

**Projet Final : Bigscreen survey**

DM2-1

Ibrahim Fanna  
Novoa Y Rodriguez Miguel-Angel

1. **Analyse du client**

Notre client, Bigscreen, est une société qui développe une application VR.  
Cette application a pour but de permettre aux utilisateurs de regarder des films, émissions TV ou jouer à des jeux vidéo (seul ou à plusieurs) sur un écran virtuel.  
  
Bigscreen souhaite récolter des informations auprès de ses utilisateurs via un sondage afin de préparer la prochaine version de leur application. Ils nous ont donc chargés de réaliser le sondage ainsi que l’interface administrateur permettant aux admins de Bigscreen de se connecter afin de consulter les résultats.

1. **Choix technologiques**

* **Back-end :**

Concernant la partie back, le framework php laravel est imposé par le brief du projet. Aucune autre technologie back n’a été ajoutée car Laravel suffisait amplement pour développer toutes les fonctionnalités de l’Api.  
  
On a cependant utilisé des logiciels tels que Postman ou RapidApi afin de tester nos routes et vérifier que nos contrôleurs fonctionnent comme attendu.

* **Front-end :**

Concernant la partie front, nous avons opté pour le framework javascript React, car nous connaissions déjà assez bien ce framework, de plus il est assez flexible et on peut y associer plusieurs bibliothèques qui nous ont été utiles au cours du développement. Les bibliothèques que nous avons utilisées sont **react router dom** afin de gérer la navigation entre les différentes pages du site ou encore **Material UI** qui propose des composants correspondants à nos besoins.  
On a aussi utilisé le **bundler vite.js** afin d’optimiser les temps de chargement des composants, la **bibliothèque axios** afin de gérer plus facilement les requêtes vers l’api et enfin **chart.js** pour l’affichage des graphiques de la page d’accueil de l’interface admin.

1. **Adresse du dépot github**

Voici le lien de l’adresse Github sur lequel figure tous les dépots du projets: <https://github.com/devbigscreen/Projet_Final_Bigscreen>

1. **Déploiement du projet**

Voici les étapes qui permettent de déployer le projet

1) **Ouvrir un terminal** dans le dossier server et entrez les commandes suivantes dans l'ordre :

- npm install

- php artisan migrate

- php artisan db:seed --class=AdminSeeder

- php artisan db:seed --class=QuestionsSeeder

- php artisan db:seed --class=UrlSeeder

- php artisan db:seed --class=UserAnswersSeeder

- php artisan serve

2) **Ouvrir un autre terminal** dans le dossier admin et entrer la commande :

- npm install

- npm run dev

3) **Liens de l'interface :**

- interface admin (page login) : **http://localhost:5173/**

**\* email** : admin@bigscreen.com

**\* mot de passe** : password

- interface utilisateur (page sondage) : [**http://localhost:5173/survey**](http://localhost:5173/survey)

Il s'agit des réponses de l'utilisateur : "test1",

- page réponses de l'utilisateur: [**http://localhost:5173/user/answers/test1**](http://localhost:5173/user/answers/test1)

les seeders ont alimentés la base de données avec **les réponses de 5 utilisateurs "test1", "test2", "test3", "test4" et "test5".**

Si vous voulez consulter les réponses des autres utilisateurs tests, modifiez le "test1" de l'url par celui que vous voulez parmi les 5.

Si vous déployez le front sur un autre port que 5173, changez ce port avec le votre dans les urls.

1. **Méthode de travail**

En ce qui concerne notre méthode de travail, nous avons scindé les tâches en deux.  
Pour la création de nos branches et commits nous avons indiqués en début de titre ou message : **“Feature”** dans le cas où il s’agissait d’une nouvelle fonctionnalité à développer, **“Fix”** dans le cas où il s’agissait de la correction d’un bug ou encore **“Feature + Fix”** dans le cas où on faisait les deux.

Après avoir terminé nos tâches en cours, on créait une pull request et on envoyait un message à notre binôme. Lorsqu’on avait le temps, on allait analyser sa pull request, afin d’émettre des retours dans le cas où on observait des erreurs, ou dans le cas où tout semblait **“ok”**, on envoyait un **“go”** pour indiquer que le merge pouvait être effectué.  
  
Dernière chose, on **s’organisait pour** ne pas travailler sur les mêmes fichiers en même temps et ainsi **éviter au maximum les conflits**.

1. **Outils utilisés**

* VS Code
* Mamp
* Postman
* RapidApi
* Adobe XD
* Discord
* Google Meets
* Google Docs
* dbDiagram.io

1. **Evalutation du temps de travail**

| **Feature** | **Temps de travail** |
| --- | --- |
| Migrations, models | ½ jour |
| Admin controller et routes associées | ½ jour |
| Questions controller et routes associées | ½ jour |
| User controller et routes associées | 1 jour |
| Affichage des graphiques sur la page home | 2 jours |
| Composant list | ½ jour |
| Views questionsList, allUsersAnswers et  userAnswers (liées au composant list) | 2 jours |
| Composant CheckAuth lié à la page View | ½ jour |
| View survey et composant Survey | 2jours |

1. **Liste fonctionnelle**

**Back :**

* Login de l’admin
* Ajout d’un administrateur à la base de données
* Suppression d’un administrateur de la base de données
* Ajout d’une question à la base de données
* Récupération de toutes les questions de la base de données
* Récupération d’une question voulue de la base de données
* Mise à jour d’une question de la base de données
* Suppression d’une question de la base de données
* Création d’un utilisateur et ajout de ses réponses dans la base de données
* Récupérer toutes les réponses de tous les utilisateurs
* Récupérer toutes les réponses d’un utilisateur
* Récupérer l’url d’un utilisateur
* Suppression d’un utilisateur de la base de données
* Vérification si l’email de l’utilisateur est déjà présent en base

**Front :**

* Requêtes pour appeler le back
* Login/Logout de l’admin
* Menu burger
* Affichage des graphiques de données des réponses
* Affichage liste des questions
* Affichage listes de toutes les réponses de tous les utilisateurs
* Page sondage et envoi des réponses de l’utilisateur
* Page liste des réponses de l’utilisateur

1. **Recettage**

**Back :**

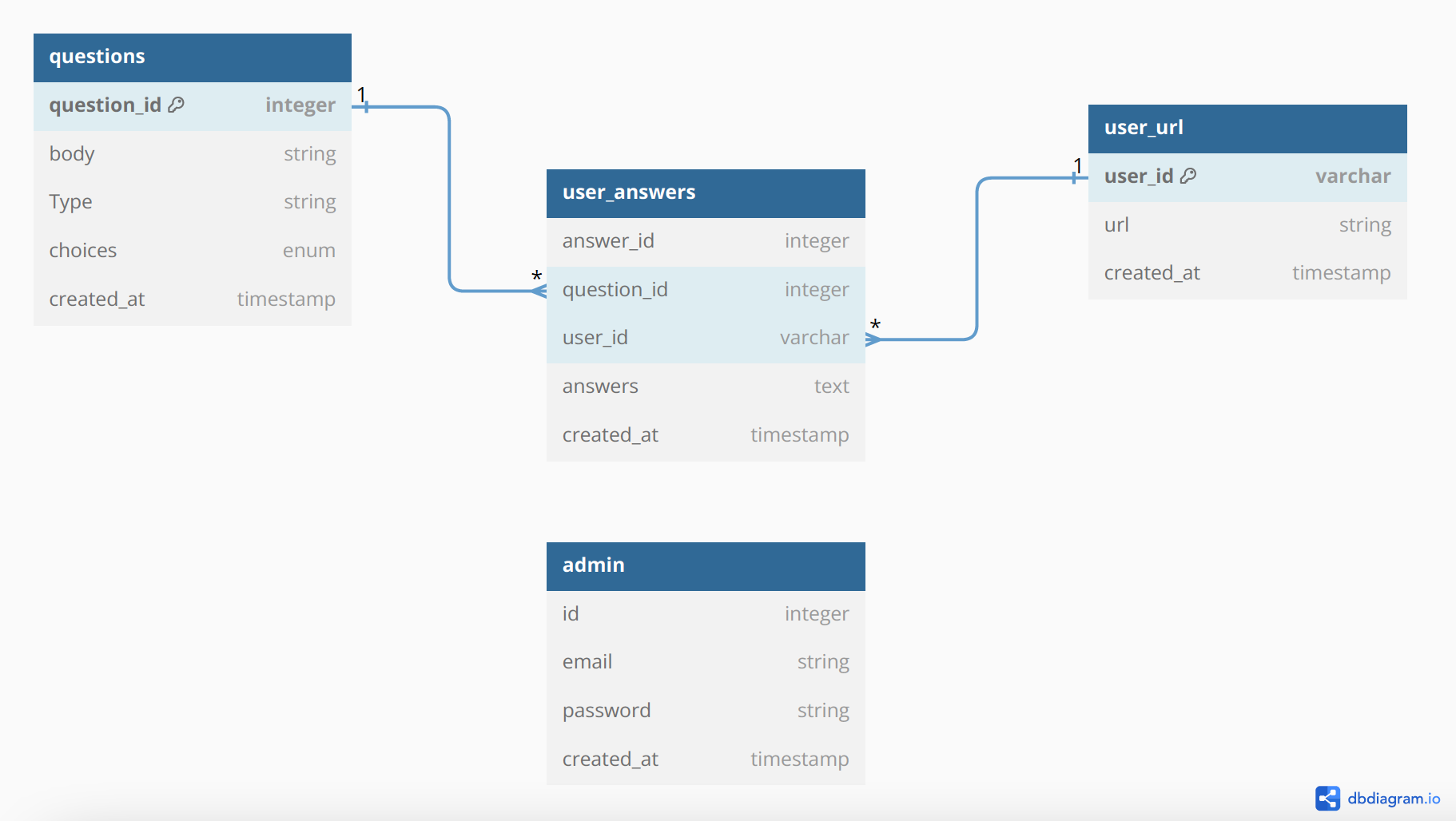
| **Fonctionnalité** | **Status** | **Description** |
| --- | --- | --- |
| Login de l’admin | **Fonctionnel** | **ok** |
| Ajout d’un administrateur à la base de données | **Non utilisé** | **Fonctionnel mais non utilisé car non utile à notre interface** |
| Suppression d’un administrateur de la base de données | **Non utilisé** | **Fonctionnel mais non utilisé car non utile à notre interface** |
| Ajout d’une question à la base de données | **Non utilisé** | **Fonctionnel mais non utilisé car non utile à notre interface** |
| Récupération de toutes les questions de la base de données | **Fonctionnel** | **ok** |
| Récupération d’une question voulue de la base de données | **Non utilisé** | **Fonctionnel mais non utilisé car non utile à notre interface** |
| Mise à jour d’une question de la base de données | **Non testé** | **Non testé mais non utile à notre interface** |
| Suppression d’une question de la base de données | **Non utilisé** | **Fonctionnel mais non utilisé car non utile à notre interface** |
| Création d’un utilisateur et ajout de ses réponses dans la base de données | **Fonctionnel** | **ok** |
| Récupérer toutes les réponses de tous les utilisateurs | **Fonctionnel** | **ok** |
| Récupérer toutes les réponses d’un utilisateur | **Fonctionnel** | **ok** |
| Récupérer l’url d’un utilisateur | **Non utilisé** | **Fonctionnel mais non utile au final car url récupérer directement dans la réponse lors de la création de l’utilisateur** |
| Suppression d’un utilisateur de la base de données | **Non utilisé** | **Fonctionnel mais non utilisé car non utile à notre interface** |
| Vérification si l’email de l’utilisateur est déjà présent en base | **Non utilisé** | **Fonctionnel mais non implémenté en front par manque de temps** |

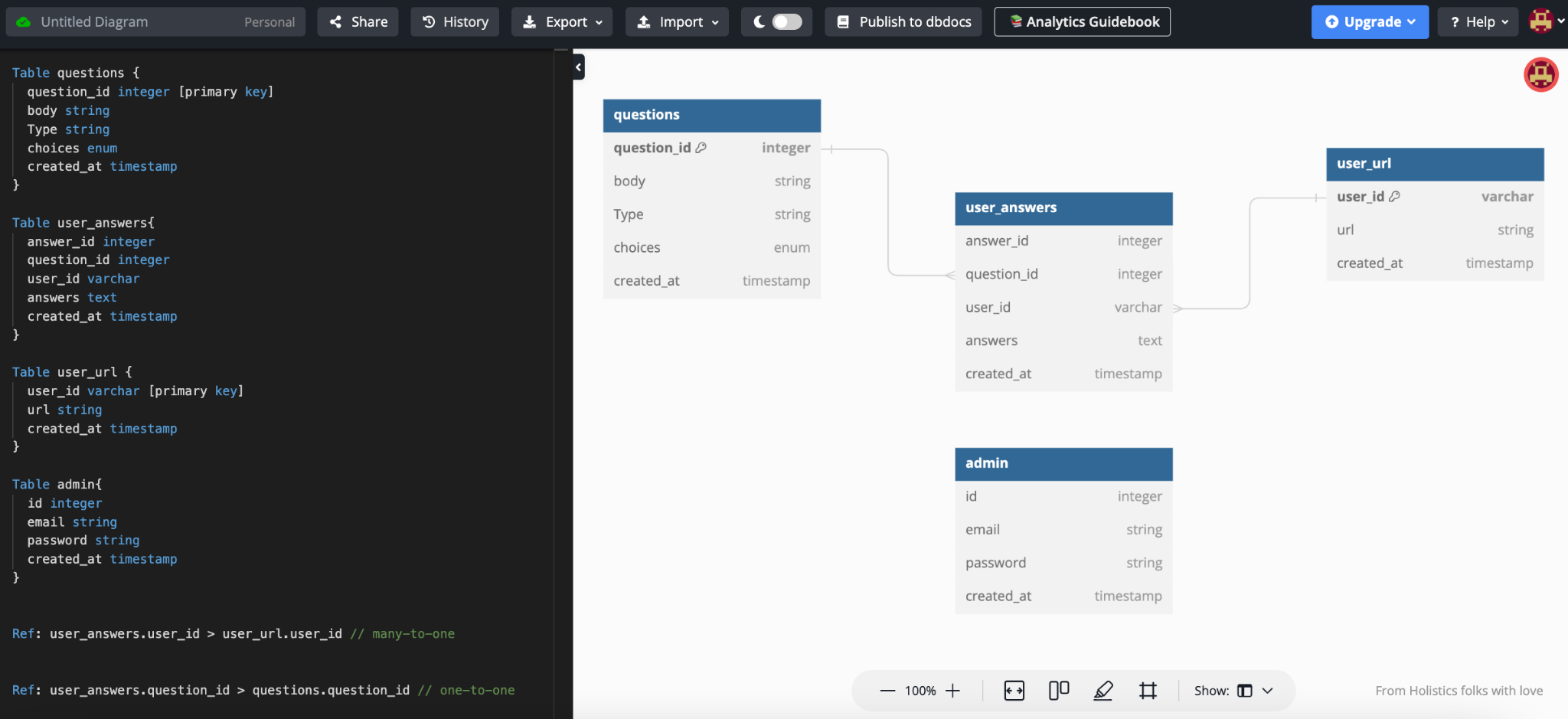
**Front :**

| **Fonctionnalité** | **Status** | **Description** |
| --- | --- | --- |
| Requêtes pour appeler le back | **Fonctionnel** | **ok** |
| Login/Logout de l’admin | **Fonctionnel** | **ok** |
| Menu burger | **Fonctionnel** | **ok** |
| Affichage des graphiques de données des réponses | **Limitation** | **Fonctionnel mais chargement long car formatage des données pour correspondre aux graphiques effectué en front, il aurait été plus judicieux de le faire en back.** |
| Affichage liste des questions | **Fonctionnel** | **ok** |
| Affichage listes de toutes les réponses de tous les utilisateurs | **Fonctionnel** | **ok** |
| Page sondage et envoi des réponses de l’utilisateur | **Fonctionnel** | **ok** |
| Page liste des réponses de l’utilisateur | **Fonctionnel** | **ok** |

1. **Diagramme de la base de données**

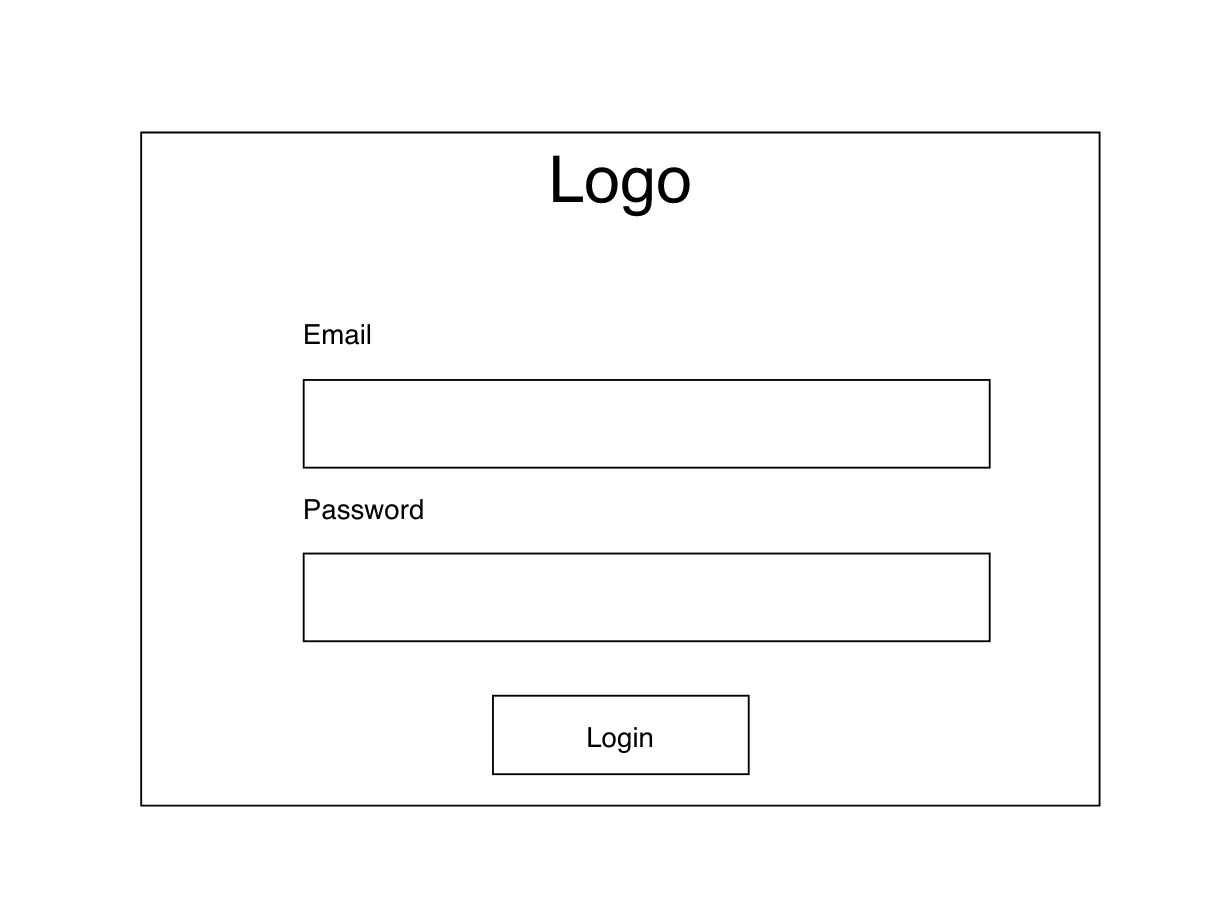
Nous avons opté pour le logiciel dbDiagram.io pour notre schéma de base de données sur le logiciel.



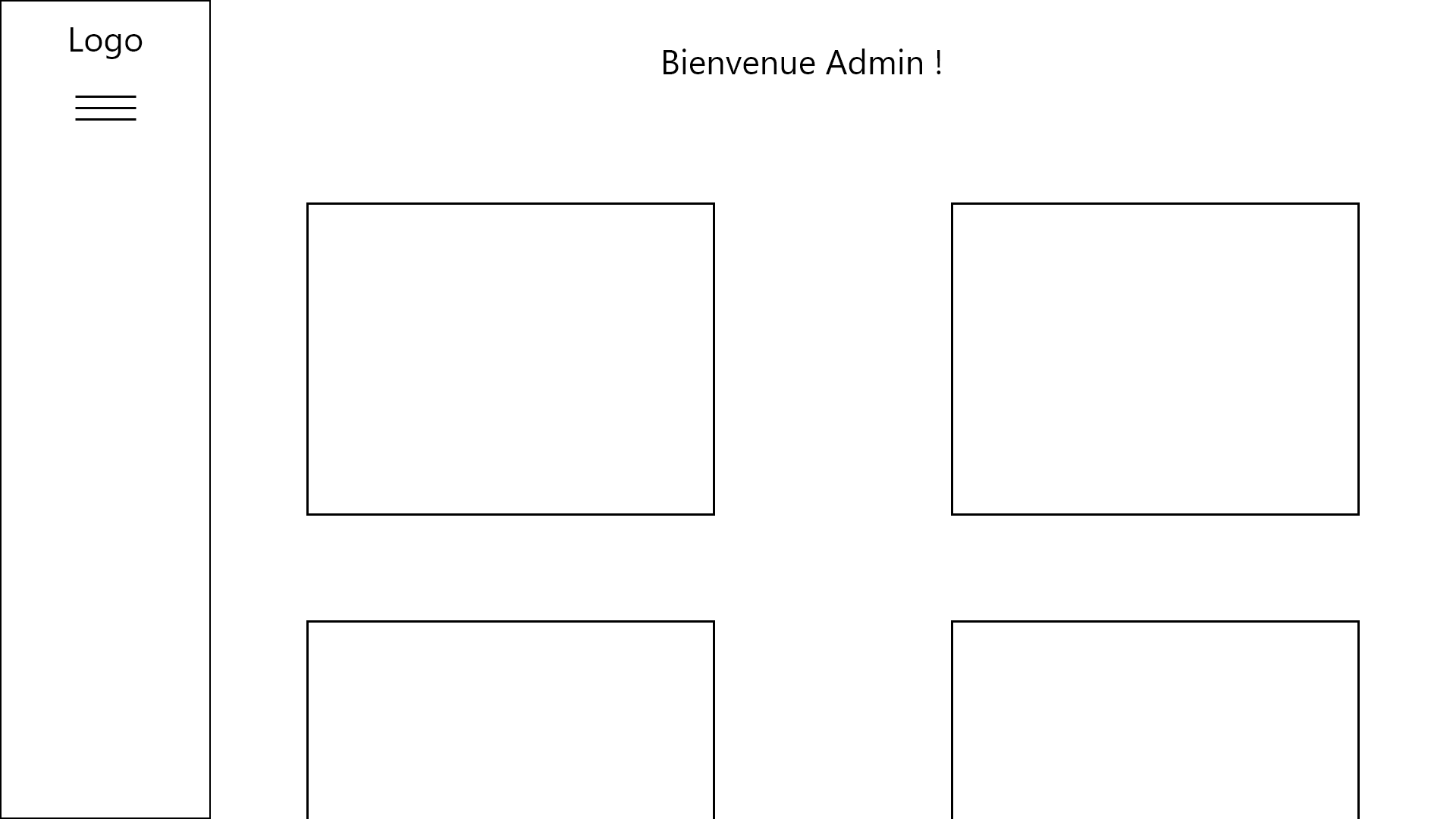


1. **Wireframes**

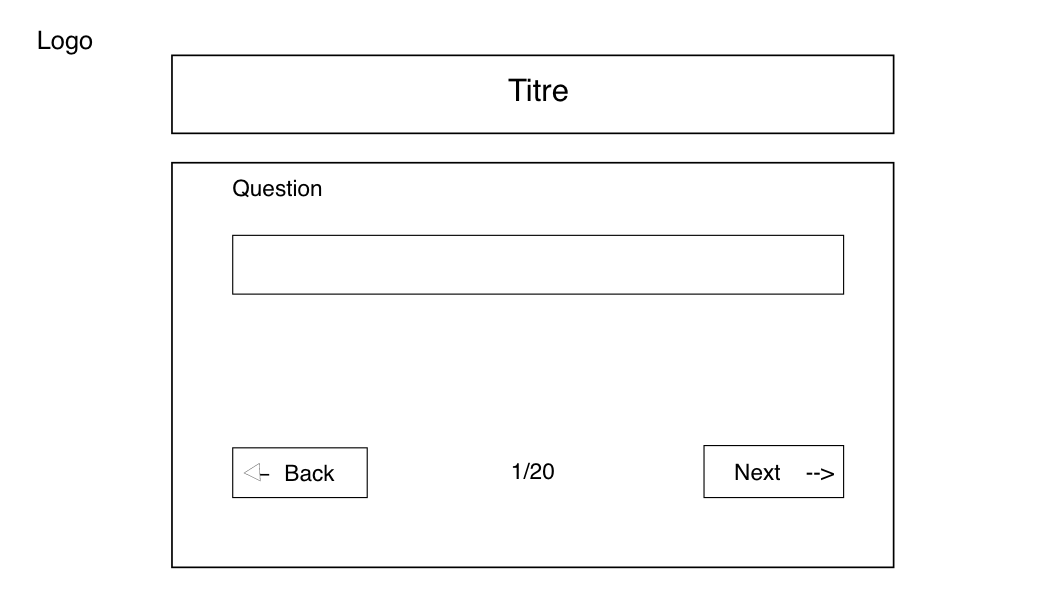
**Login Page**



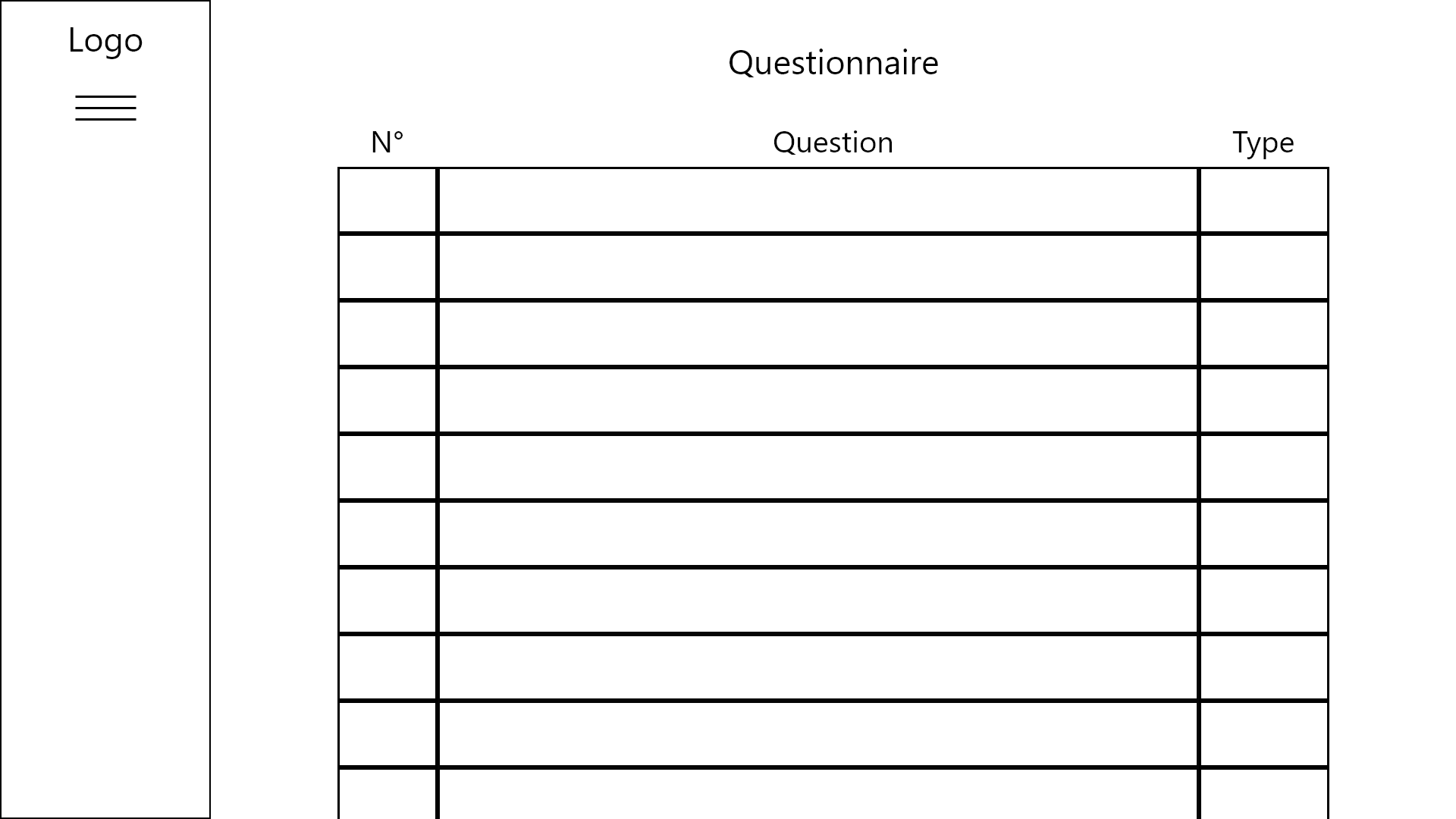
**Accueil**



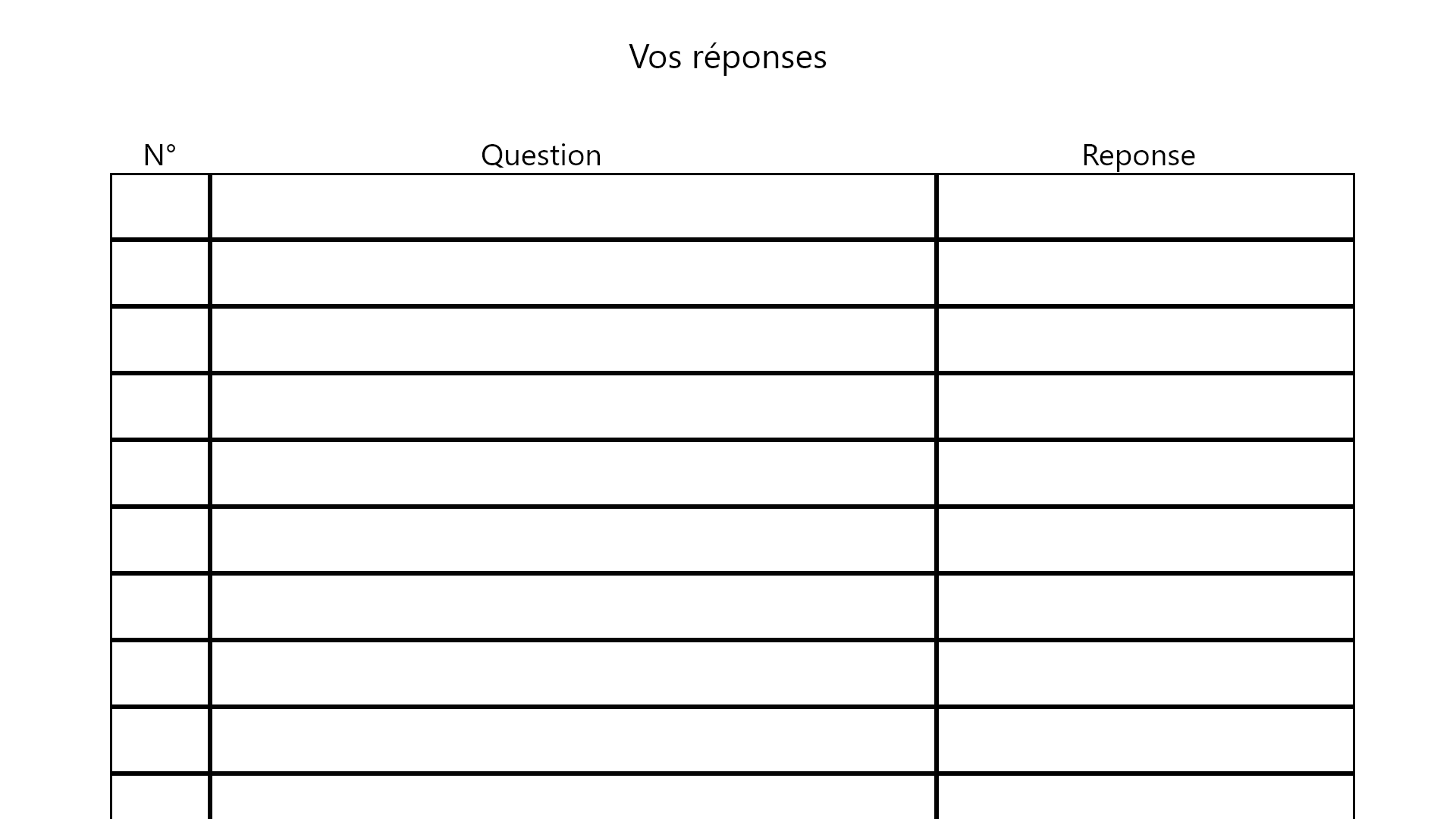
**Formulaires**

****

**Questionnaire**

****

Réponses



1. **Documentation du code**

**Back-end :**

**AdminController**

#### **Login (/admin/login)**

**Endpoint pour l'authentification des administrateurs.**

* **Méthode HTTP: POST**
* **Paramètres de la requête:**
  + **email (obligatoire) : Adresse e-mail de l'administrateur**
  + **password (obligatoire) : Mot de passe de l'administrateur**
* **Réponses:**
  + **200 OK: Authentification réussie. Retourne un jeton d'accès.**
  + **401 Unauthorized: Échec de l'authentification.**

#### **Store (/admin/register)**

**Endpoint pour l'ajout d'un nouvel administrateur.**

* **Méthode HTTP: POST**
* **Paramètres de la requête:**
  + **email (obligatoire) : Adresse e-mail du nouvel administrateur**
  + **password (obligatoire) : Mot de passe du nouvel administrateur**
* **Réponses:**
  + **200 OK: Administrateur ajouté avec succès.**

#### **Destroy (/admin/delete/{id})**

**Endpoint pour la suppression d'un administrateur.**

* **Méthode HTTP: DELETE**
* **Paramètres de la requête:**
  + **id (obligatoire) : ID de l'administrateur à supprimer**
* **Réponses:**
  + **200 OK: Administrateur supprimé avec succès.**
  + **404 Not Found: Administrateur non trouvé.**

### **QuestionController**

#### **Add Question (/questions/add)**

**Endpoint pour l'ajout d'une nouvelle question.**

* **Méthode HTTP: POST**
* **Paramètres de la requête:**
  + **type (obligatoire) : Type de question (une lettre)**
  + **body (obligatoire) : Corps de la question**
  + **choices : Choix de réponse (s'il y a lieu)**
* **Réponses:**
  + **200 OK: Question ajoutée avec succès.**

#### **Get All Questions (/questions/get)**

**Endpoint pour récupérer toutes les questions.**

* **Méthode HTTP: GET**
* **Réponses:**
  + **200 OK: Questions récupérées avec succès.**

#### **Update Question (/questions/update/{question})**

**Endpoint pour mettre à jour une question existante.**

* **Méthode HTTP: PUT**
* **Paramètres de la requête:**
  + **id (obligatoire) : ID de la question à mettre à jour**
  + **type (obligatoire) : Nouveau type de question (une lettre)**
  + **body (obligatoire) : Nouveau corps de la question**
* **Réponses:**
  + **200 OK: Question mise à jour avec succès.**
  + **404 Not Found: Question non trouvée.**

#### **Destroy Question (/questions/destroy/{id})**

**Endpoint pour la suppression d'une question.**

* **Méthode HTTP: DELETE**
* **Paramètres de la requête:**
  + **id (obligatoire) : ID de la question à supprimer**
* **Réponses:**
  + **200 OK: Question supprimée avec succès.**
  + **404 Not Found: Question non trouvée**

### **UserController**

#### **Add User (/user/add)**

**Endpoint pour l'ajout d'un nouvel utilisateur avec ses réponses.**

* **Méthode HTTP: POST**
* **Paramètres de la requête:**
  + **answers (obligatoire) : Tableau d'objets contenant les réponses de l'utilisateur**
  + **url (obligatoire) : URL associée à l'utilisateur**
* **Réponses:**
  + **200 OK: Utilisateur ajouté avec succès.**

#### **Get All Users Answers (/users/answers)**

**Endpoint pour récupérer toutes les réponses des utilisateurs.**

* **Méthode HTTP: GET**
* **Réponses:**
  + **200 OK: Réponses d'utilisateurs récupérées avec succès.**

#### **Get One User Answers (/user/answers/{id})**

**Endpoint pour récupérer les réponses d'un utilisateur spécifié.**

* **Méthode HTTP: GET**
* **Paramètres de la requête:**
  + **id (obligatoire) : ID de l'utilisateur**
* **Réponses:**
  + **200 OK: Réponses de l'utilisateur récupérées avec succès.**
  + **404 Not Found: Utilisateur non trouvé.**

#### **Get User URL (/user/url/{id})**

**Endpoint pour récupérer l'URL associée à un utilisateur.**

* **Méthode HTTP: GET**
* **Paramètres de la requête:**
  + **id (obligatoire) : ID de l'utilisateur**
* **Réponses:**
  + **200 OK: URL de l'utilisateur récupérée avec succès.**
  + **404 Not Found: Utilisateur non trouvé.**

#### **Destroy User (/user/destroy/{id})**

**Endpoint pour la suppression d'un utilisateur et de ses réponses.**

* **Méthode HTTP: DELETE**
* **Paramètres de la requête:**
  + **id (obligatoire) : ID de l'utilisateur à supprimer**
* **Réponses:**
  + **200 OK: Utilisateur et réponses supprimées avec succès.**
  + **404 Not Found: Utilisateur non trouvé.**

#### **Check Email Existence (/users/check/email)**

**Endpoint pour vérifier si une adresse e-mail existe dans la base de données.**

* **Méthode HTTP: POST**
* **Paramètres de la requête:**
  + **email (obligatoire) : Adresse e-mail à vérifier**
* **Réponses:**
  + **200 OK: E-mail existant.**
  + **200 OK: E-mail inexistant.**

**Front-end :**

#### **Requêtes API**

##### **getAllAnswers()**

Récupérer toutes les réponses de tous les utilisateurs depuis le serveur.

* Retourne: Une promesse résolvant la réponse du serveur.
* Catch: Si une erreur survient pendant la requête API.

##### **getAllQuestions()**

Récupérer toutes les questions depuis le serveur.

* Retourne: Une promesse résolvant la réponse du serveur.
* Catch: Si une erreur survient pendant la requête API.

##### **getOneUserAnswers(userId)**

**Récupérer les réponses d'un utilisateur spécifique du serveur.**

* Paramètres:
  + **userId (string)** - L'ID de l'utilisateur pour lequel les réponses sont demandées.
* Retourne: Une promesse résolvant la réponse du serveur.
* Catch: Si une erreur survient pendant la requête API.

##### 

##### **login(email, password)**

**Initie le processus de connexion en envoyant les identifiants de l'administrateur au serveur.**

* Paramètres:
  + **email (string)** - L'adresse e-mail de l'administrateur.
  + **password (string)** - Le mot de passe de l'administrateur.
* Retourne: Une promesse résolvant la réponse du serveur.
* Catch: Si une erreur survient pendant la requête API. Supprime le token du stockage local en cas d'erreur.

##### **addUserDb(userAnswers)**

**Ajoute les réponses de l'utilisateur au serveur.**

* Paramètres:
  + **userAnswers (Array)** - Un tableau de réponses utilisateur à ajouter.
* Retourne: Une promesse résolvant la réponse du serveur.
* Catch: Si une erreur survient pendant la requête API.

#### **Page de Connexion**

##### **LoginHandleChange(event)**

Gère l'événement de modification du formulaire de connexion, empêche le comportement par défaut, extrait l'e-mail et le mot de passe du formulaire et initie le processus de connexion. En cas de succès, navigue vers la route "/home".

* Paramètres:
  + event (Event) - L'objet d'événement associé à la modification du formulaire.

#### **Page d'Accueil**

##### **returnPieChartsDatas(questionId)**

Génère des données pour les graphiques type “pie” en fonction de l'ID de la question spécifiée.

* Paramètres:
  + **questionId (number)** - L'ID de la question pour laquelle les données du graphique sont demandées.
* Retourne: Un objet contenant les données du graphique.

##### **returnRadarChartsDatas(questionId)**

**Génère des données pour les graphiques type “radar” en fonction de l'ID de la question spécifiée.**

* Paramètres:
  + **questionId (number)** - L'ID de la question pour laquelle les données du graphique sont demandées.
* Retourne: Un objet contenant les données du graphique.

#### 

#### **Navbar**

##### **handleClickListItem(event)**

**Gère l'événement de clic sur un élément de liste, en définissant l'élément d'ancrage.**

* Paramètres:
  + **event (Event)** - L'objet d'événement associé au clic.

##### **handleClose()**

Ferme le menu de l'élément de liste en définissant l'élément d'ancrage sur null.

#### **Survey component**

##### **isValidEmail(email)**

**Vérifie si l'e-mail donné est valide en fonction d'une expression régulière.**

* Paramètres:
  + **email (string)** - L'adresse e-mail à valider.
* Retourne: True si l'e-mail est valide, false sinon.

##### **updateAnswersState(questionId, inputValue)**

**Met à jour l'état des réponses enregistrées pour un ID de question donné.**

* Paramètres:
  + q**uestionId (number)** - L'ID de la question.
  + **inputValue (string) -** La valeur de saisie de l'utilisateur pour la question.

##### 

##### **handleNext()**

Gère la navigation vers l'étape suivante du questionnaire. Valide l'entrée en fonction du type de question et met à jour les réponses en conséquence.

##### 

##### **handleBack()**

Gère la navigation vers l'étape précédente du questionnaire.

##### **submitAnswers()**

Soumet les réponses formatées à la base de données et redirige l'utilisateur vers sa page de réponses.

#### **Survey view**

##### **Fonctions de Génération d'Entrée**

* **generateCheckboxChoices(question):** Génère des inputs type checkbox en fonction des choix disponibles pour les questions de type “A”.
* **generateTextInput(question):** Génère un input type e-mail pour la question 1 et un input type text pour les questions de type “B”.
* **generateSelectChoices(question):** Génère un input de type select pour les questions de type “C”.

##### **generateInputsArray()**

**Génère un tableau d'inputs en fonction des types de questions.**

* Retourne: Un tableau d'éléments JSX représentant des champs.

##### **generateQuestionsDatasArray()**

**Génère un tableau de données de questions pour alimenter le composant stepper du formulaire.**

* Retourne: Un tableau d'objets représentant les données des questions.