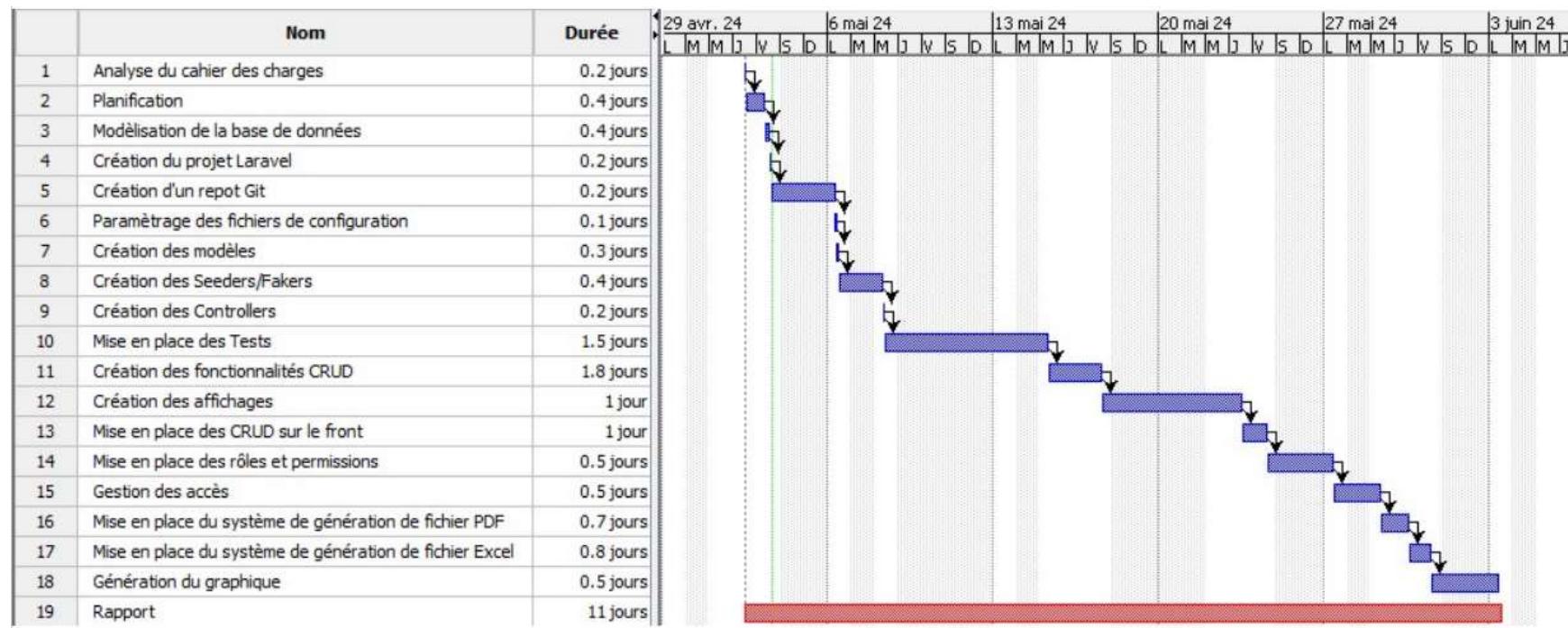
C'est un processus de vérification à effectuer sur les différentes fonctionnalités de l'application pour vérifier que celle-ci sont bien fonctionnelle.

UUID

Est l'acronyme de « Universally unique identifier ». C'est un ID générer aléatoirement contenant trente-six caractères.

7. Annexes

7.1 Planification Initial



7.2 Journal de travail

Tâches	Dates	Durées [min]	Explications / Etats	Sources, liens, références, commits ...
Bilan du jour :				
Cahier des charges		50	Prendre connaissance et analyse du cahier des charges	
Visite Expert		30	Visite du premier expert, explication du déroulement du TPI	
Rapport		15	Création du Rapport	
Création JDT	02.05.2024	10	Création du journal de travail + remplissage	
Planification		75	Début de la planification	
Bilan du jour :				
Planification		25	Finalisation de la planification + envoie	
Modélisation DB		150	Analyse et création du MCD et MLD	
Création Projet	03.05.2024	90	Création du Projet avec Laravel, installation des différents packages + paramétrage des fichiers DB	
Création Repot Git		25	Création du repot git + invitation du chef de projet	
Rapport		120	Commencer le Rapport	
JDT		5	Remplire le journal de travail	
Bilan du jour :				
Création Model		70	Création de tout les modèles + Correction des fichiers influencé par le changer de la table users	
Création Faker		45	Création de tout les fakers et seeders	
Rapport	06.05.2024	160	Remplir le rapport avec : "La mise en place de l'environnement, Model et Faker/Seeder"	
JDT		5	Remplir le journal de travail	
Pause		5	Pause donnée par le surveillant	
Branch Git		5	Création d'une nouvelle branche pour implémenter les fonctionnalités CRUD	
Création Controller		50	Création des controllers	
API Resource		20	Se renseigner sur les API Resources	

Tâches	Dates	Durées [min]	Explications / Etats	Sources, liens, références, commits ...
Bilan du jour :				
Rapport		70	Mise en forme du rapport + réécriture de certain points	
Analyse Test		60	Réflexion sur la stratégie de test et comment la mettre en place	
API Resource		70	Comprendre le fonctionnement des fichiers resources et les mettre en place	Pour le début, ils ne me sont pas trop utile, mais il me serviront plus tard
CRUD	08.05.2024	155	Commencer à mettre en place les fonctionnalités CRUD + création des affichages pour les sondages	
JDT		5	Remplire le journal de travail	
Bilan du jour :				
Congé		180	Ascension	
	09.05.2024			
Bilan du jour :				
Congé		405	Ascension	
	10.05.2024			

Tâches	Dates	Durées [min]	Explications / Etats	Sources, liens, références, commis ...
Bilan du jour :				
CRUD		300	Réaliser la possibilité aux utilisateurs de créer des sondages, ceux-ci génère directement les questions et les réponses	Cela m'a pris plus de temps que prévu
Affichage Sondage		50	Création de l'affichage des sondages (En cours)	
Pause		5	Pause donnée par le surveillant	
JDT		5	Remplir le journal de travail	
Bilan du jour :				
CRUD		250	Finaliser la création des sondages + permettre la suppression des sondages + permettre la modification des éléments	
Rapport		80	Commencer à remplir la partie sur l'implémentation du CRUD	
JDT		5	Remplir le journal de travail	
Visite Expert		25	Visite du 2ème Expert	
Bilan du jour :				
CRUD		85	Finir la modification des sondages	
Rapport		90	Documenter la partie de DELETE et UPDATE dans la réalisation	
JDT		5	Remplir le journal de travail + Corriger celui de la veille	

Page 1

Tâches	Dates	Durées [min]	Explications / Etats	Sources, liens, références, commits ...
Bilan du jour :				
CRUD	17.05.2024	130	Faire le CRUD pour les questions (En cours)	Presque fini, il ne manque juste une partie de la modification
Rapport		110	Ajouter l'explication sur les modifications des sondages et finir la suppression	
JDT		5	Remplire le journal de travail	
Maquette		160	Réalisation du MockUp du site	https://www.figma.com/design/H1gNDwyQJ5LUQ61ruCoF9L/Survey?t=Za44vT8V6ikkdLHV-1
Bilan du jour :				
Maquette	22.05.2024	305	Finalisation du MockUp	https://www.figma.com/proto/H1gNDwyQJ5LUQ61ruCoF9L/Survey?t=0IKZ3g8HqjLMEtff-0&scaling=contain&page-id=%3A1&node-id=12-1922
Affichage Sondage		20	Commencer les modifications de l'affichage	
Rapport		30	Mettre à jour les maquettes dans le rapport	
JDT		5	Remplire le journal de travail	
Bilan du jour :				
Maquette	23.05.2024	20	Discussion avec le chef de projet sur les points à modifier et les bons points	
Affichage Sondage		155	Appliquer les maquettes sur l'application	
JDT		5	Remplire le journal de travail	

Tâches	Dates	Durées [min]	Explications / Etats	Sources, liens, références, commits ...
Bilan du jour :				
CRUD		50	Finaliser les fonctionnalités CRUD pour les questions	CRUD terminé
Graphique		190	Implémentation du graphique pour la page d'accueil, création du controller et du fichier resources	Utilisé uniquement pour rendre le formulaire dynamique https://www.youtube.com/watch?v=4MdRYloy5Ms&ab_channel=MohitKumarToshniwal
Rapport		160	Apporter des corrections sur les CRUD et remplire la partie graphique	
JDT		5	Remplire le journal de travail	
Bilan du jour :				
Test		30	Mettre à jour les tests existants	
Gestion rôles		190	Commencer la gestion des roles + création de l'interface Admin (En cours)	
JDT		5	Remplire le journal de travail	
Bilan du jour :				
Gestion rôles		150	Finir l'implémentation de la gestion des rôles	
Rapport		100	Remplire la partie Rôles et permissions	
Affichage Admin		55	Création de l'affichage admin	
Export PDF		50	Commencer l'exportation PDF	
JDT		5	Remplire le journal de travail	

Tâches	Dates	Durées [min]	Explications / Etats	Sources, liens, références, commits ...
Bilan du jour :				
Rapport		70	Commenter la partie Export PDF et la partie Export Excel	
Export PDF		20	Finaliser l'exportation PDF	
Export Excel		50	Permettre à l'utilisateur d'exporter les utilisateurs sous format Excel	
Affichage réponse		35	Commencer l'affichage des réponse du formulaires	
JDT		5	Remplire le journal de travail	
Bilan du jour :				
Rapport		100	REMPLIR la partie Réponse et remplire les sources	
Démonstration		30	Présenter le rendu final au chef de projet	
JDT		5	REMPLIR le journal de travail	
Affichage réponse		270	Sécuriser l'accès au sondage et afficher les réponses	
Bilan du jour :				
Rapport		120	Intégration du glossaire, explications des tests unitaires et commencer la partie affichage des réponses	
Bilan du jour :				
Rapport		255	Finalisation du Rapport	
JDT		15	Vérification du journal de travail	
Total en heures		85.4		

7.2 Mockup

The image shows a wireframe representation of a login interface. At the top center, the word "Connexion" is displayed. Below it, there are two input fields: one for "Email" and one for "Mot de passe". Underneath the password field is a checkbox labeled "Se souvenir de moi". At the bottom of the form, there are two buttons: a white button with the text "Créer un compte" and a blue button with the text "Connexion".

Créer un compte

Prénom

Nom

Email

Mot de passe

Confirmer le mot de passe

[Vous avez déjà un compte ?](#)

Créer

Bienvenu sur le site de Sondix

[Accueil](#) [Déconnexion](#)

Votre compte est en train d'être valider par un administrateur.

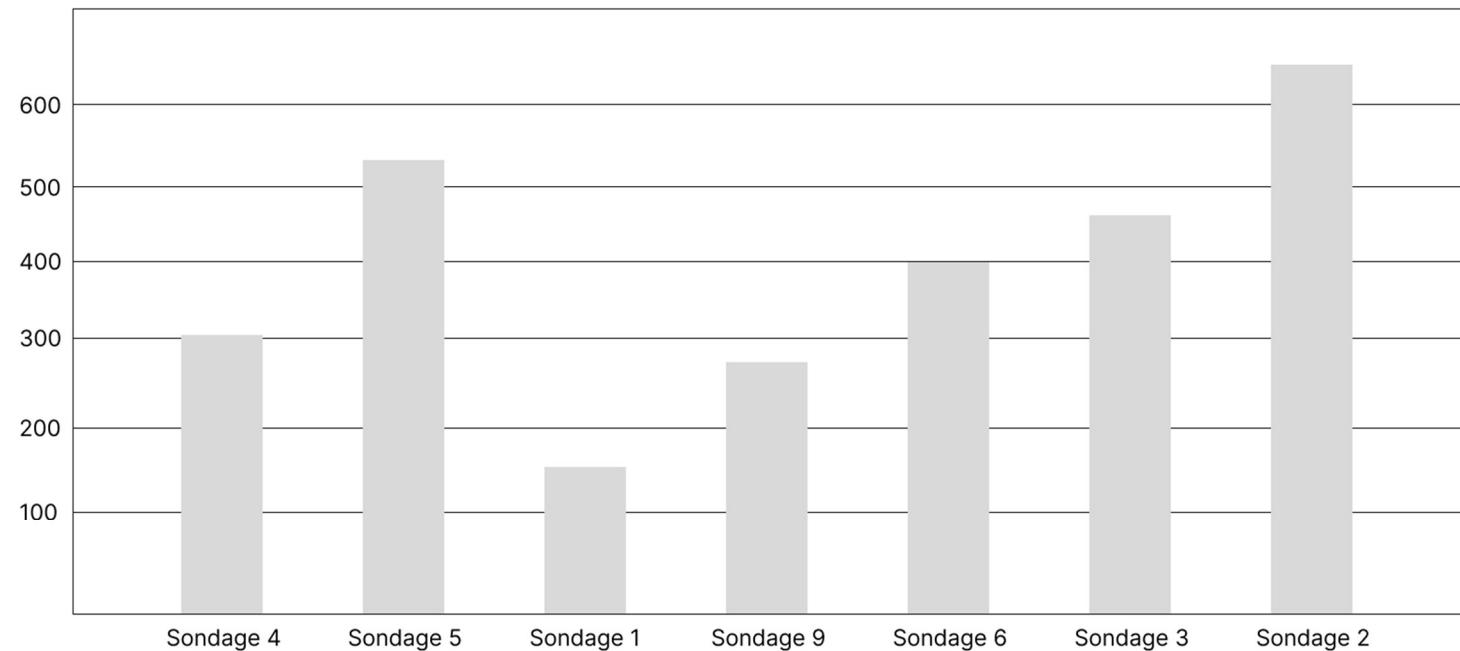


Vos sondages

Sondages

Nom utilisateur

Vos sondages





Vos sondages

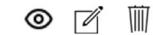
Sondages

Nom utilisateur

Gestion des sondages

Créer un sondage

Sondage numéro 1



Sondage numéro 2



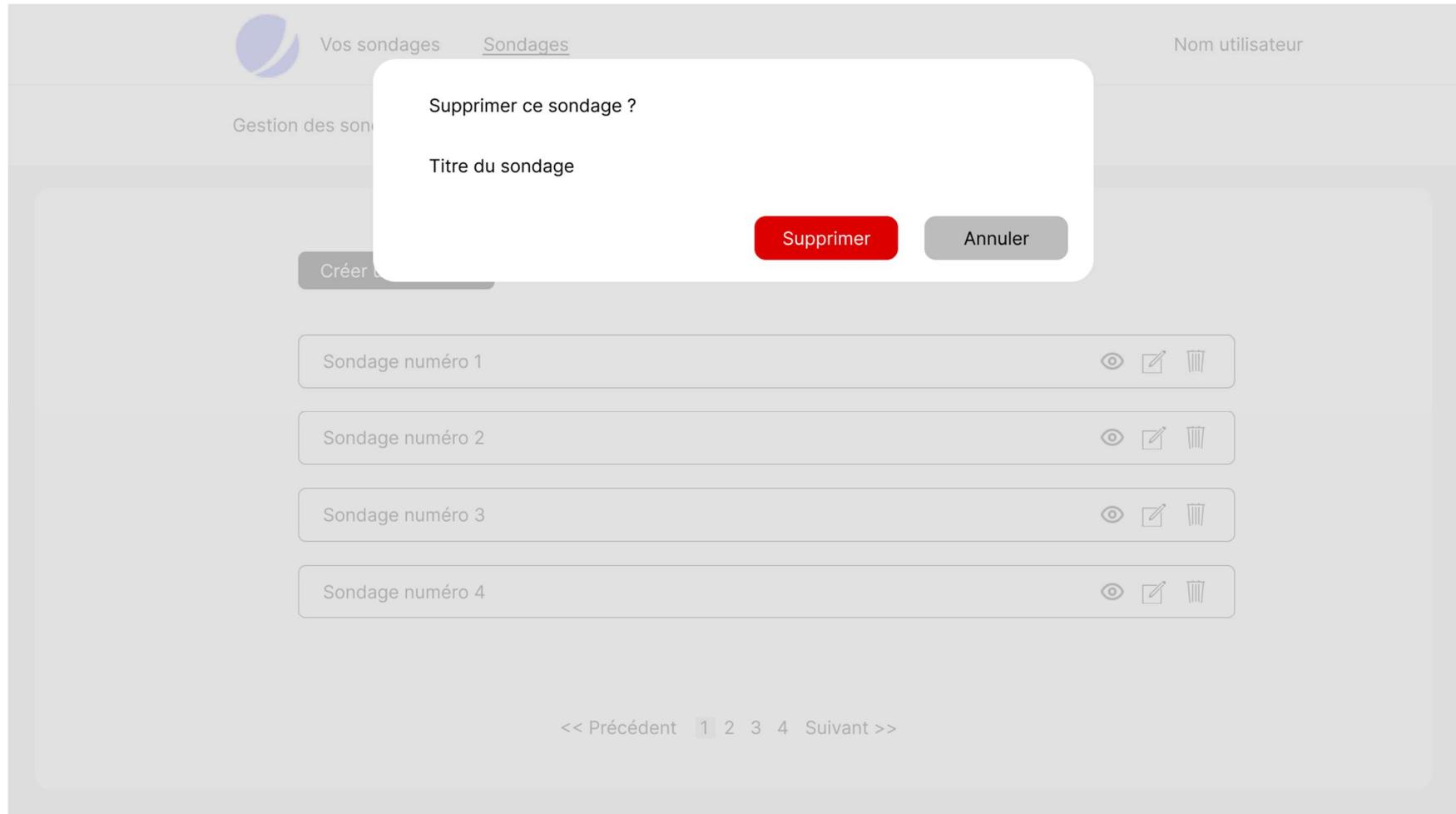
Sondage numéro 3



Sondage numéro 4



<< Précédent 1 2 3 4 Suivant >>





Vos sondages

Sondages

Nom utilisateur

Modifier un sondage

Modifier un sondage

Titre

Titre sondage 1

Image

Image sondage 1

Modifier



Vos sondages

Sondages

Nom utilisateur

Créer un nouveau sondage

Création d'un nouveau sondage

Titre

Image

Créer



Vos sondages

[Sondages](#)

Nom utilisateur

Contenu du sondage

[Voir les résultats](#)

[Gérer les questions](#)

Nom du sondage

Image du sondage

Questions

Aucun questions n'existent



Vos sondages

Sondages

Nom utilisateur

Résultat du sondage

[Voir le sondage](#)

[Voir les questions](#)

Réponse du sondage Nom du sondage

Question numéro 1

Réponse A	Réponse B	Réponse C	Réponse D
Nombre de réponse	Nombre de réponse	Nombre de réponse	Nombre de réponse

Question numéro 2

Réponse numéro 1

Réponse numéro 2

Réponse numéro 3

Réponse numéro 4

<< Précédent 1 2 3 4 Suivant >>

Question numéro 3

Réponse 1

Nombre de réponse

Réponse 2

Nombre de réponse

Réponse 3

Nombre de réponse

[Répondre](#)



Vos sondages

Sondages

Nom utilisateur

Gestions des questions

Titre du sondage

[Créer une question](#)

[Voir le sondage](#)

Question numéro 1



Question numéro 2



Question numéro 3



Question numéro 4



<< Précédent 1 2 3 4 Suivant >>



Vos sondages

Sondages

Nom utilisateur

Créer une nouvelle question

Création d'une nouvelle question

Titre de la question

Type de la question 1

Text

Select box

Select Multiple Box

Contenu de la question 1

Créer



Vos sondages

Sondages

Nom utilisateur

Créer une nouvelle question

Création d'une nouvelle question

Titre de la question

Type de la question 1

Text

Select box

Select Multiple Box

Nombre de réponse

▼

Réponse 1

Contenu de la question 1

Créer



Vos sondages

Sondages

Nom utilisateur

Contenu du sondage

Voir les résultats

Gérer les questions

Nom du sondage

Image du sondage

Questions

Question numéro 1

Question numéro 2

Question numéro 3

Réponse 1

Réponse 2

Réponse 3



Vos sondages Sondages

Nom utilisateur

Modifier votre profile

Modifier votre profile

Nom

Prénom

Adresse email

Modifier

Modifier votre MDP

Mot de passe actuelle

Nouveau mot de passe

Confirmer le mot de passe

Modifier

Vue Admin



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Gestion des utilisateurs

Gérer les inscriptions

[Créer un utilisateur](#)

[Exporter les utilisateurs](#)

Utilisateur numéro 1



Utilisateur numéro 2



Utilisateur numéro 3



Utilisateur numéro 4



<< Précédent 1 2 3 4 Suivant >>



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Exporter les créateurs

Exporter les créateurs au format Excel

Quel champs souhaitez vous exporter :

Nombre de sondage

Nom

Prénom

Exporter



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Gestion des utilisateurs

Gérer les inscriptions

Nouveau utilisateur numéro 1

Nouveau utilisateur numéro 2

Nouveau utilisateur numéro 3

Nouveau utilisateur numéro 4

<< Précédent 1 2 3 4 Suivant >>

Vos sondages Sondages Utilisateurs

Nom utilisateur

Gestion des utilisateurs

Rejeter cette inscription ?

Nom et prénom de l'utilisateur

Rejeter Annuler

Nouveau utilisateur numéro 1

Nouveau utilisateur numéro 2

Nouveau utilisateur numéro 3

Nouveau utilisateur numéro 4

<< Précédent 1 2 3 4 Suivant >>

Vos sondages Sondages Utilisateurs

Nom utilisateur

Gestion des utilisateurs

Valider cette inscription ?

Nom et prénom de l'utilisateur

Nouveau utilisateur numéro 1

Nouveau utilisateur numéro 2

Nouveau utilisateur numéro 3

Nouveau utilisateur numéro 4

<< Précédent 1 2 3 4 Suivant >>

Valider **Annuler**



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Créer un utilisateur

Créer une nouvelle utilisateur

Nom

Prénom

Email

Mot de passe

Confirmer le mot de passe

Créer



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Modifier l'utilisateur

Modifier l'utilisateur

Nom

Prénom

Adresse email

Modifier

Modifier le MDP de l'utilisateur

Nouveau mot de passe

Confirmer le mot de passe

Modifier



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Gestion des sondages

[Créer un sondage](#)

[Exporter les sondages](#)

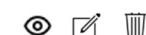
Sondage numéro 1



Sondage numéro 2



Sondage numéro 3



Sondage numéro 4



<< Précédent 1 2 3 4 Suivant >>



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Exporter les sondages

Exporter les sondages au format PDF

Quel champs souhaitez vous exporter :

Titre

Image

Exporter



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Résultat du sondage

[Voir le sondage](#)

[Voir les questions](#)

Réponse du sondage Nom du sondage

Question numéro 1

Réponse A	Réponse B	Réponse C	Réponse D
Nombre de réponse	Nombre de réponse	Nombre de réponse	Nombre de réponse

Question numéro 2

Réponse numéro 1		
Réponse numéro 2		
Réponse numéro 3		
Réponse numéro 4		

<< Précédent 1 2 3 4 Suivant >>

Question numéro 3

Réponse 1	Réponse 2	Réponse 3
Nombre de réponse	Nombre de réponse	Nombre de réponse

[Répondre](#)



Vos sondages

Sondages

Utilisateurs

Nom utilisateur

Modifier une réponse

Modification d'une réponse

Contenu de la réponse

Contenu Réponse 1

Modifier

Vue Répondant

Nom du sondage

Image du sondage

Vos informations

Nom

Prénom

Ville

Questions

Question numéro 1

Question numéro 2

Question numéro 3

Réponse 1 Réponse 2 Réponse 3

Répondre