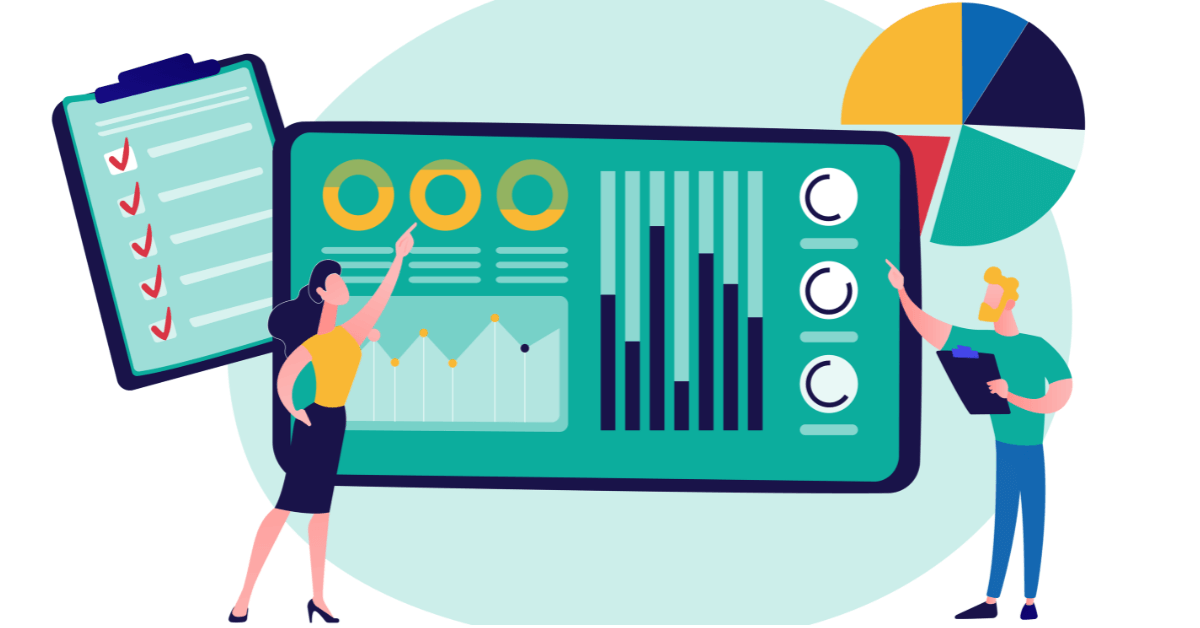
**Création d’un outil de sondage en ligne**



**Auteur : Robustiano LOMBARDO  
Chef de projet : Karim BOURAHLA**

**Experts : M. MALHERBE et M. BERNEY**

**Version : 5.1**

**Date de dernière mise à jour : 22.05.2024**

Table des matières

[**1.** **Presentation du projet** 1](#_Toc167993297)

[**2.** **Analyse / Conception** 2](#_Toc167993298)

[2.1 Base de données 2](#_Toc167993299)

[2.2 Stratégie de test 3](#_Toc167993300)

[**3.** **Implémentation** 3](#_Toc167993301)

[3.1 Mise en place de l’environnement 3](#_Toc167993302)

[3.2 Model 4](#_Toc167993303)

[3.3 Faker/Seeder 5](#_Toc167993304)

[3.4 CRUD pour les sondages 5](#_Toc167993305)

[3.5 Graphique 9](#_Toc167993306)

[3.6 Rôles et permissions 11](#_Toc167993307)

[3.7 Export PDF 14](#_Toc167993308)

[3.8 Export Excel 15](#_Toc167993309)

[3.9 Description des tests effectués 18](#_Toc167993310)

[3.10 Problèmes rencontrer 18](#_Toc167993311)

[**4.** **Conclusion** 18](#_Toc167993312)

[**5.** **Annexes** 18](#_Toc167993313)

# **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/>.

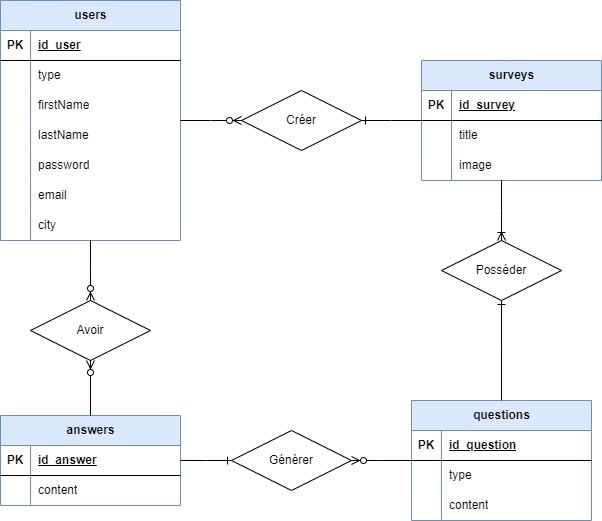
Le client évalue les points suivant :

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 chois des attributs.
4. L’administrateur peut générer un rapport pdf avec la liste des sondages et chois 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.

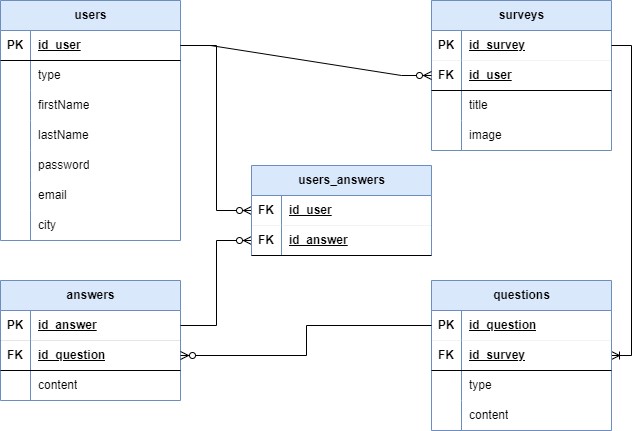
# **Analyse / Conception**

## 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.

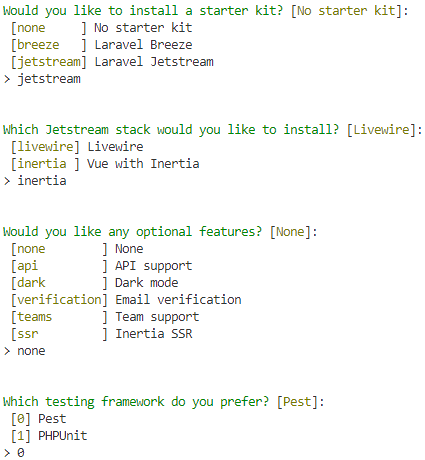
L’application s’occupent de générer de fausse données pour que les tests puissent s’exécuter sans problème. Les tests couvrent environ 90% des fonctionnalités, par exemple les tests unitaires ne peuvent pas vérifier le bon affichage du graphique, mais plutôt uniquement les données que celui-ci affiche.

# **Implémentation**

## 3.1 Mise en place de l’environnement

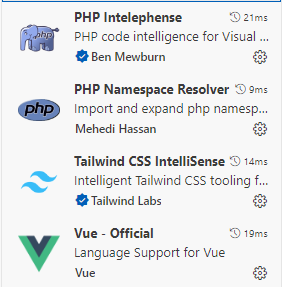
Pour commencer, le projet nécessite d’avoir une version de PHP supérieur ou égal à la 8.2, la dernière version de Composer, Node.js et Xampp installé.

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



Lorsque de la création du projet, installé dans celui-ci les 4 packages suivants : pour la gestion des rôles et des permissions, pour la génération de graphique, pour la génération de fichier PDF utiliser et pour finir qui permet 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 :

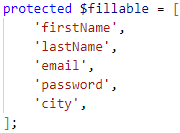
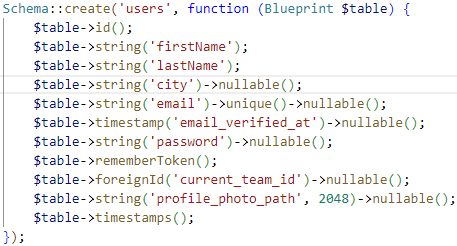


L’environnement est donc désormais propice au bon développement de cette application Laravel avec un kit Jetstream.

## 3.2 Model

La création des modèles s’effectue grâce à la commande 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.

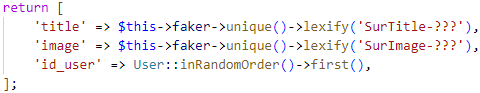
Jetsteam ayant déjà un système de gestion des utilisateurs, la modification du fichier model et du fichier de migration de la table users pour appliquer les modifications apportées à cette table.

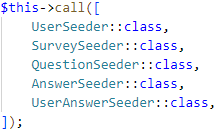
La modification des fichiers Vue ou PHP contenant des références au champ « name » est obligatoire, celui-ci n’existant plus la valeur de ce champ retourne une erreur.

## 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 :



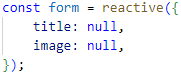
Pour les fichiers seeders appelle uniquement la classe correspondante à son faker, utilisé dans le seeder pour générer le nombre de données de prédéfini. Et dans le fichier « DatabaSeeder.php » appelé les tous les autres seeders, comme ceux-ci :



## 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** doit permettre la création d’élément **surveys**. Dans l’affichage, la création de variable est nécessaire pour récupérer données entrée par l’utilisateur.

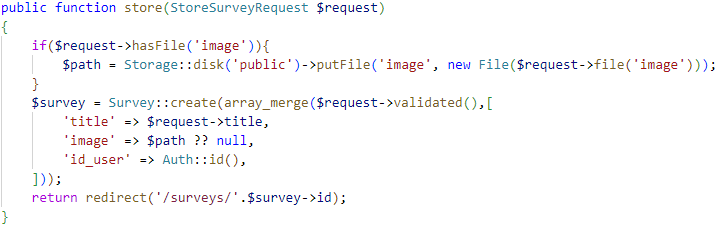


Ici la variable **form** contient un élément pour chaque **input** du formulaire, ajouter en autant que nécessaire, le but et récupérer toutes les données.

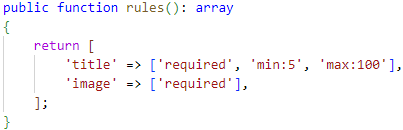
Ensuite le formulaire est envoyé au **controller** en utilisant la méthode **POST**.



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



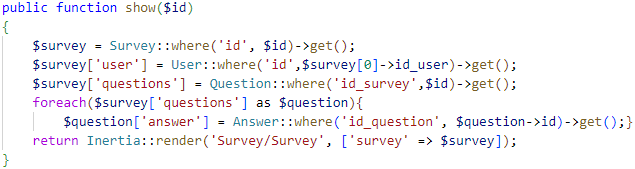
Tout d’abord, une vérification est effectuée sur la présence de l’image fournit par l’utilisateur. Ensuite si celle-ci existe, l’application va la stocker et fourni le chemin d’accès à cette image. Puis une validation des informations envoyer par l’utilisateur est effectuer. Les conditions de validation se trouve dans le fichier **StoreQuestionRequest.php** et est gérer avec la fonction **rules**.



En spécifiant les éléments du formulaire et les conditions que ceux-ci doivent respecter. Si le formulaire passe la validation, le **survey** peut être créer, sinon une message d’erreur et retourner.

Attention, la création d’un lien symbolique est obligatoire pour permettre à l’application d’enregistrer les images fournit par l’utilisateur. Ce lien se créer grâce à la commande cette commande va donc créer un lien entre le répertoire public et le répertoire où son 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 son liée. Donc voici à quoi ressemble le code dans le **controller** :



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 appliqué ces modifications.

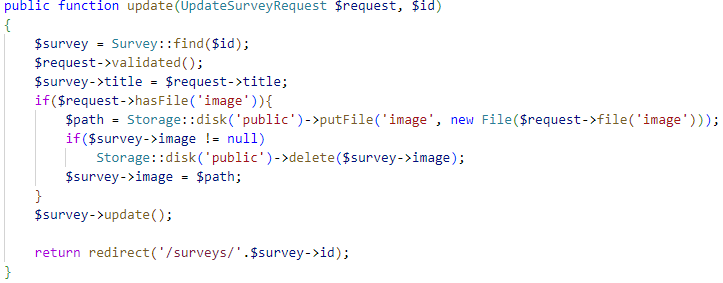
Pour le front, l’utilisation de **onBeforeMount** fourni par VueJs semble être parfaite, car celle-ci permet de définir des valeurs à certaines variables avant même que l’affichage soit chargé. Par exemple voici comment est prédéfini le titre du formulaire :



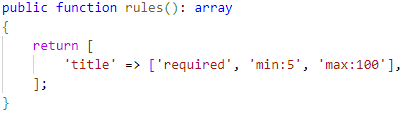
Un point très important est que les méthodes **PUT** et **PATCH** ne fonctionne pas pour l’envoie d’image vers le **controller**. Donc la seule méthode permettant d’envoyer des données restante est la méthode **POST**.



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 dans le formulaire.



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



Ici uniquement le champ **title** est vérifier 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.

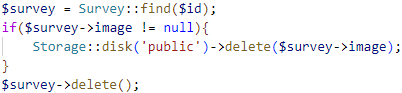
La fonctionnalité **DELETE** peut être abordé de plusieurs façon, soit en supprimant tous les éléments 1 par 1 manuellement dans le **controller**, soit en ayant paramétrer une suppression des relations en cascade. La meilleure option dans cette situation semble être la 2ème.

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 :



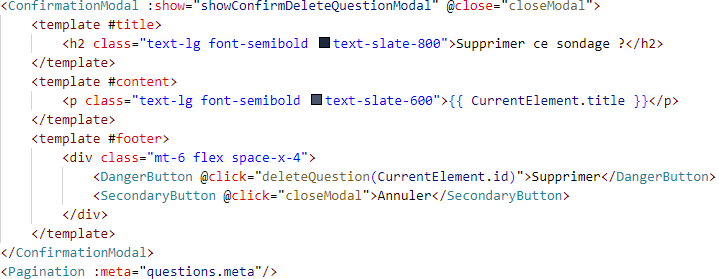
Le premier paramètre est le nom du champ, le deuxième est le nom de table pour effectuer la relation. Ensuite ce sont les actions à effectuer en cas de mise à jour ou de suppression de l’éléments en lien.

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



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

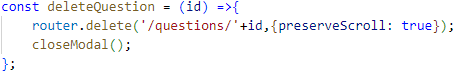
Pour la validation de suppression, en utilisant le kit de démarrage Jetstream tous les éléments nécessaires sont déjà fourni. Donc pour éviter de faire le travail en double utiliser ces composants. Les composants utiles sont « **ConfirmationModal**, **DangerButton** et **SecondaryButton** ».



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



Finalement, pour préserver le scroll de la page. Celui-ci s’applique à l’envoie 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.



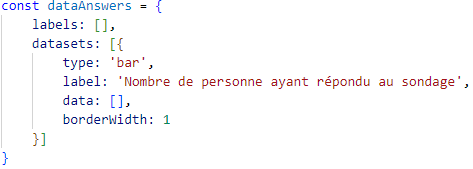
## 3.5 Graphique

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

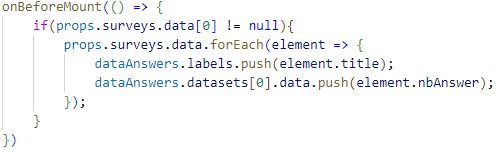
Le graphique s’affiche sur la **Homepage** de l’application. Donc pour y afficher un graphique importer sur la page ces éléments :



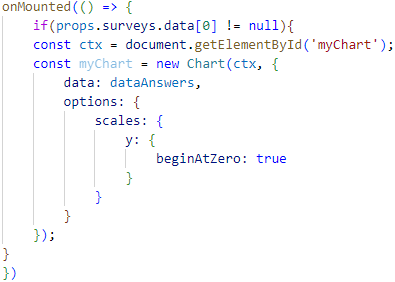
Le premier permet d’utiliser la classe **Chart** qui sers à la génération et l’affichage du graphique. **onBeforeMount** permet de charger des données avant la génération de l’affichage et **onMounted** va afficher le graphique.



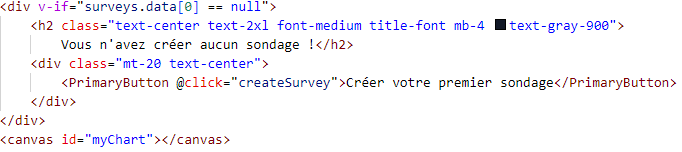
Ensuite la création d’une variable permettant par la suite de transmettre au graphique les données à afficher.



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



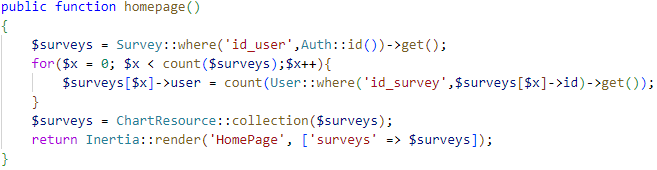
Ici, le graphique est créé avec les données qui viennent d’être récupérer. Cette méthode récupère aussi un élément html avec grâce à l’id **myChart**. Cette élément doit être de type **canvas** pour pouvoir afficher le graph.



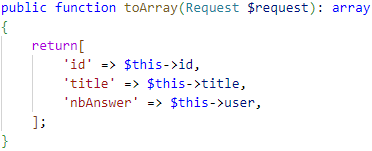
Voici à quoi ressemble la partie html permettant l’affichage, de plus si l’utilisateur n’a pas encore créer de sondage, le graph ne s’affiche pas mais à la place, du texte lui est afficher. Celui-ci lui dit qu’aucun sondage n’a été créée par l’utilisateur et un bouton est aussi afficher lui proposant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



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



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

## 3.6 Rôles et permissions

L’application doit pouvoir gérer les accès de certain utilisateur grâce au rôle et aux permissions qui leur sont accordés. Donc comme pour les graphiques la librairie **Laravel-permission** permettant cette gestion est déjà installé.

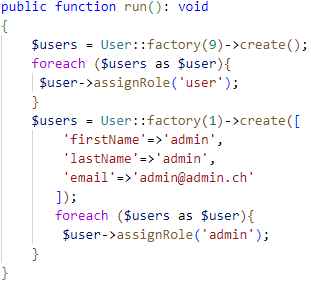
Tout d’abord, récupérer les fichiers permettant la configuration et la création de nouveau rôles et de permission grâce à la commande :



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



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

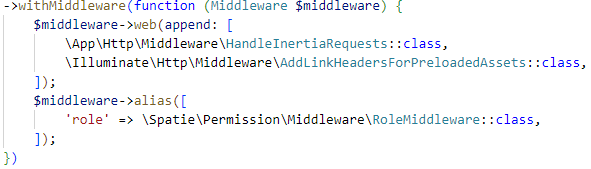


La prochaine étape est la vérification des accès permettant de restreindre les accès d’un certain rôle. Celle-ci peut s’effectuer à plusieurs endroits dans l’application. En premier, c’est dans le fichier de **route** **web.php**. Ce fichier permet d’attribuer les différentes requêtes **http** à une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

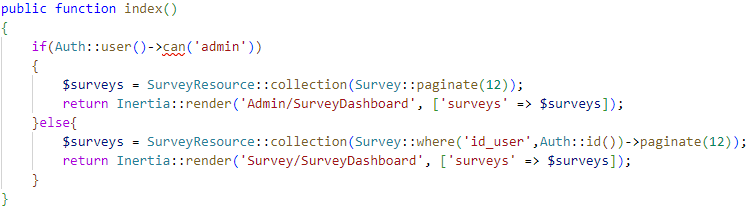
La vérification ici ce fait aux travers du **middelware**, en ajoutant par exemple cette élément qui va vérifier si l’utilisateur qui souhaite accéder à l’**url** « **/survey/create** » possède le rôle « **user** ». Si celui-ci dispose belle et bien de ce rôle, l’accès lui est autoriser, sinon l’application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



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



Pour utiliser les rôles dans le **controller** c’est beaucoup plus simple. En ajoutant la référence dans le fichier désirer. Cela permet à l’application de récupérer les informations de l’utilisateur qui est actuellement connectez donc savoir son rôle. Ensuite vérifier son rôle grâce à l’objet « **Auth** ». Voici un exemple dans le fichier **SurveyController.php**.



Le premier cas retourne toutes les données de la table **surveys** vu que l’utilisateur actuellement connecter est un administrateur. Dans l’autre cas, les données retourner sont uniquement celles que l’utilisateur à créer.

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 toutes les affichages dispose de cette information grâce à « **$page.props.auth.user** » qui contient les information de l’utilisateur. Voici comment c’est appliqué dans le fichier **AppLayout.vue** qui sers de **Header** :



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 installer 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 exporter et en second vient la génération du fichier PDF et son affichage.

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

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

L’objet **PDF** dans la fonction « export » sers 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 » est à l’intérieur de celui-ci, créer un fichier se nommant « pdf.blade.php » qui sers donc de modèle.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

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 choisi. 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 envoyer au travers d’un **GET** et via un tag « **a** ».

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Et voici l’affichage qui est retourner à l’utilisateur :

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

## 3.8 Export Excel

La librairie permettant les exports Excel a été installer au début de l’application. La libraire 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, aucunement besoin de se pencher sur l’import.

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

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Comme pour les PDFs, le **controller** ne contient que deux fonctions. La première qui sert à afficher la vue avec les différents champs exportable. Et la seconde qui génère le fichier **XLSX**. La génération se fait en plusieurs étape, 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.

Une image contenant texte, ligne, Police, capture d’écran

Description générée automatiquement

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** formater pour générer le fichier **XLSX**.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Le fichier peut dès à présent être générer 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 formes de **collection,** la **class** doit implémenter ces deux interfaces.



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

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

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

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Mais une dernière partie doit être effectuer. C’est l’affichage des champs qui vont être retrouver 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.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Le formulaire et la méthode d’envoie est identique à celui de l’exportation pdf avec juste un champ qui a été ajouter.



## 3.9 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,

## 3.10 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,

# **Conclusion**

# **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 - 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 :

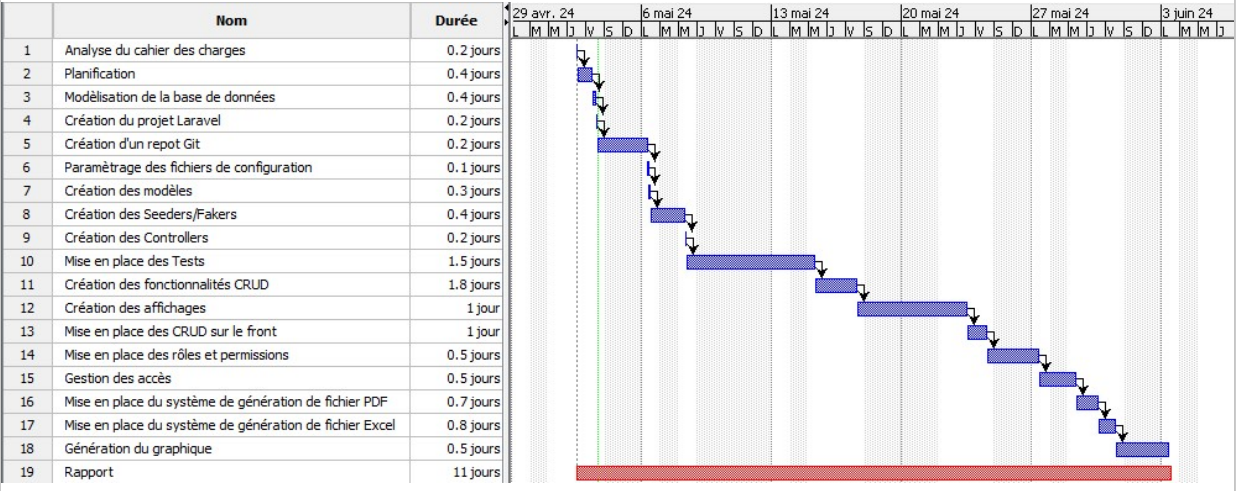
* Mise en place de l’exportation PDF : <https://laraveldaily.com/post/laravel-dompdf-generate-simple-invoice-pdf-with-images-css>
* Supprimer un lien symbolique : <https://stackoverflow.com/questions/48510683/laravel-how-to-revert-local-storage-symlink-and-refresh>
* Mise en place des rôles :

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

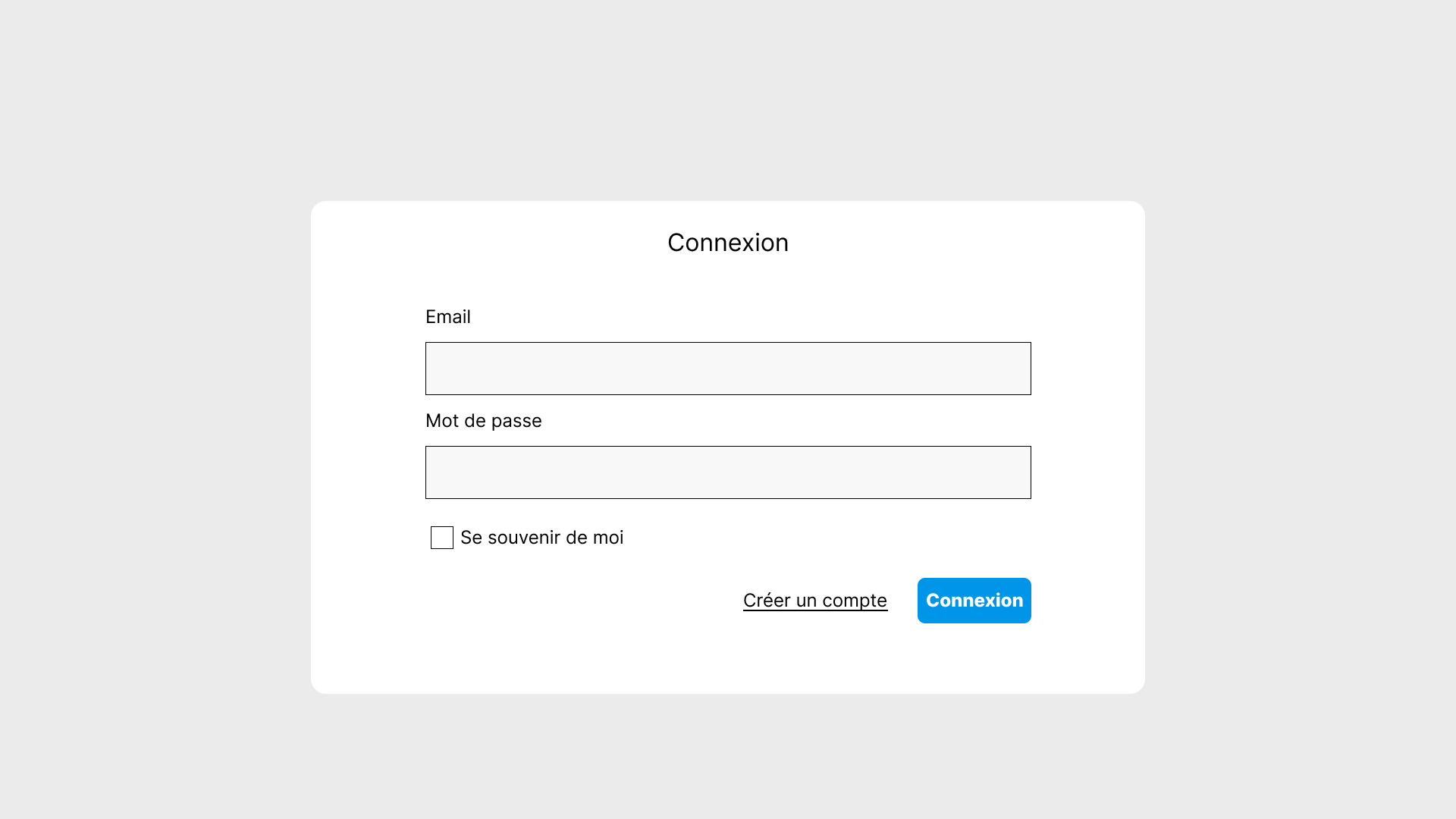
* Implémentation du graphique :

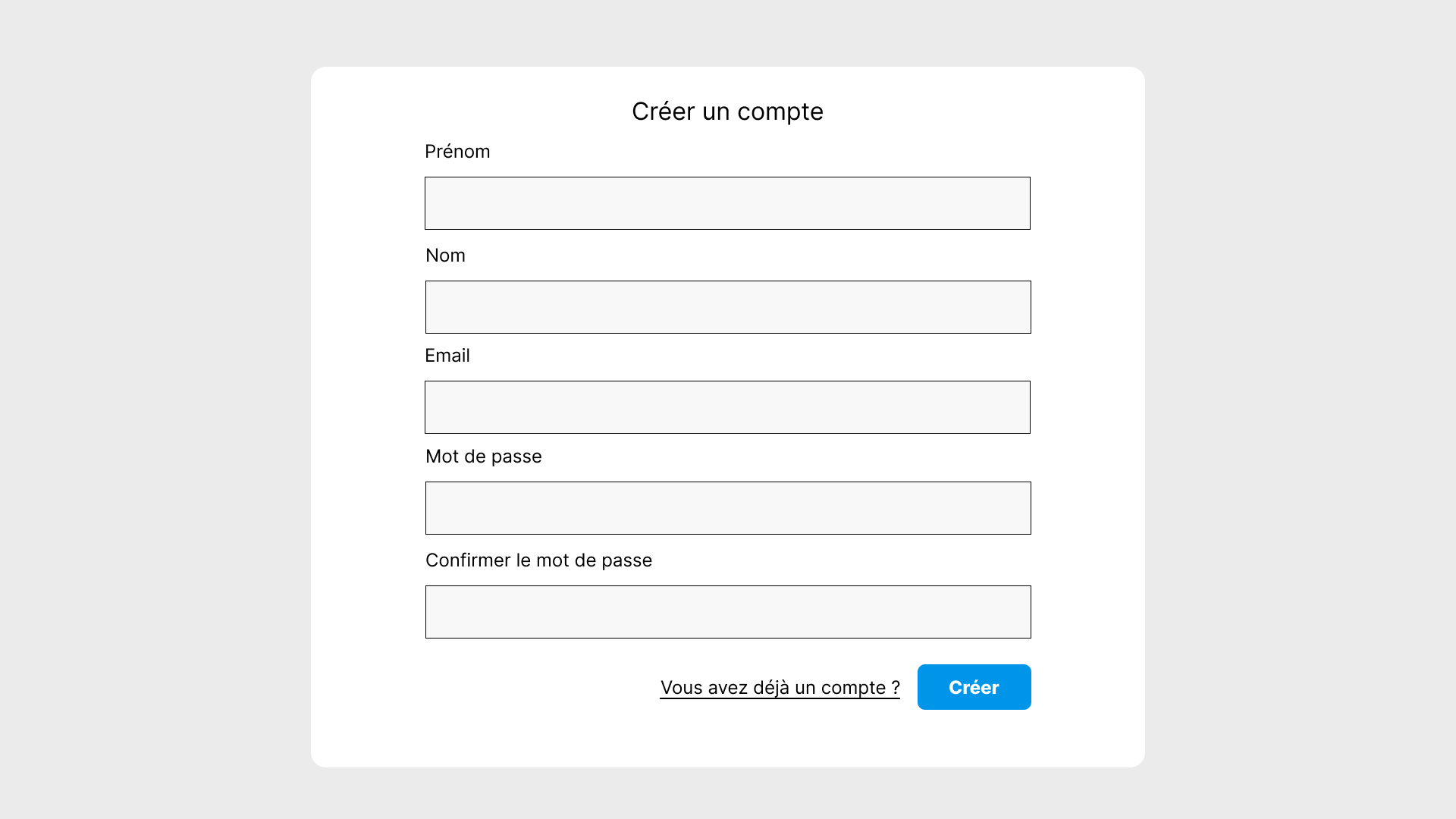
<https://www.chartjs.org/>

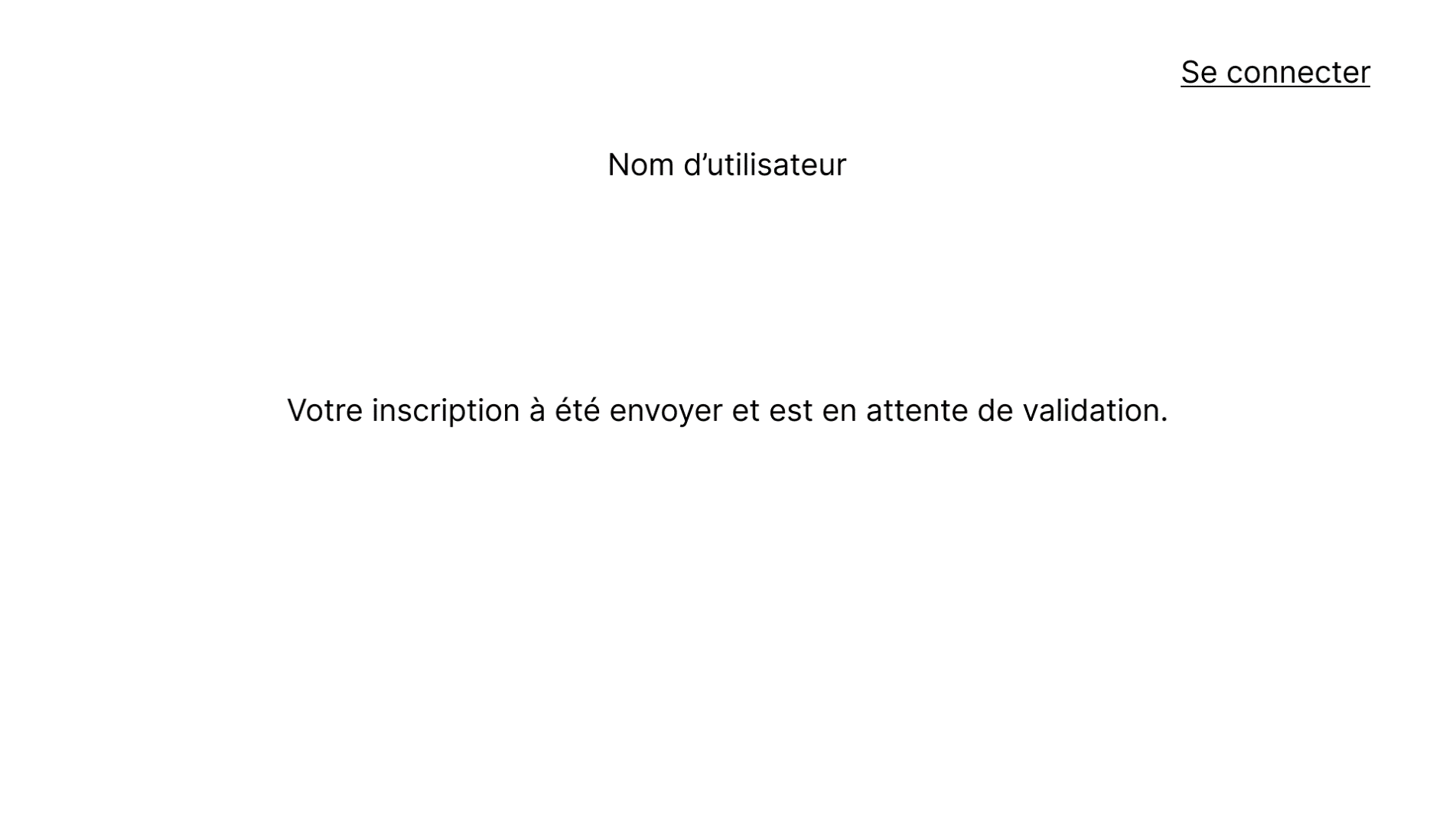
* 1. **Planification Initial**

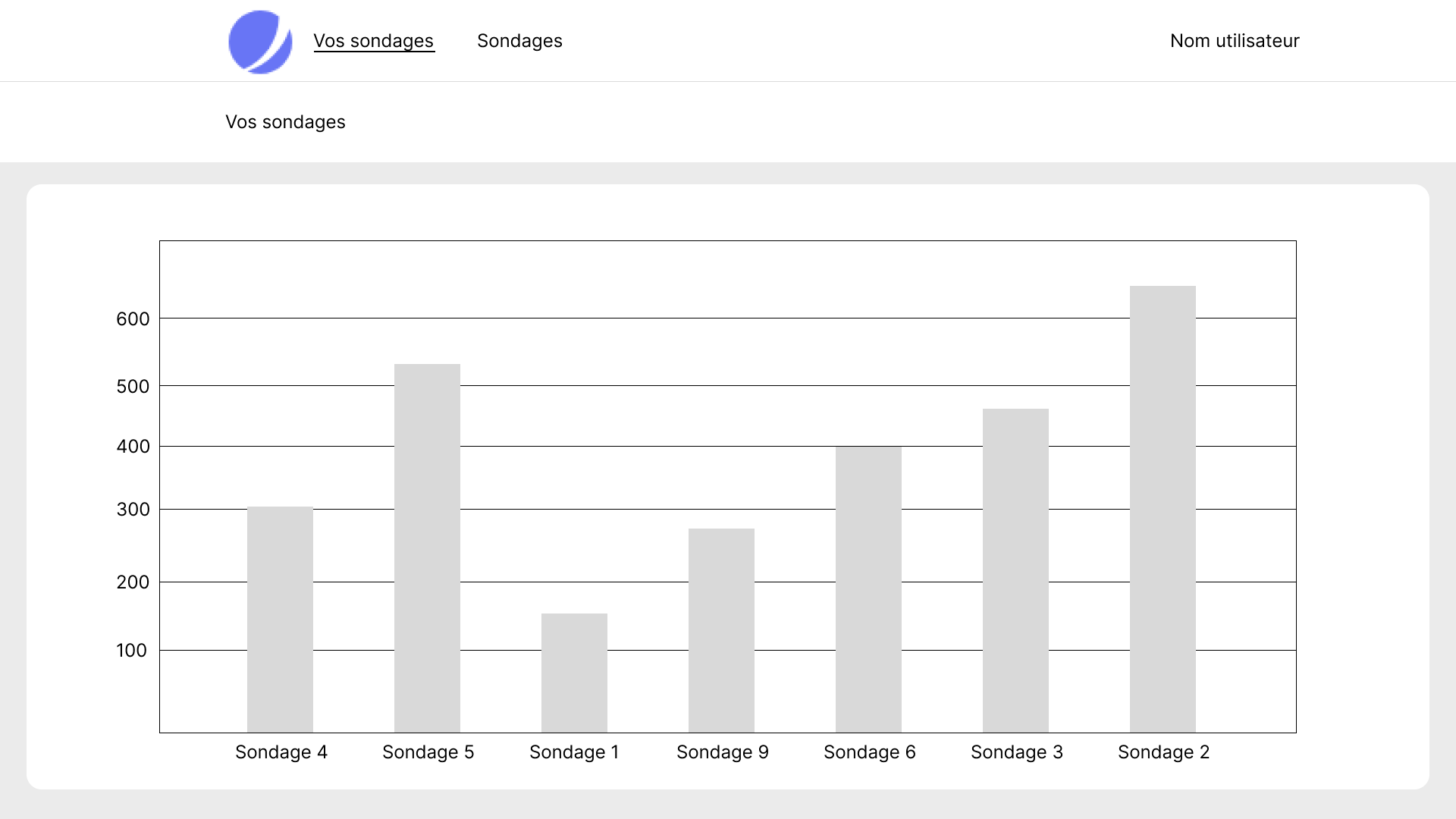


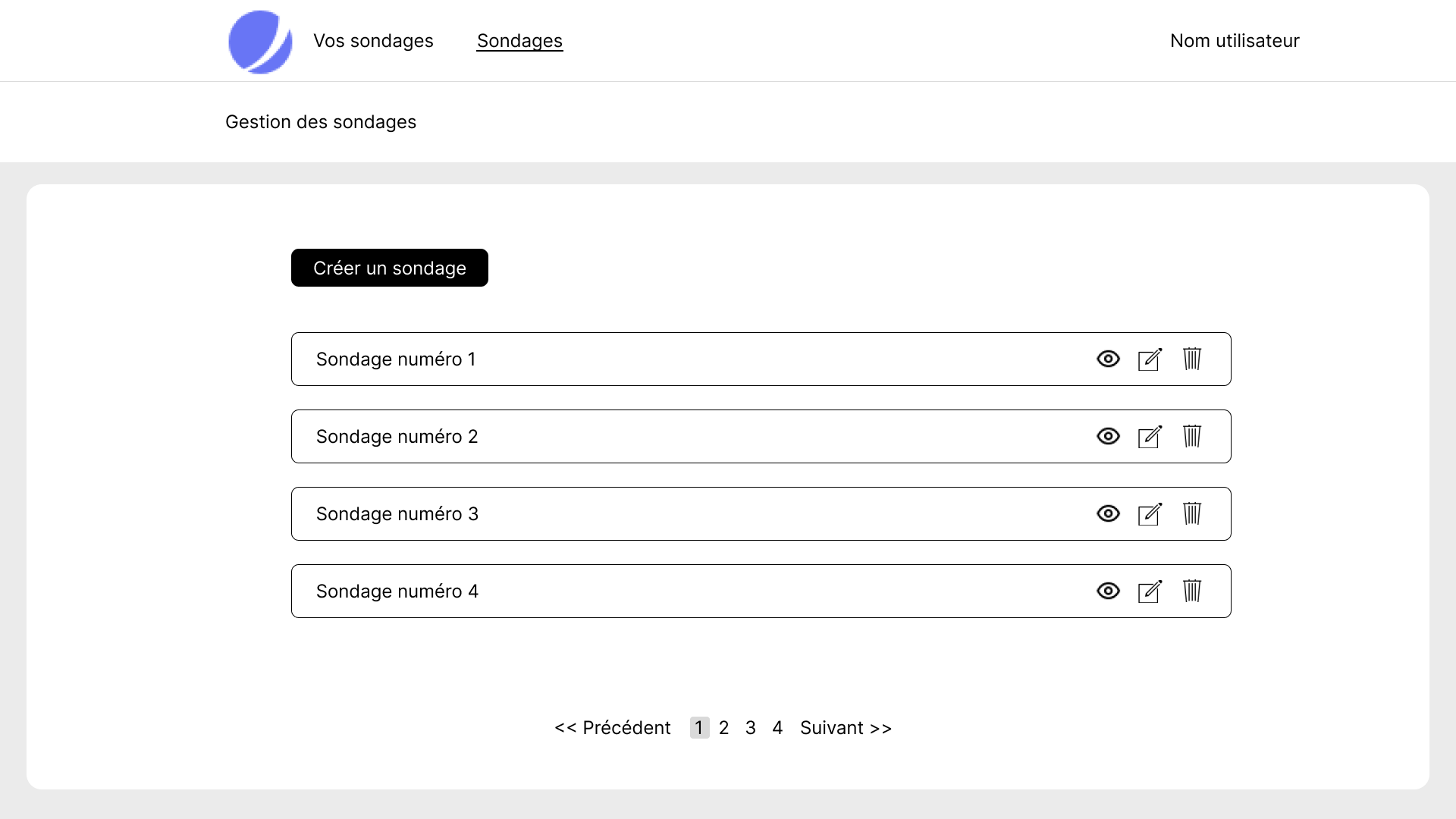
* 1. **Mockup**

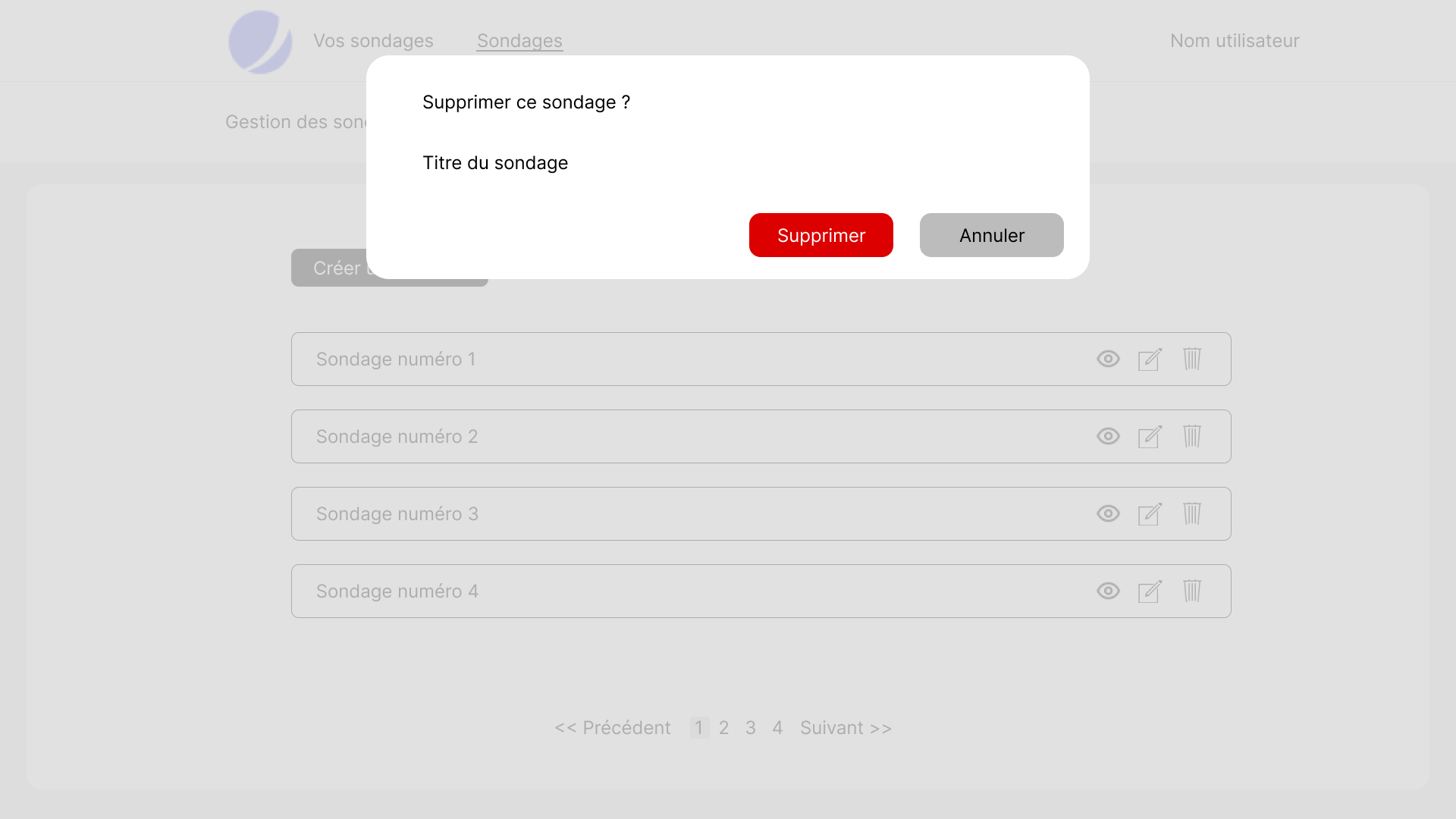
****

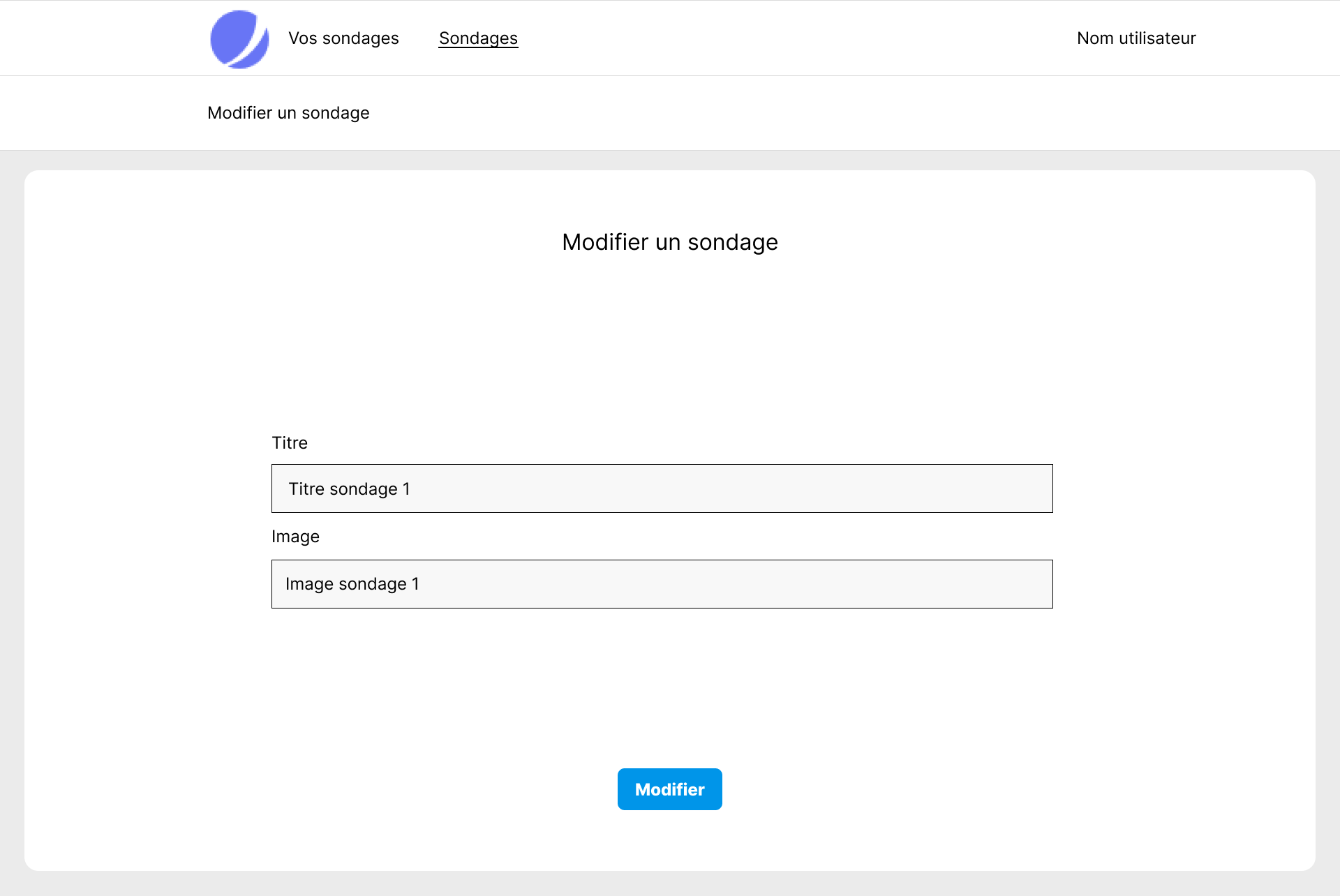
****

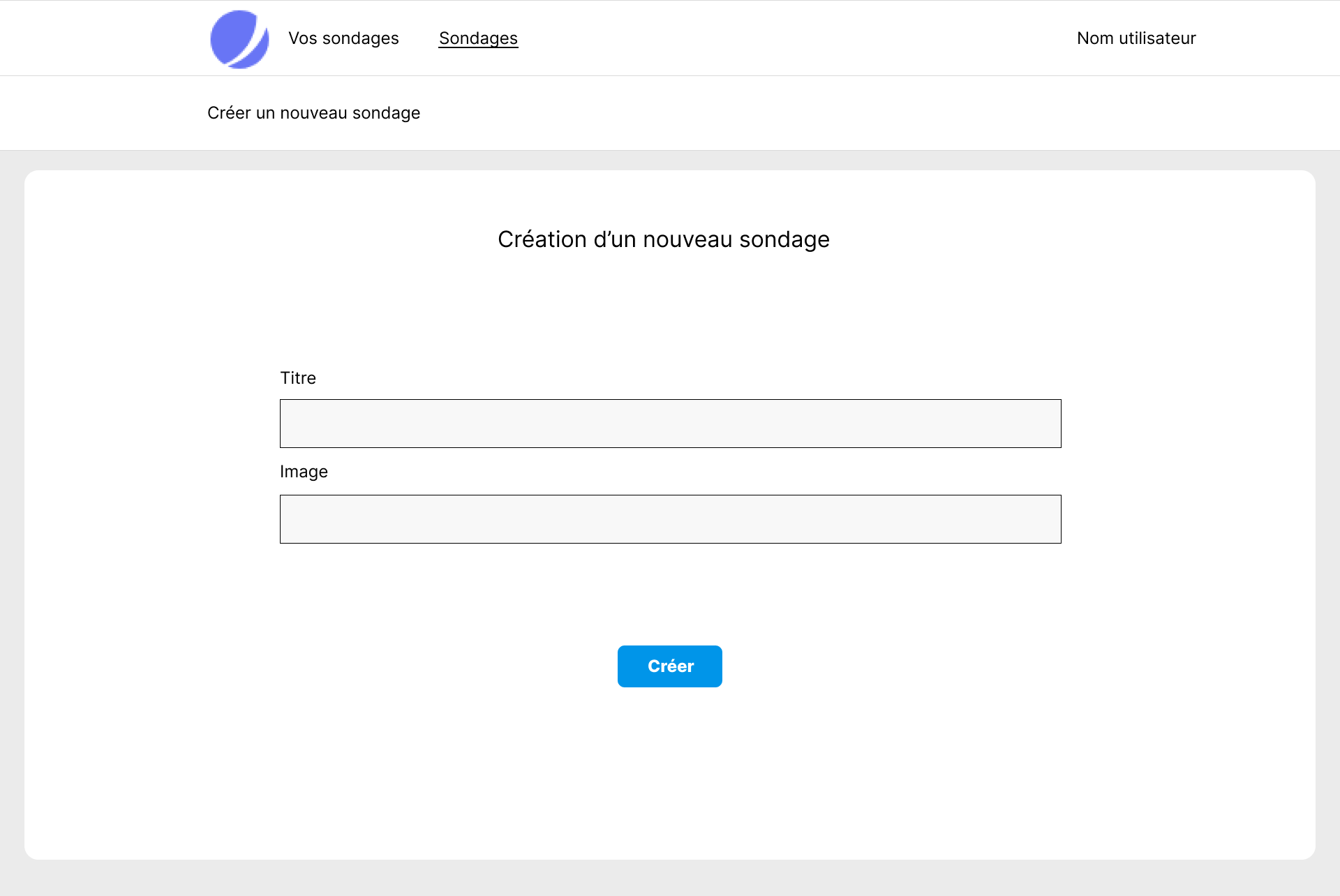
****

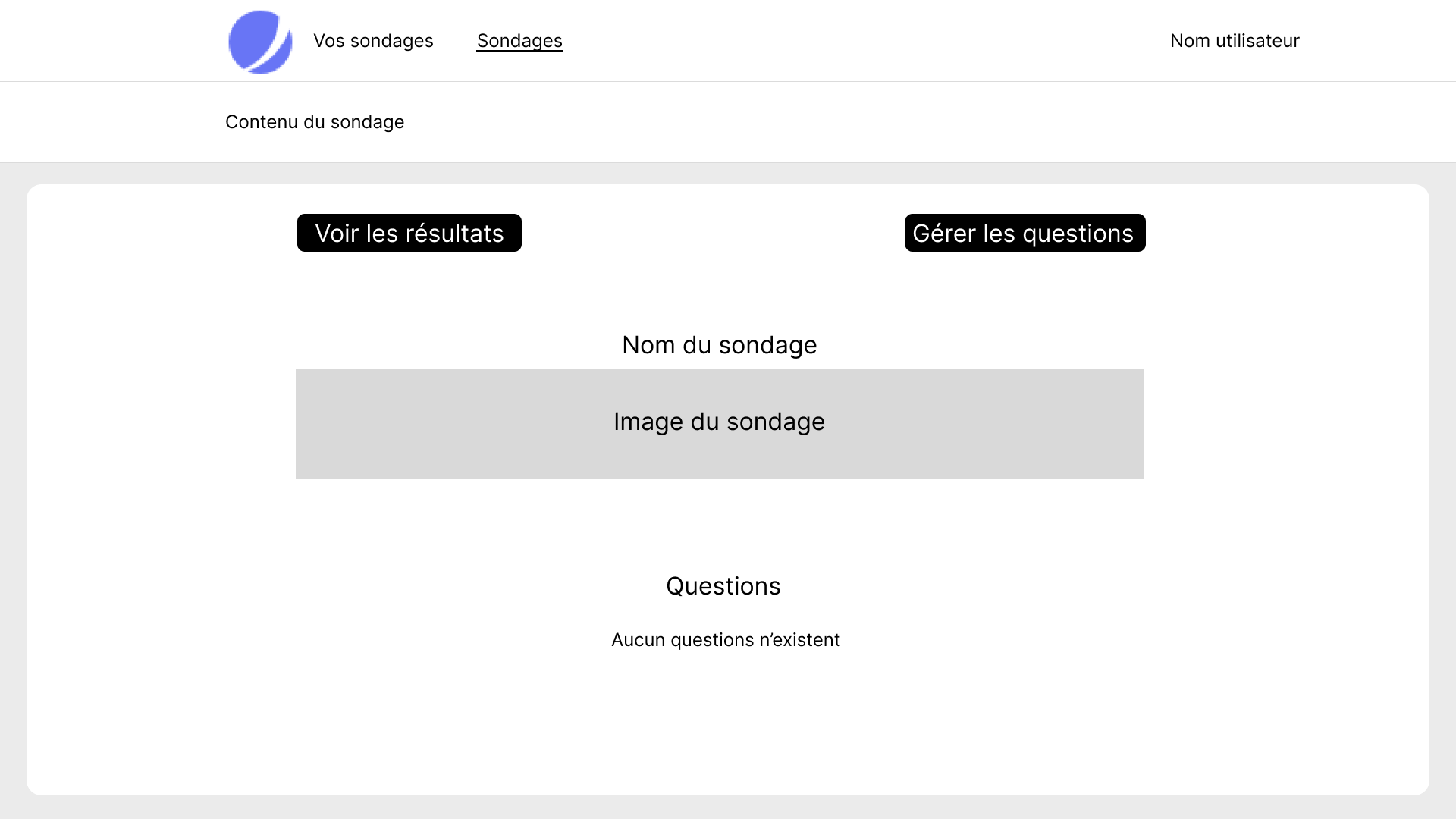
****

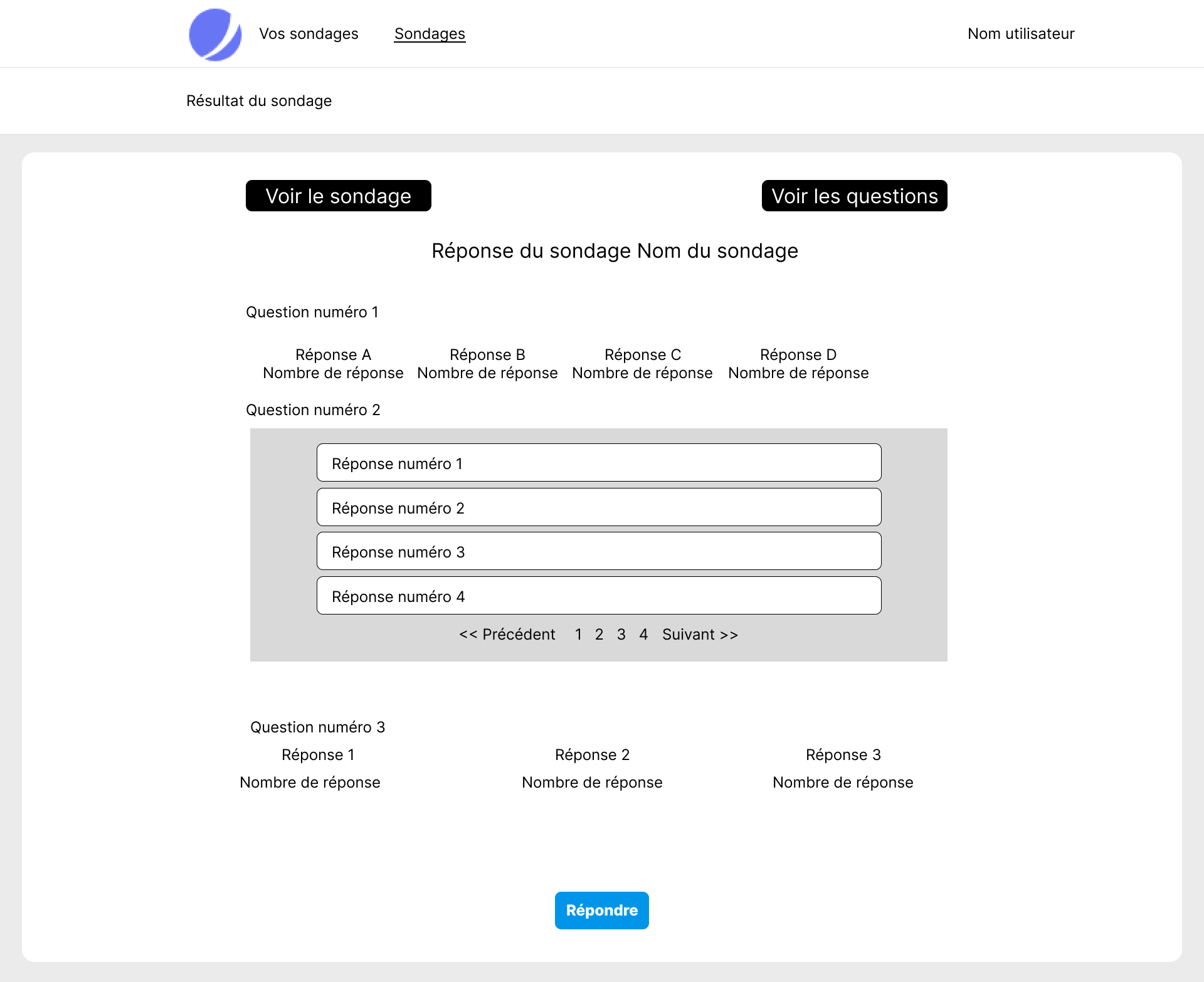
****

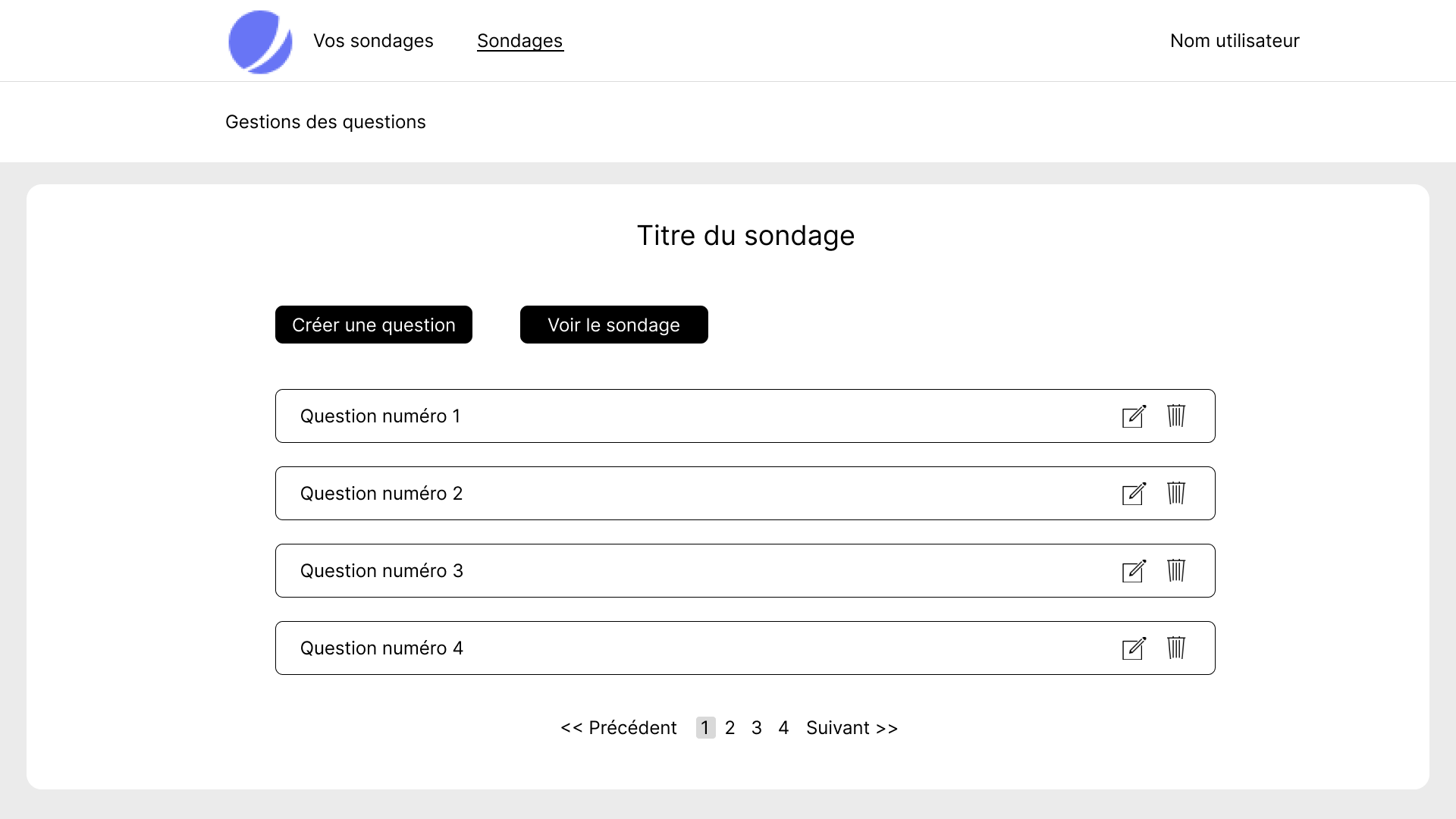
****

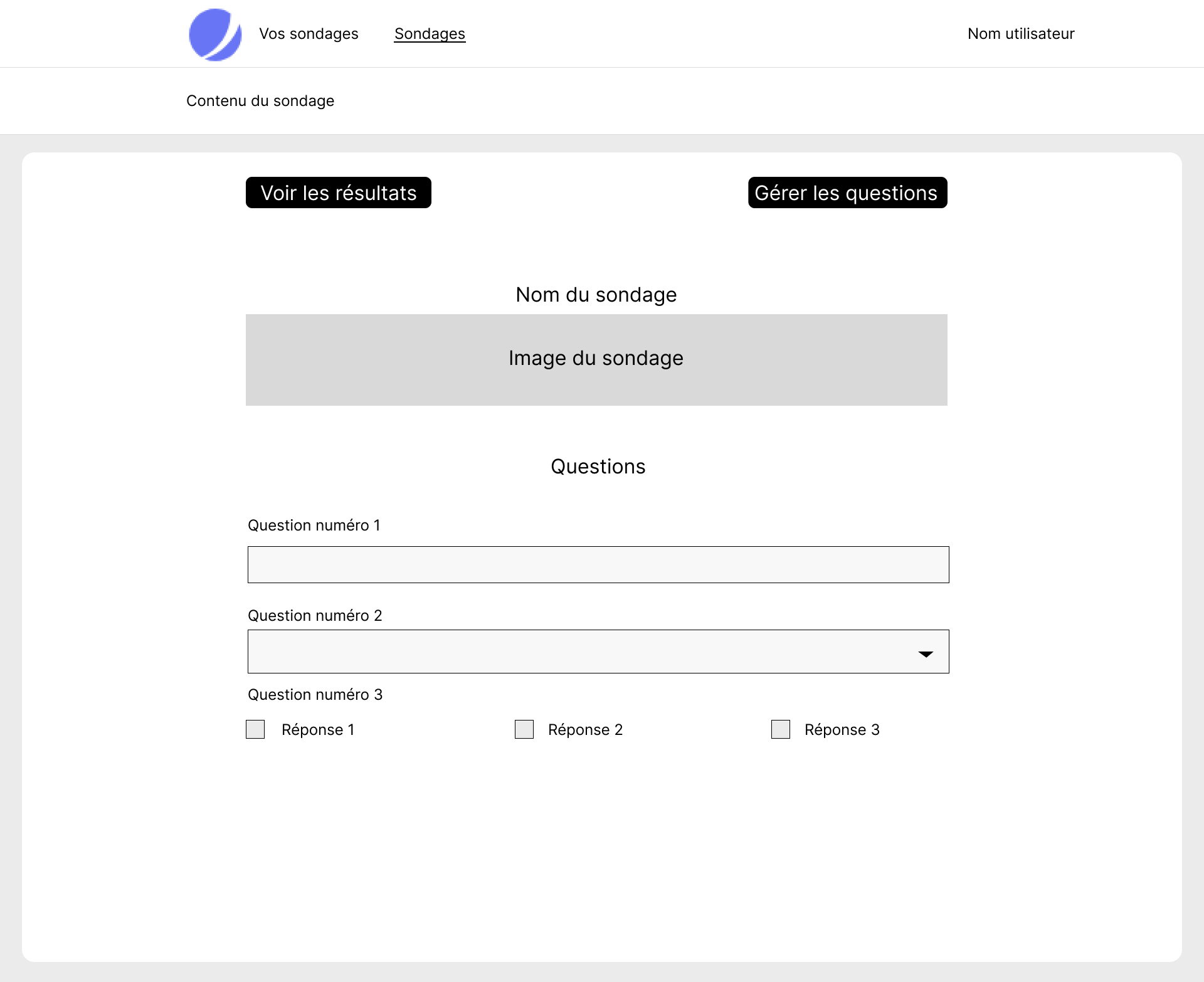
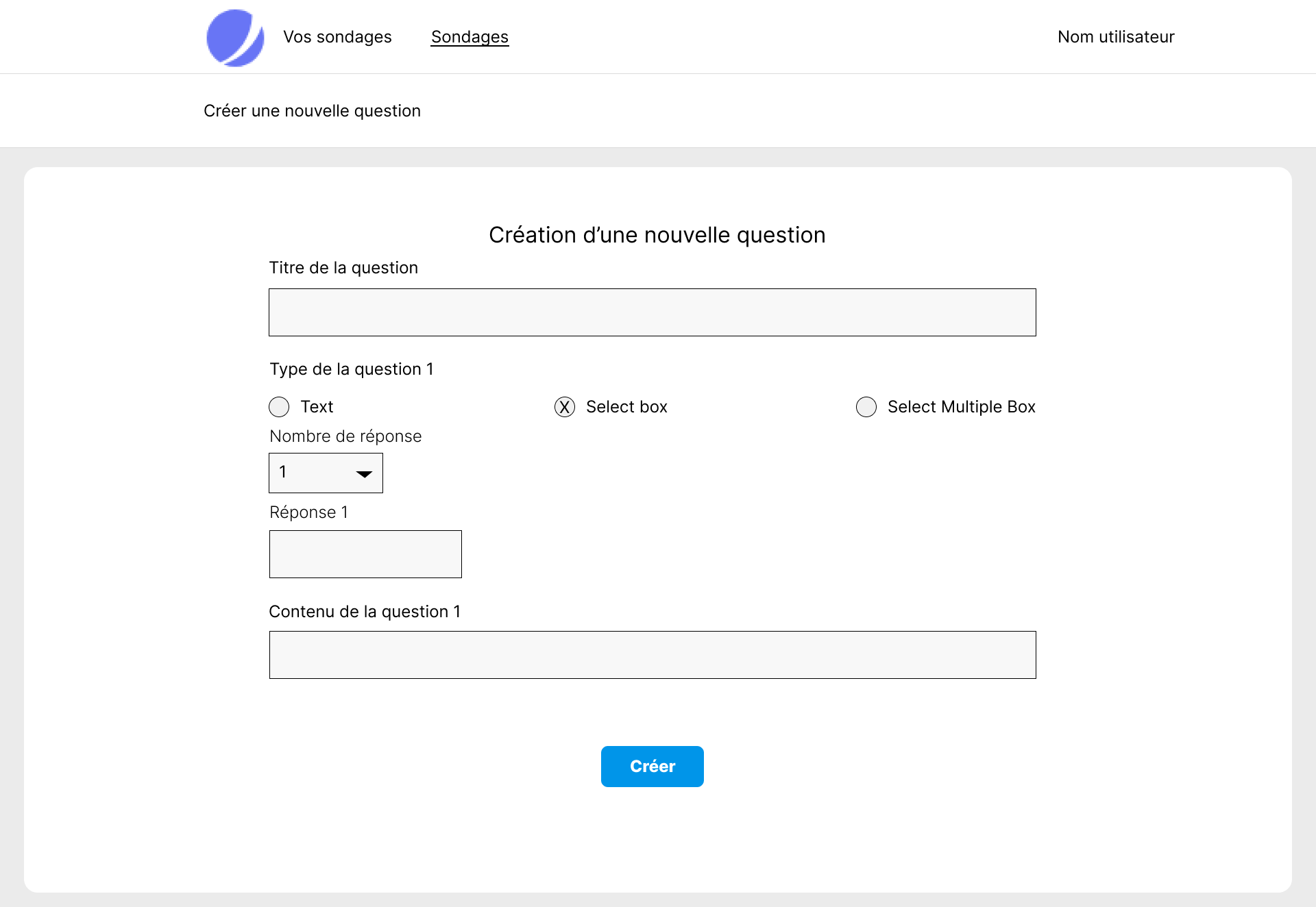
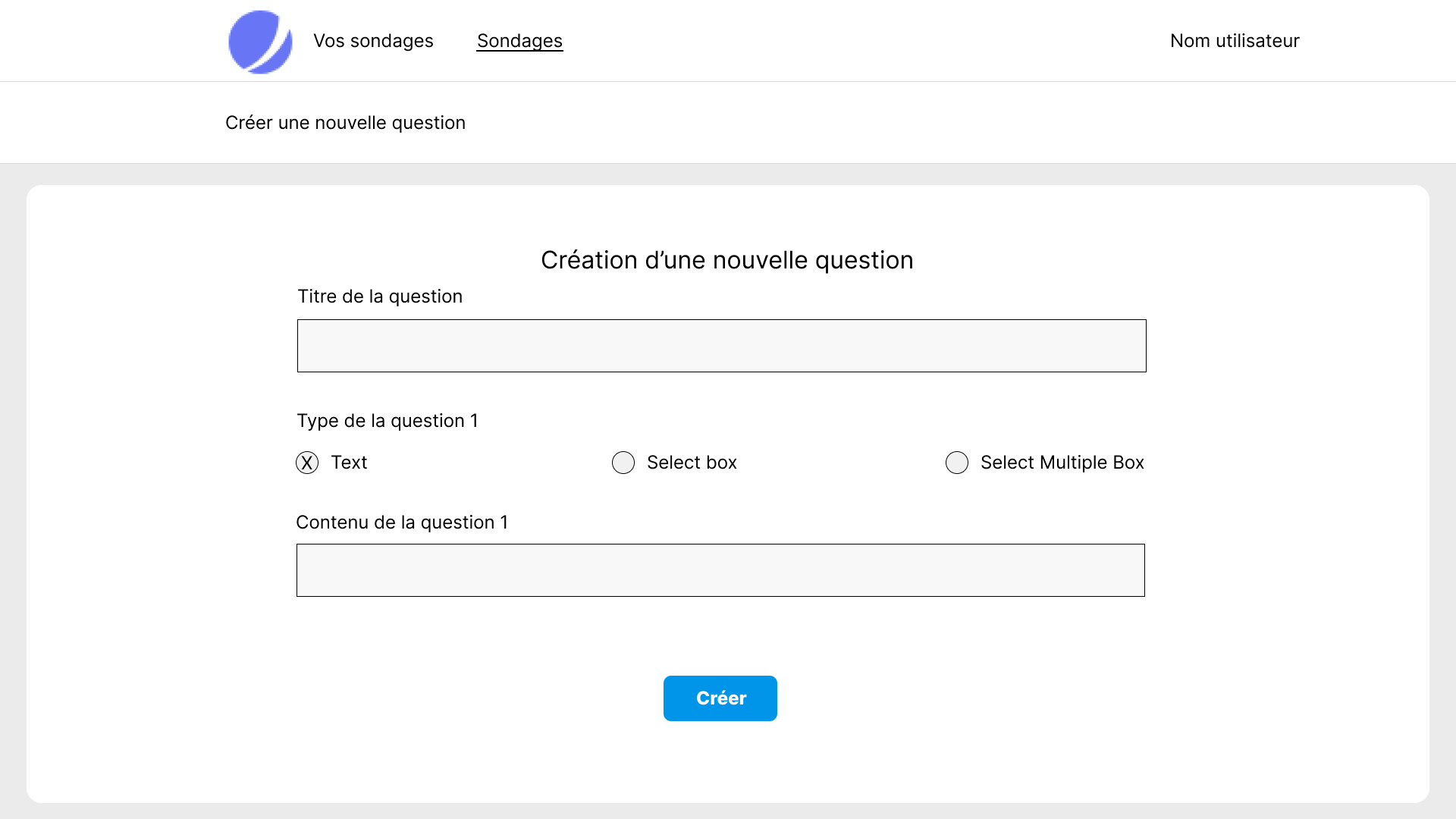
****

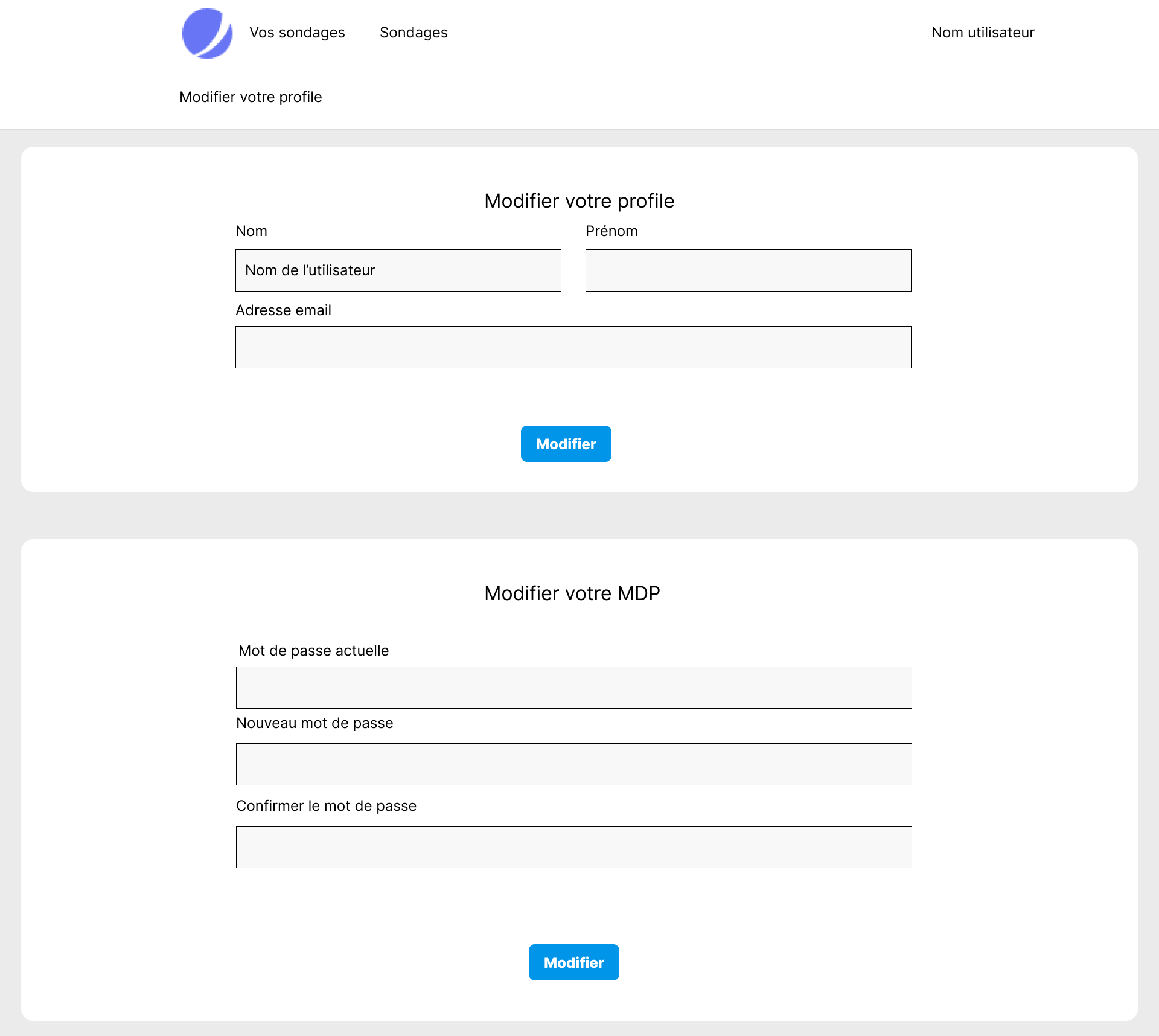
****

****

****

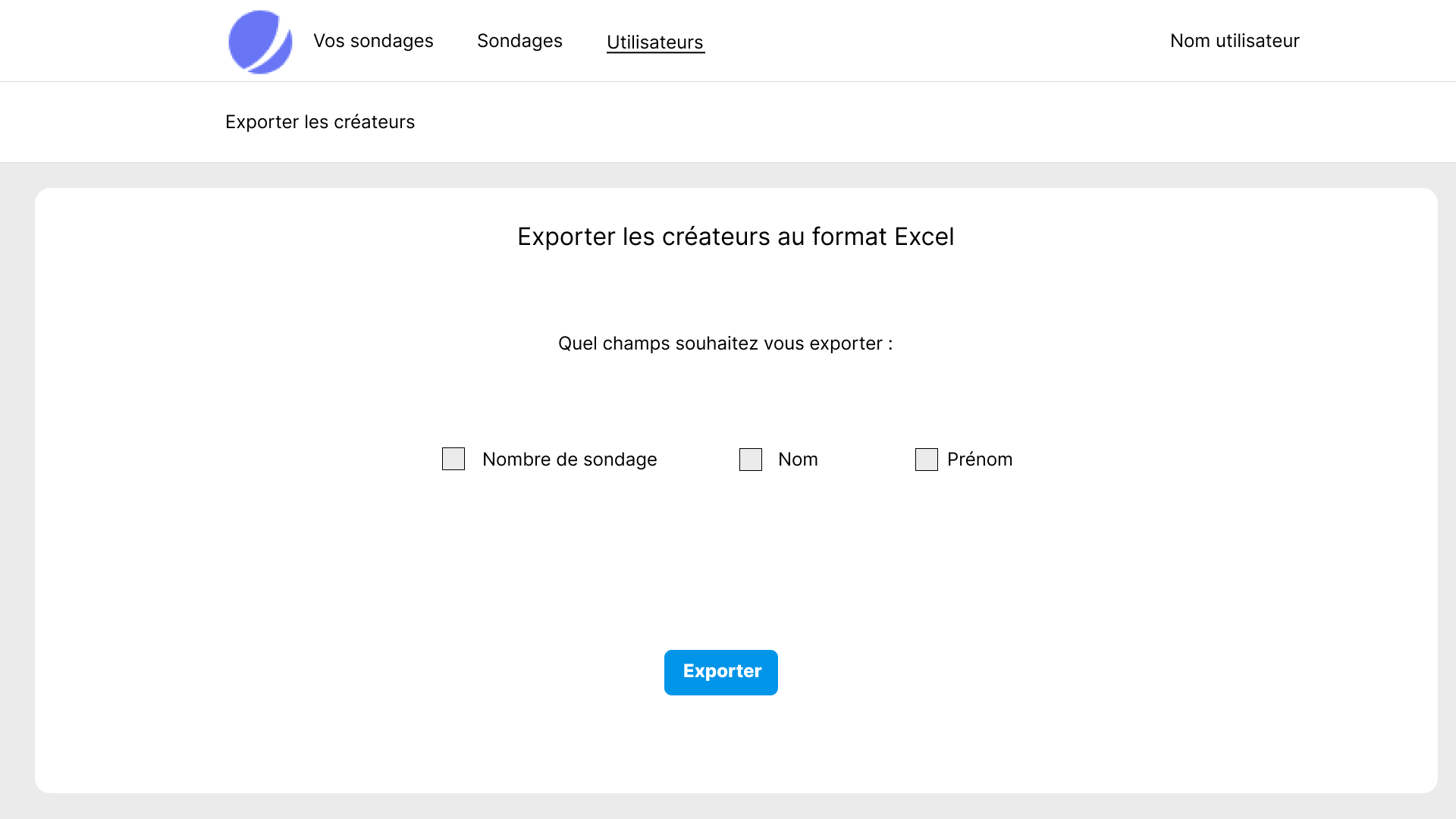
****

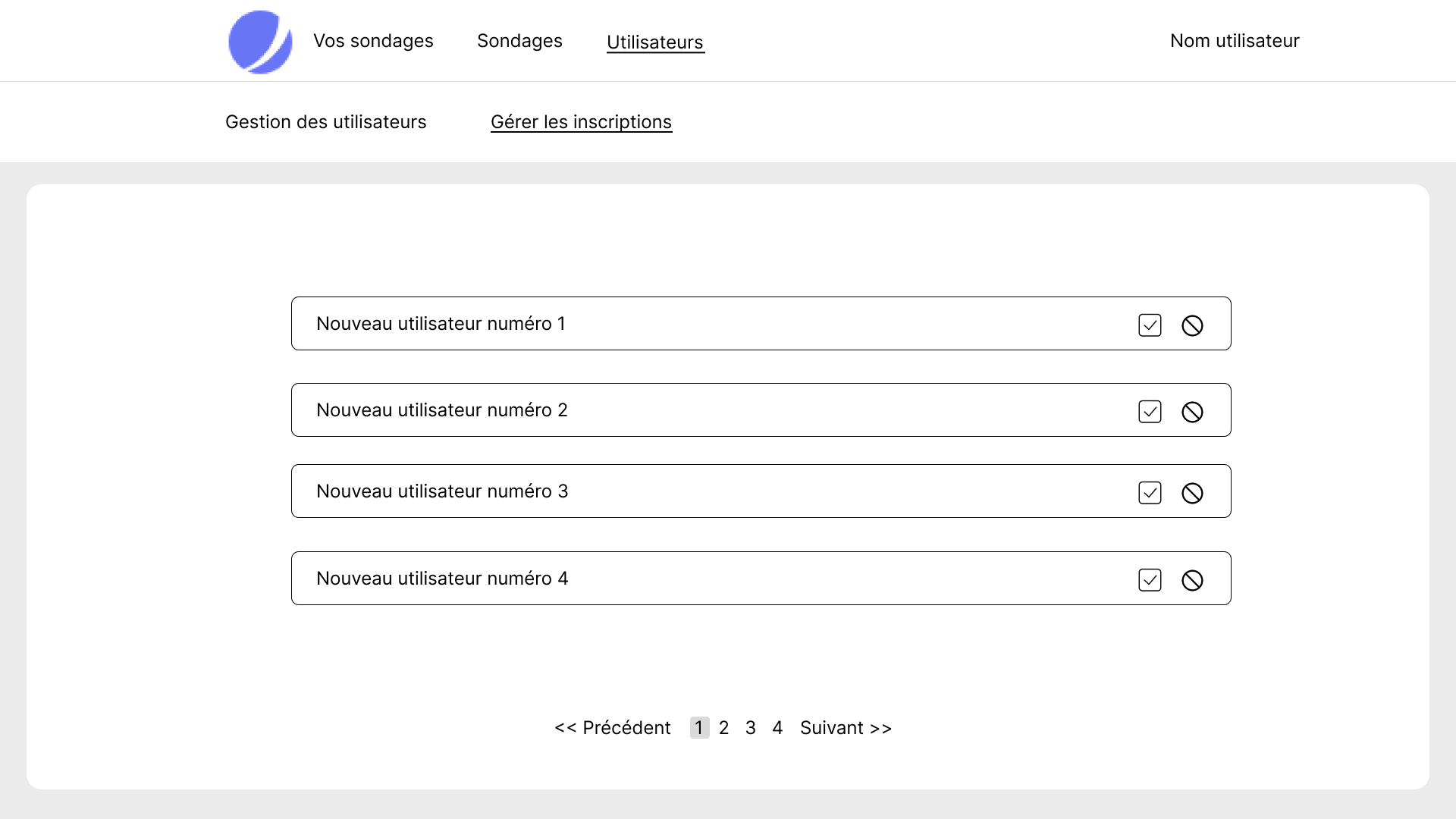
****

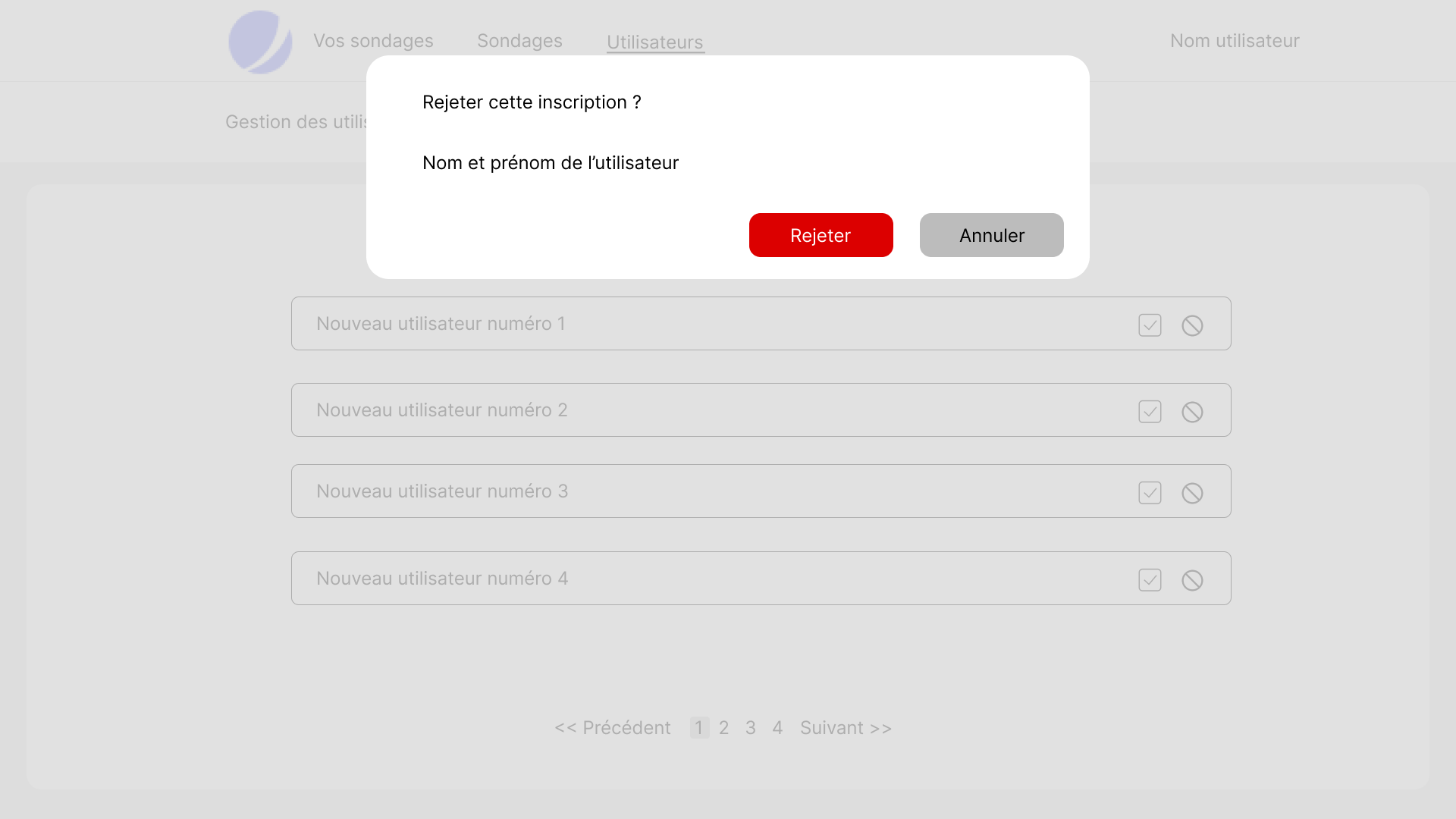
****

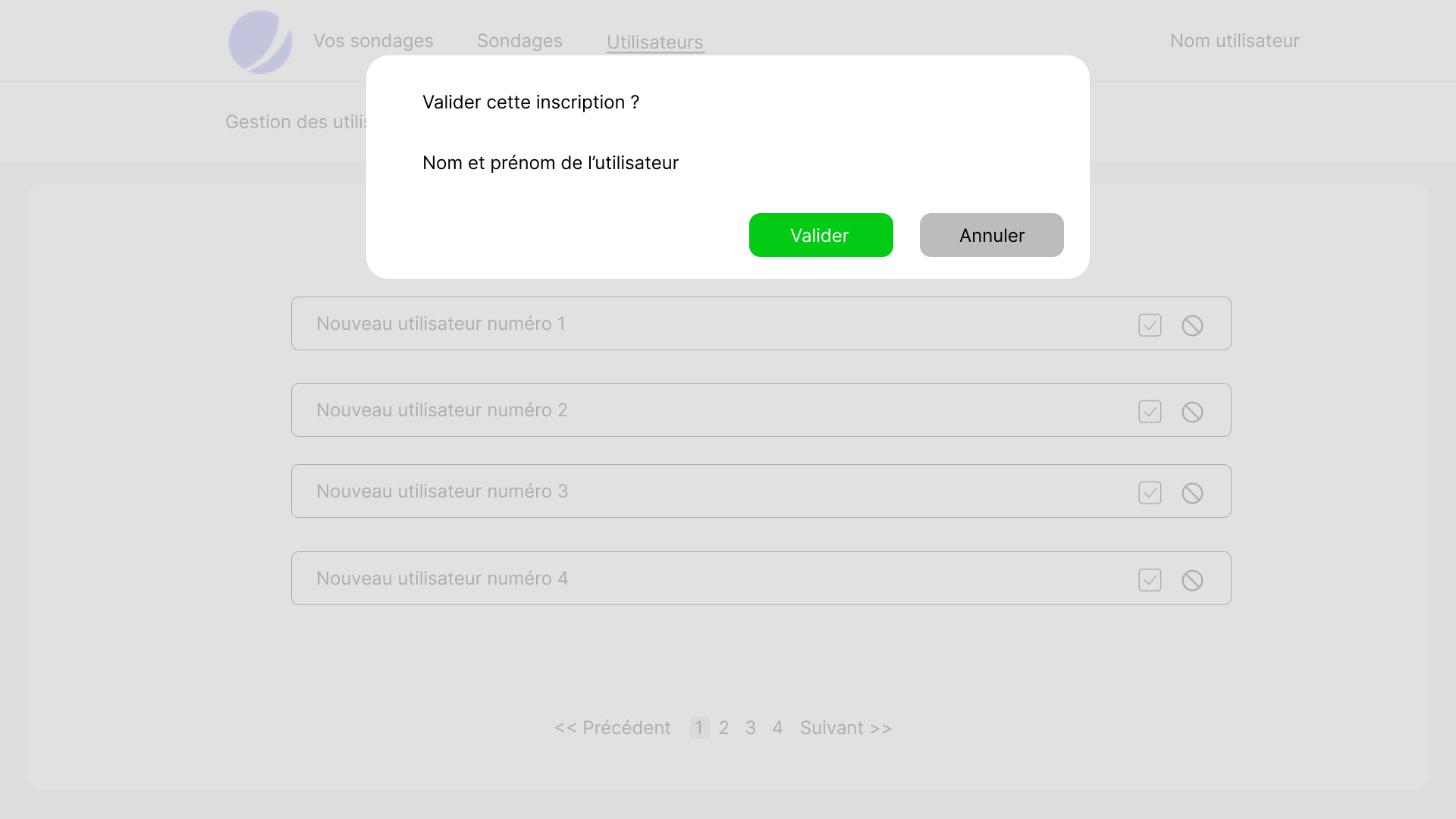
**Vue Admin**

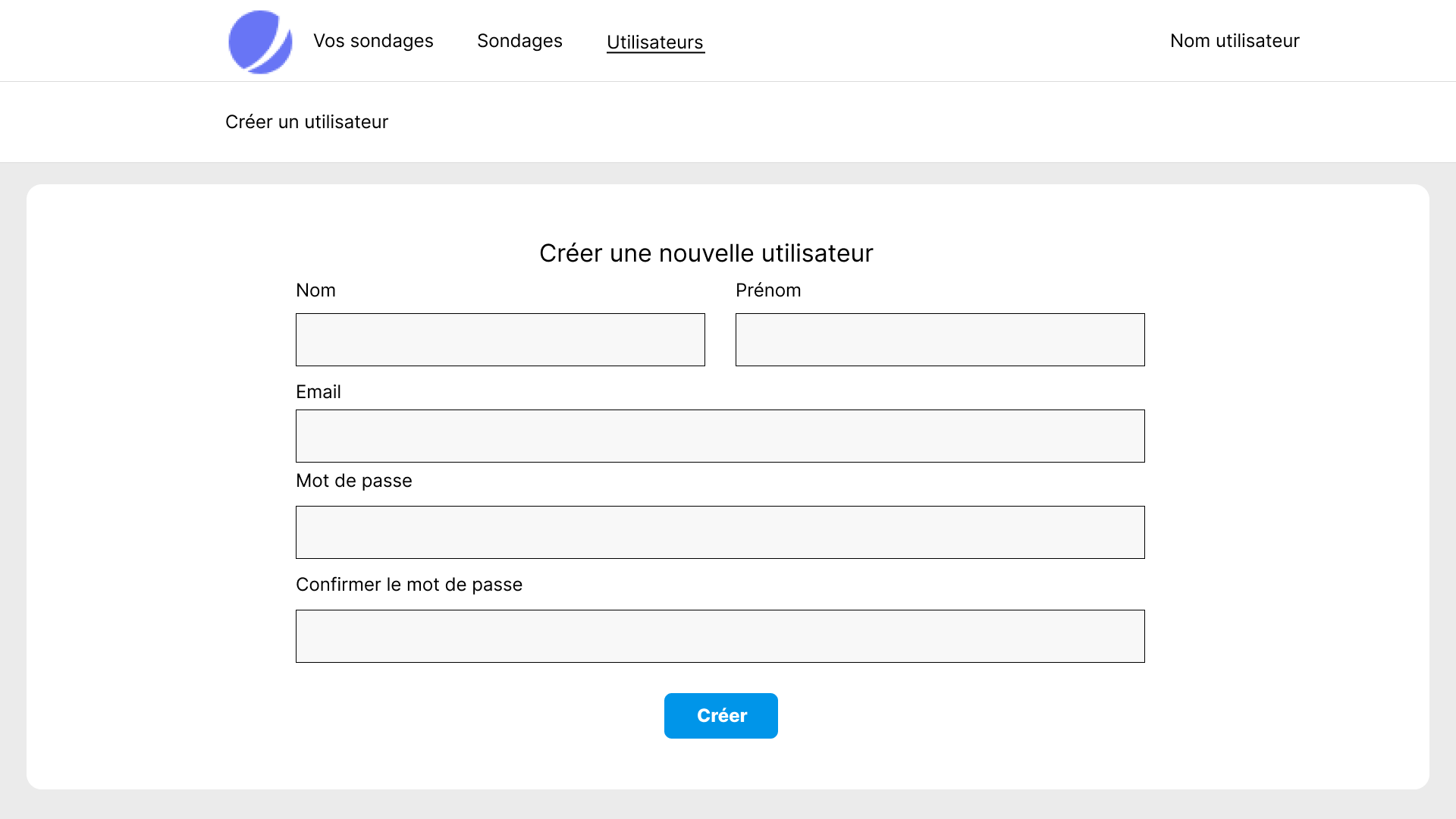


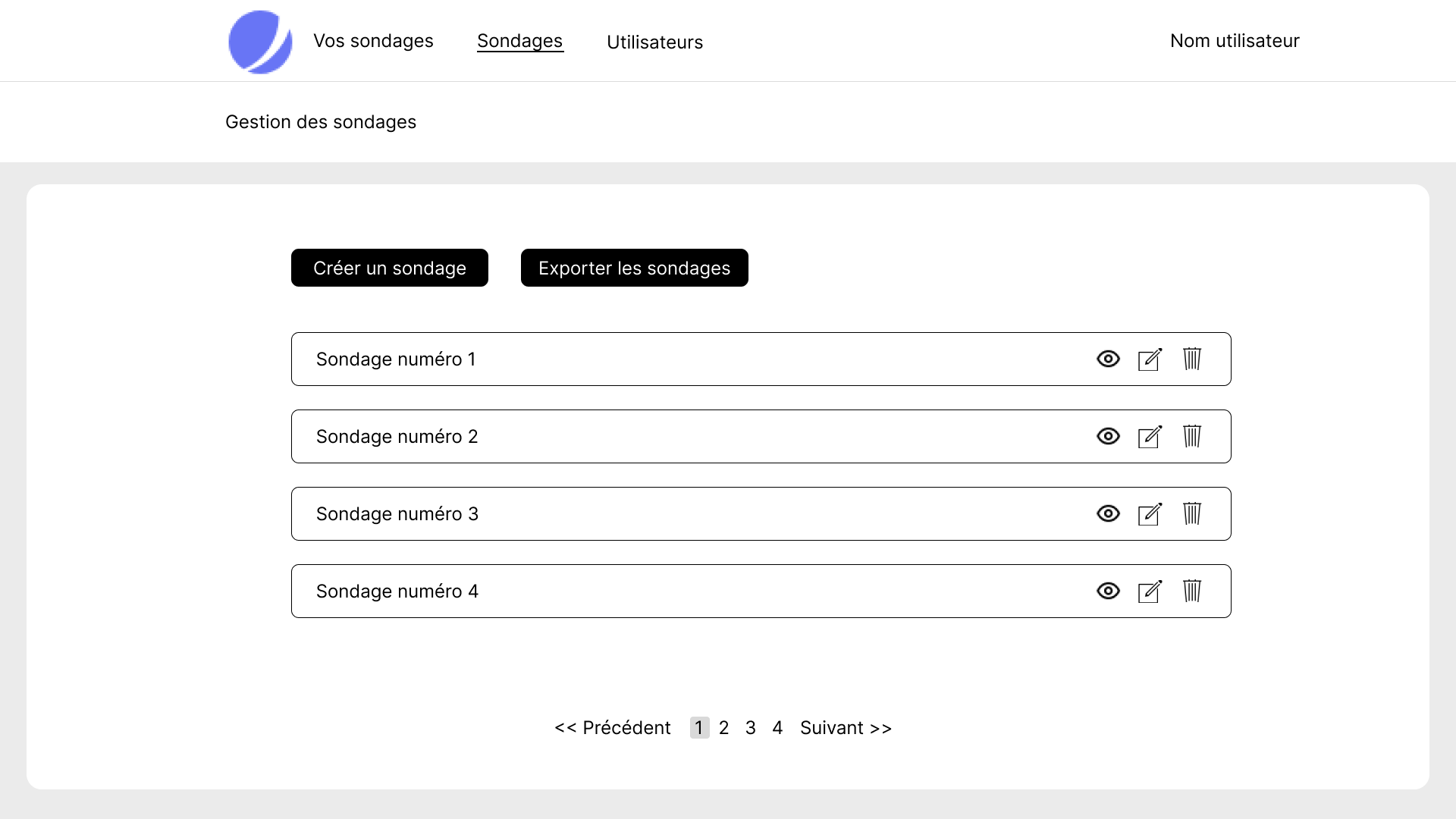
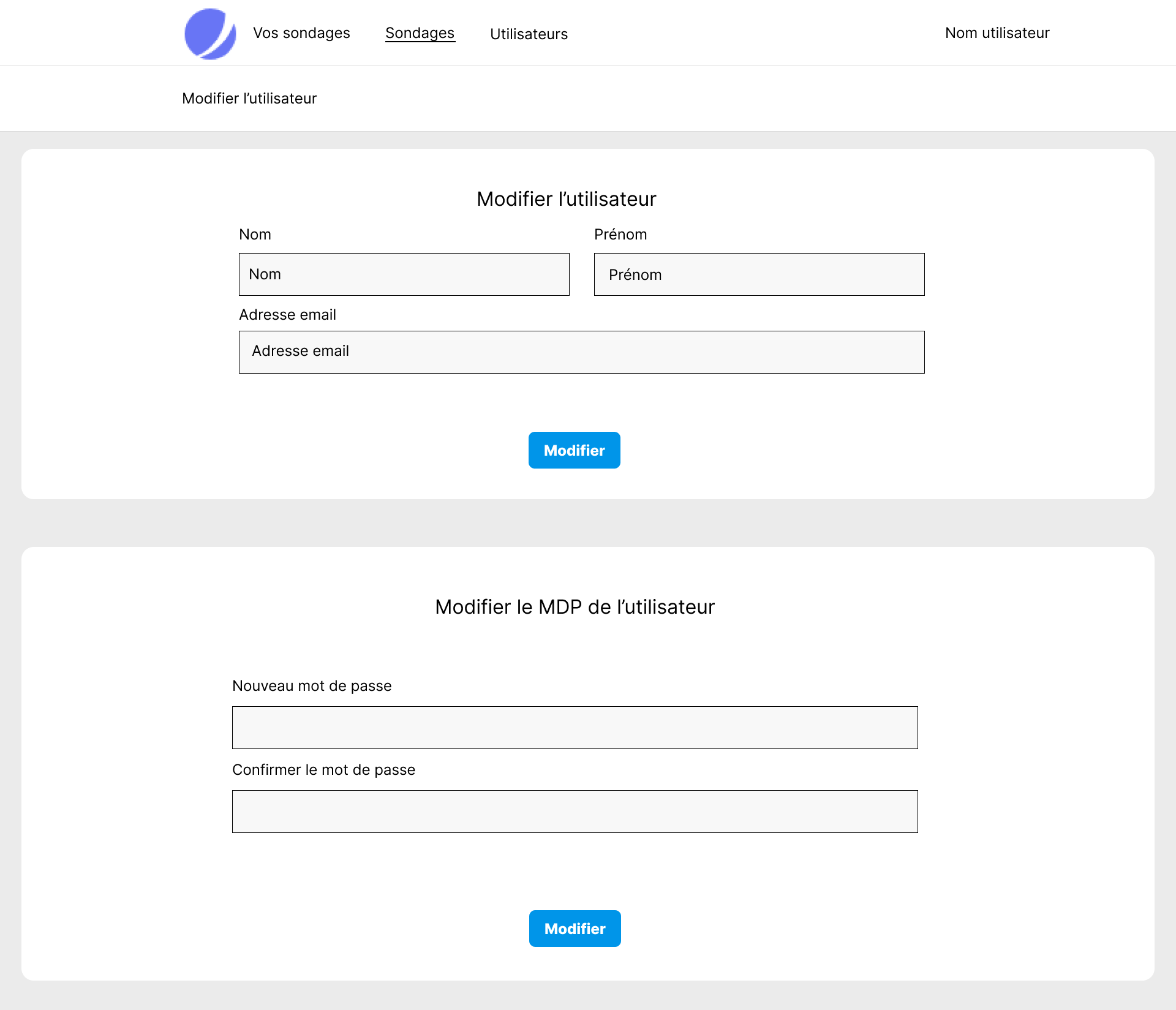


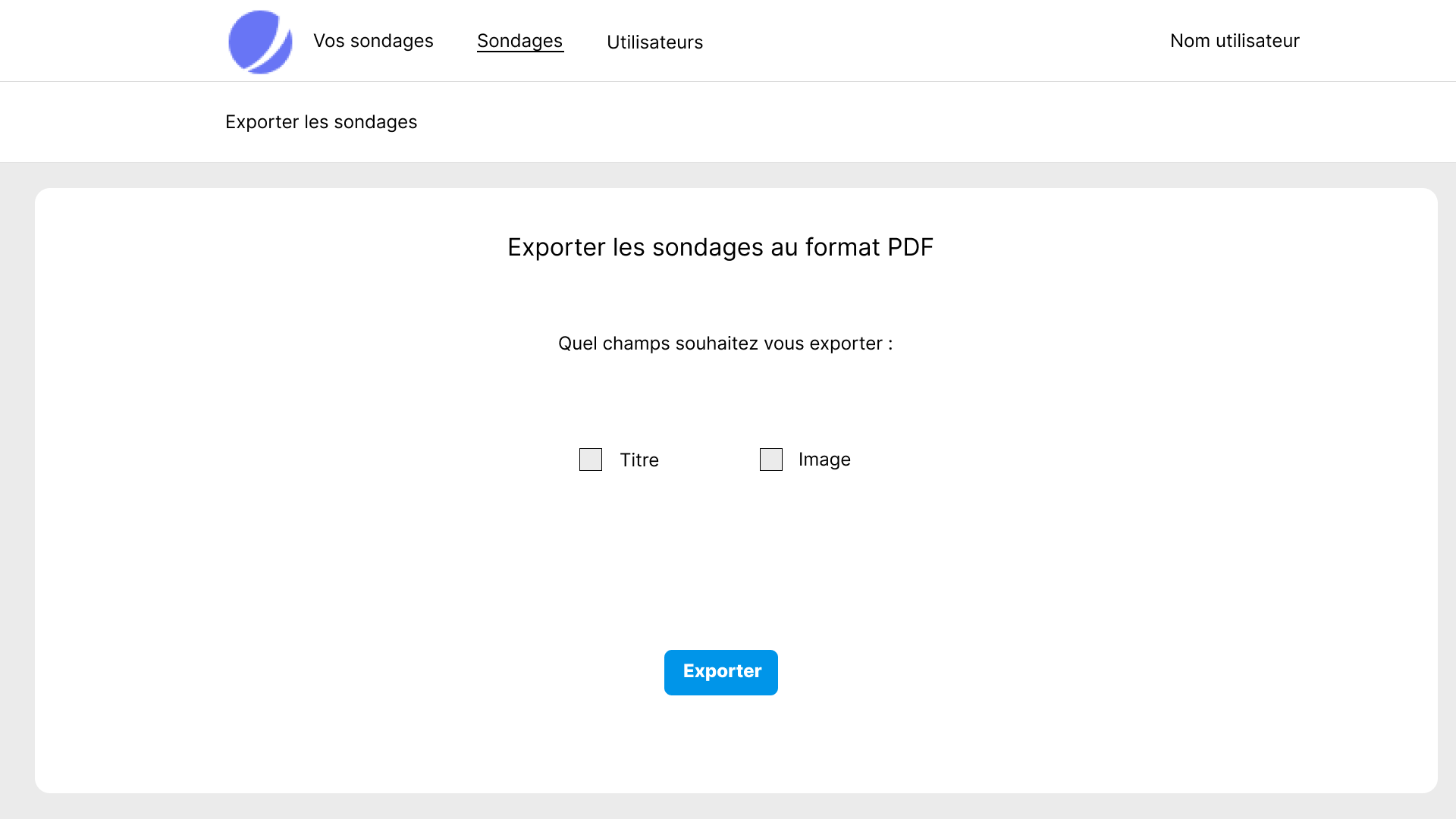


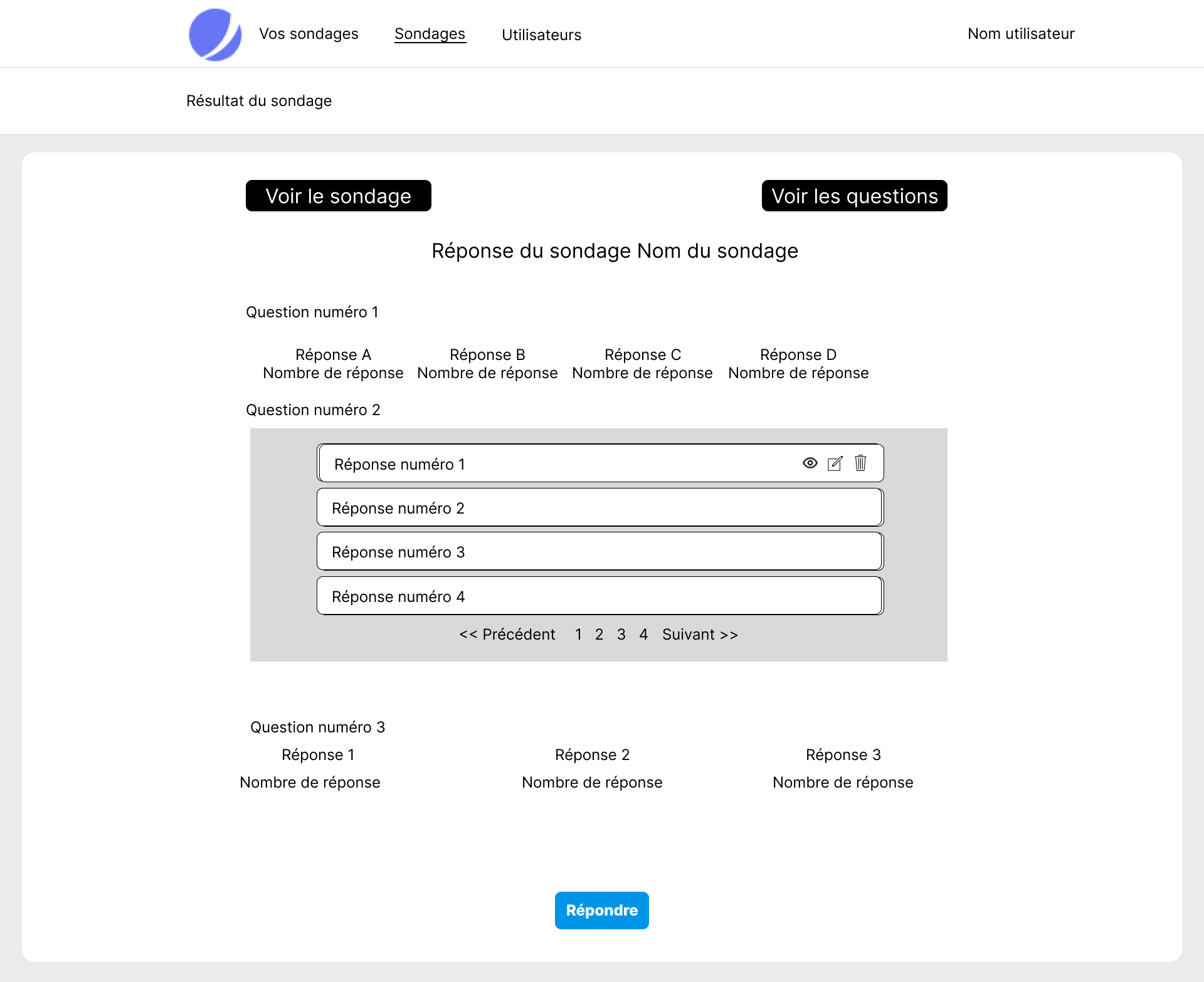


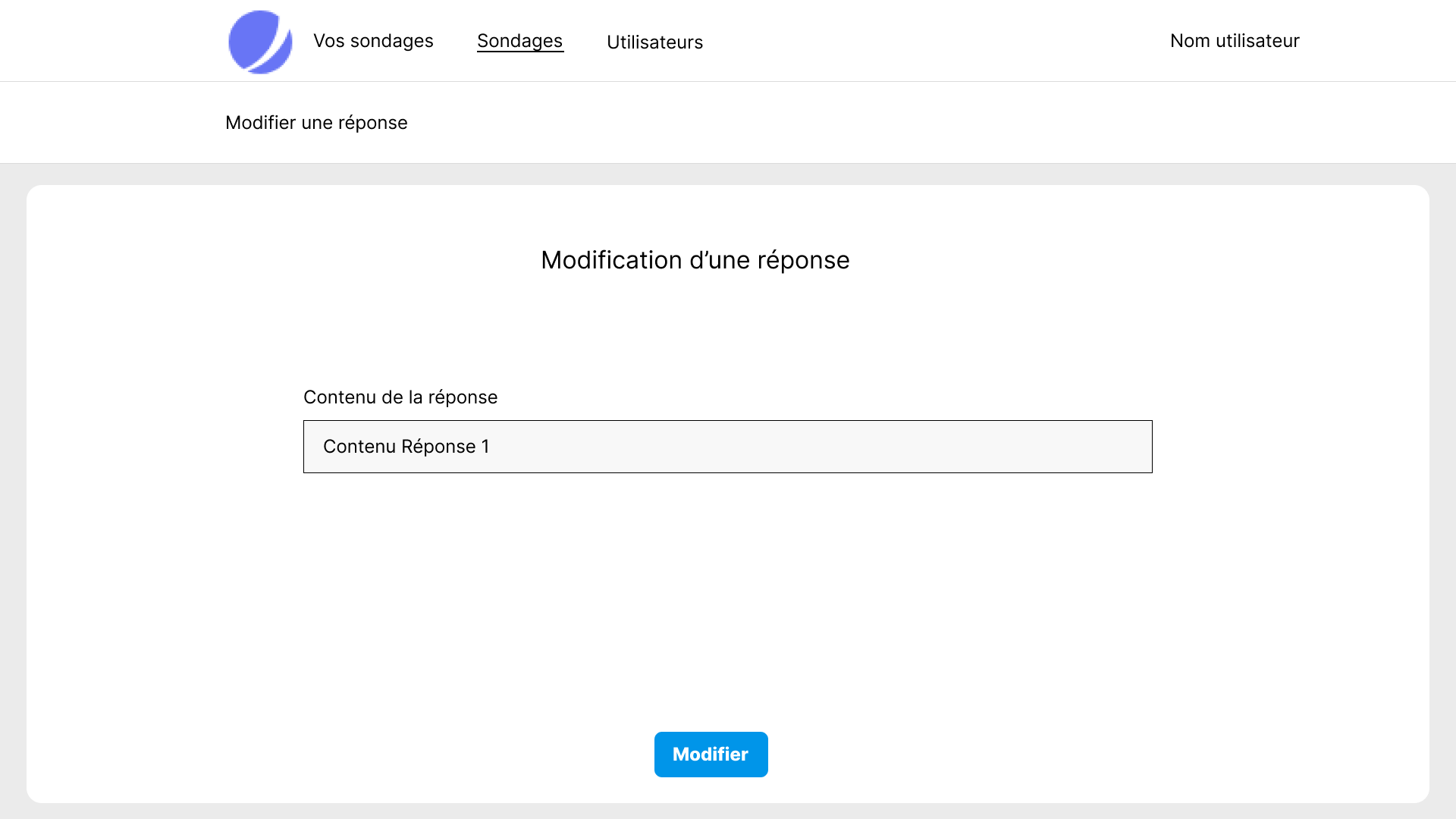




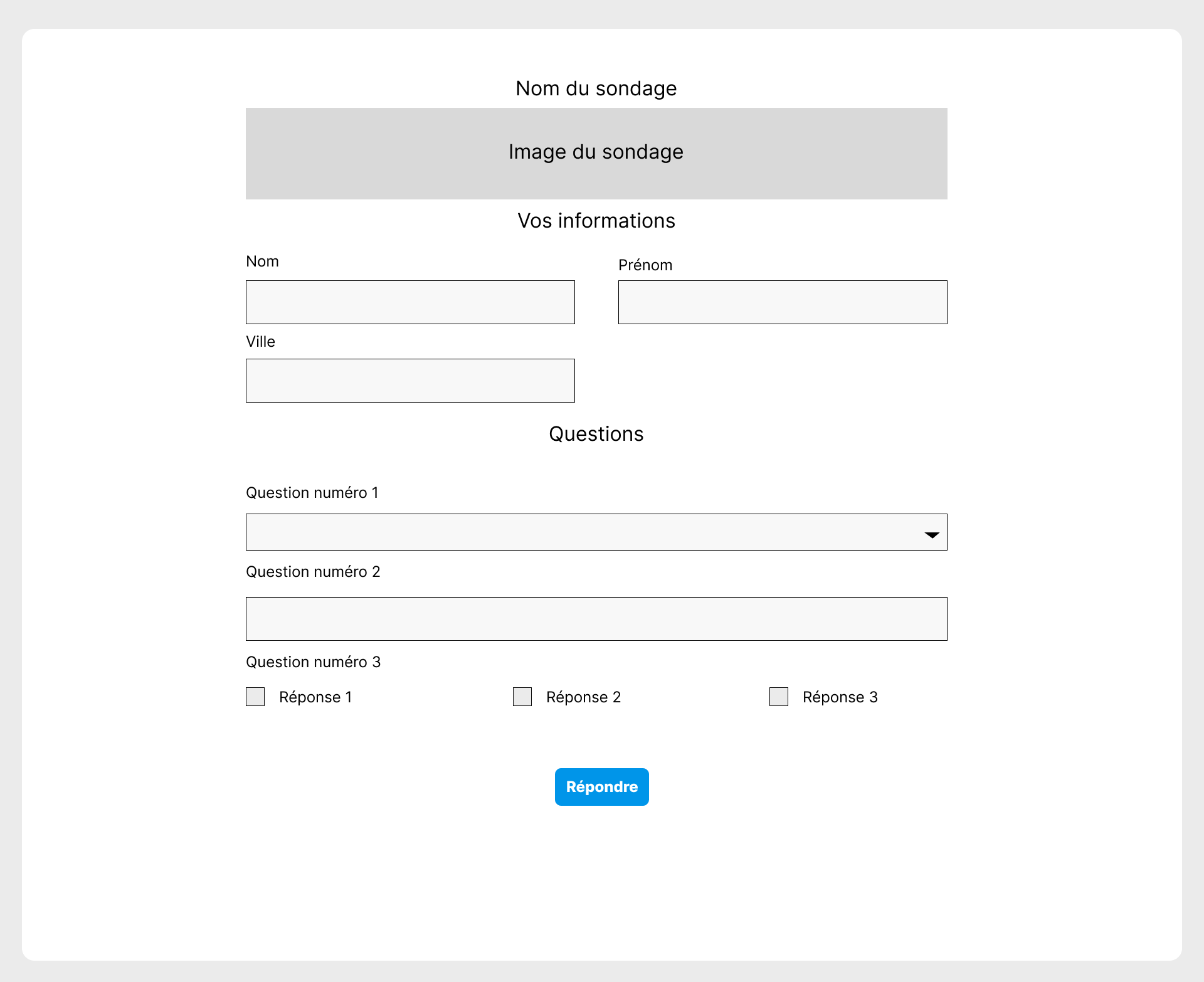






****

**Vue Répondant**

****

* 1. **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,