Cahier des charges

Haojing Xu, Clément Spileers



# Analyse client

## Objectifs du Client:

### Développement d'une Application Web:

* + Bigscreen, notre client, souhaite la création d'une application web pour collecter des informations auprès des utilisateurs de leur application VR.
  + Le choix du framework Laravel côté serveur est indiqué pour une préférence pour une solution robuste et bien établie.

### Collecte d'Informations via un Sondage en Ligne:

* + L'objectif principal de Bigscreen est d'obtenir des données utilisateur significatives via un sondage en ligne.
  + Trois types de questions différents suggèrent une variété dans le type d'informations recherchées.

### Deux parties distinctes:

* + La division en deux parties distinctes (interface publique, administration privée) souligne la volonté d'offrir une expérience utilisateur complète.

### Administration Restreinte:

* + La nécessité d'une authentification pour accéder à la partie d'administration indique la confidentialité des données et la singularité de l'administrateur.

### Utilisation de Technologies Spécifiques:

* + Le choix spécifique de Laravel, Vue ainsi que l'utilisation de GitHub, reflète la volonté de Bigscreen d'adopter des technologies solides et modernes.

## Contraintes Techniques:

### Langage et Nommage:

* + L'insistance sur l'utilisation de l'anglais et du camelCase/PascalCase suggère une préférence pour des normes de codage cohérentes.

### Validation de Formulaire:

* + L'utilisation du service de validation de Laravel souligne l'importance d'une gestion efficace des données utilisateur.

### Gestion de Données:

* + L'ORM Eloquent de Laravel, associé aux migrations et seeders, témoigne de la volonté d'assurer une gestion de données structurée et évolutive.

# 

# Choix technologiques

## 

## Côté Front-end

**Vue.js (Framework JavaScript)**

* + **Justification :** Vue.js a été choisi pour sa simplicité, sa flexibilité et sa facilité d'intégration. Il offre une approche composant permettant une organisation modulaire du code, ce qui facilite la maintenance et l'évolutivité de l'application.

**Pinia (Gestionnaire d'état pour Vue.js)**

* + **Justification :** Pinia a été préférée pour sa simplicité et sa compatibilité native avec Vue 3. Il offre une gestion d'état réactive facilitant la communication entre les composants et le stockage des données.

**ChartJS (Intégration de graphiques)**

* + **Justification :** ChartJS est utilisé afin de pouvoir créer des graphiques qui permettent de visualiser plus simplement différentes données.

**Vue-chartjs (Dépendance complémentaire à ChartJS)**

* + **Justification :** Vue-chartjs est un ajout pour ChartJS spécialement conçu pour faciliter l'intégration des graphiques grâce à des composants prédéfinis.

**Axios (Bibliothèque HTTP)**

* + **Justification :** Axios est utilisé pour effectuer des requêtes HTTP depuis le front-end vers le back-end de manière asynchrone. Il est largement utilisé, offre une syntaxe claire et s'intègre bien avec Vue.js.

**Bootstrap (Framework CSS)**

* + **Justification :** Bootstrap a été choisi pour sa facilité d'utilisation et son système de grille qui simplifie le développement d'interfaces réactives. Il fournit également des composants prêts à l'emploi qui accélèrent le processus de conception.

## Côté Back-end

**Laravel (Framework PHP)**

* + **Justification :** Laravel a été choisi pour son élégance, sa simplicité, et sa puissance. Il offre une architecture MVC robuste, des fonctionnalités de sécurité intégrées et une syntaxe expressive qui accélère le développement.

**MySQL (Système de gestion de base de données)**

* + **Justification :** MySQL a été choisi pour sa fiabilité, sa performance et sa compatibilité avec Laravel. Il offre un support étendu pour les fonctionnalités de base de données nécessaires à l'application.

# Adresse Github

Front : [https://github.com/Irjean/ProjetFinalLEM\_front](https://github.com/Irjean/ProjetFinalLEM_back)

Back: <https://github.com/Irjean/ProjetFinalLEM_back>

# Déploiement

## Technologies à vérifier

* S'assurer que le serveur d'hébergement (par exemple, Apache, Nginx) est configuré pour prendre en charge l'application.
* Vérifier que votre version de PHP au dessus de 8.1 et votre version de NodeJS est au dessus de 8.9
* Assurez-vous que la base de données (MySQL, PostgreSQL, etc.) est correctement configurée.

## Actions à effectuer dans le terminal

Front :

- npm install

-npm run dev

Back :

- composer install

- php artisan key:generate

- php artisan migrate

- php artisan db:seed

- php artisan serve

## Autres actions nécessaires pour arriver au rendu final

Pour le back :

* Faire une copie du fichier “.env.exemple” et le renommer “.env”
* Configurer le fichier “.env” avec les données nécessaire pour vous connecter à votre base de données

# 

# Méthode de travail

Méthode de travail utilisée : **Waterfall**.

**Github** : une branche “dev” fût d’abord créée pour pouvoir tester branches intégrés dans le but de déceler les possibles bugs ou conflits et de les régler avant de passer le projet en production.

**Trello** : Dans le but de pouvoir se répartir les tâches, un espace de travail sur Trello à été créé, chaque tâche à été triée en fonction des pages, routes, features à développer, si c’était pour le front, le back, la documentation, l’organisation etc.

Les tâches ont été divisées en deux parties : Côté utilisateur et côté administrateur.

**Front** : Pour chaque page à intégrer, une nouvelle branche devait être créée sur github.

**Back** : Une branche devait être créée pour le développement des seeders, des migrations ainsi que chaque route à développer.

**Code Review** : Lorsque le travail était terminé pour une page/route, une merge request vers la branche “dev” devait être créé afin que le binôme puisse vérifier l’intégrité du code.

# Outils utilisés

## Logiciels

* Environnement de développement intégré (IDE) : **Visual** **Studio Code**
* Système de gestion de version : **Git**

## Outils et services de communication

* Messagerie instantanée : **Discord**
* Gestion de projet : **Trello**

## Outils de développement

* Pour la gestion des dépendances PHP : **Composer**
* Pour la gestion des dépendances JavaScript : **NPM**
* Pour les tests des routes : **Postman**
* Pour visualiser le diagramme de la base de données : **dbdiagram.io**
* Pour la création du zoning et du wireframe : **Figma**

## Service en lignes

* Pour l'hébergement du code source : **GitHub**

# Évaluation du temps de travail

| **Poste de Développement** | **Temps de Travail** | **Développeur** | **Remarques** |
| --- | --- | --- | --- |
| Configuration de la Base de Données (Laravel) | 1 jours | Clément/Haojing | Configuration du fichier .env, création de migrations, modèles Eloquent, et relations. |
| UX/UI Design | 2 jours | Haojing | Création du zoning et des wireframes en desktop et mobile pour l’intégration avec Figma. |
| Définition des Modèles Eloquent | 1 jour | Clément | Création des modèles Eloquent pour les questions et les réponses, et définir les relations. |
| Migrations & Seeders | 1 jour | Clément/Haojing | Utilisation de migrations pour définir la structure de la base de données et création de seeders pour peupler la base de données avec des données de test. |
| Routes API & Contrôleurs | 2 jours | Clément/Haojing | Création des routes API pour le front-end, implémentation des contrôleurs pour la logique métier. |
| Validation des Données (Laravel) | 1 jour | Clément | Mise en place des règles de validation pour les formulaires. |
| Front - Questionnaire | 5 jours | Haojing | Créations des pages concernant le questionnaire, la récupération des questions, l’envoie des réponses au back et récupérer les réponses envoyées pour l’utilisateur. |
| Front - Administration | 5 jours | Clément | Créations des pages de l’administration, de la récupération des données (questions, réponses, profils). |
| Rédaction de la documentation et du cahier des charges | 3 jours | Clément/Haojing | Rédaction de la documentation pour le code afin d’aider à la compréhension de l’installation. Rédaction du cahier des charges pour faire un compte-rendu au client. |
| Total | 14 jours |  |  |

# 

# Liste fonctionnelle

### Utilisateur

* Accéder à un sondage
* Répondre à un sondage.
* Visualiser ses réponses à un sondage.

### Administrateur

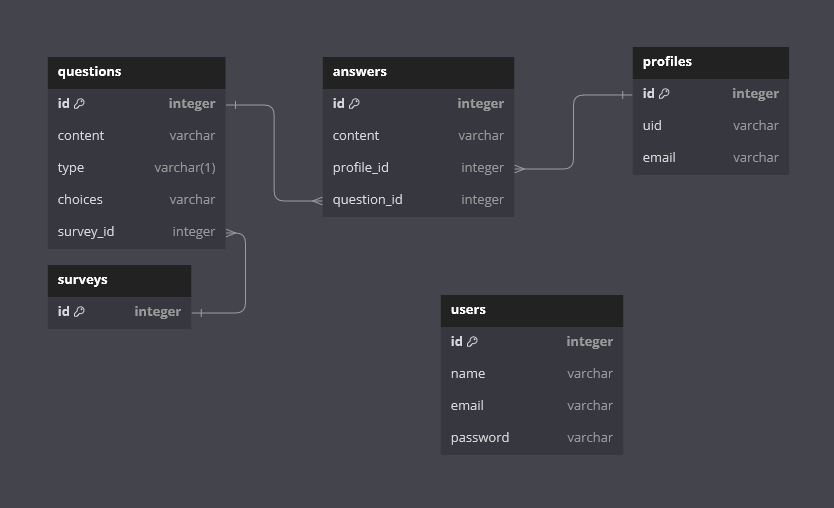
* Se connecter à l’interface administrateur
* Visualiser des graphiques liés aux réponses au sondage
* Visualiser les questions du sondage
* Visualiser les réponses par profil

## Recettage

| **Fonctionnalités** | **Status** | **Commentaires** |
| --- | --- | --- |
| Accéder à un sondage | Opérationnelle | Aucun bug |
| Répondre au sondage | Opérationnelle | Aucun bug |
| Visualiser ses réponses | Opérationnelle | Aucun bug |
| Connexion à l’administration | Opérationnelle | Aucun bug |
| Visualiser les graphiques | Opérationnelle | Aucun bug |
| Visualiser le questionnaire | Opérationnelle | Aucun bug |
| Visualiser les réponses | Opérationnelle | Aucun bug |

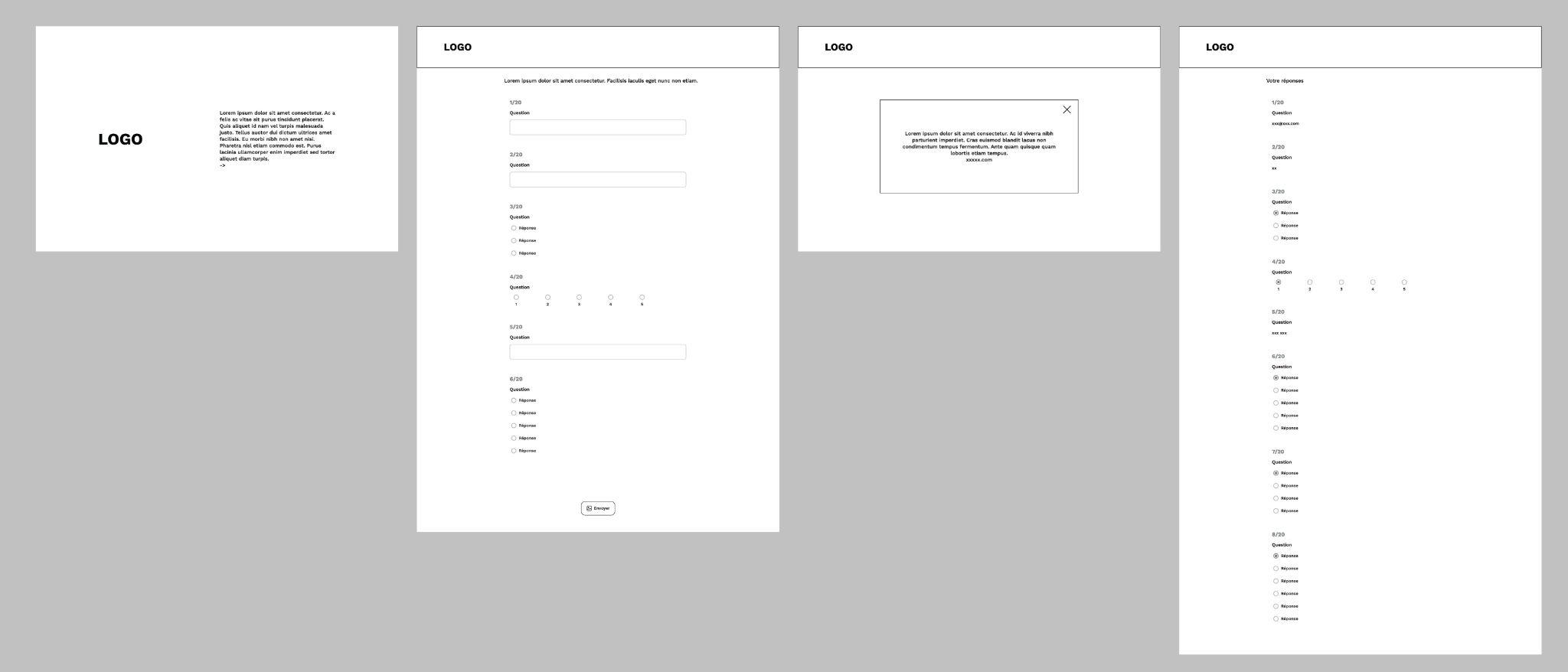
# 

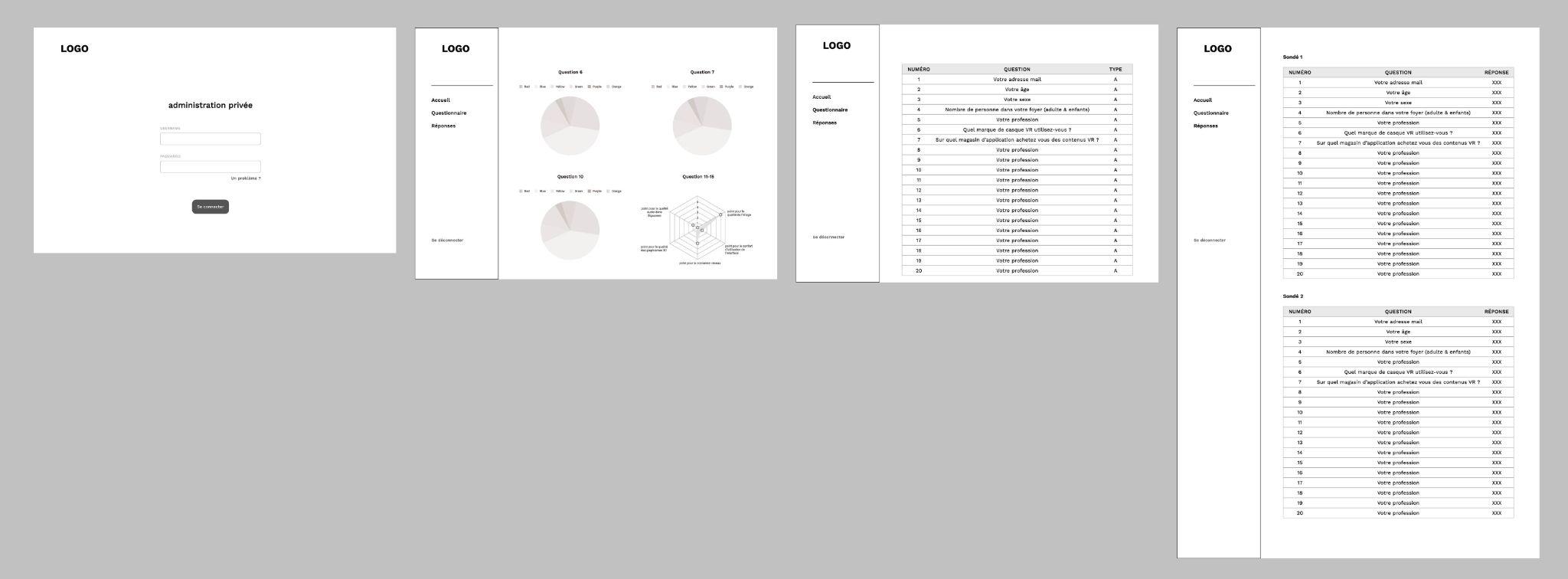
# Diagramme de la base de données

**

# 

# Wireframes

**



# 

# Documentation du code

## Back

**Tables :**

* Users : Contient un nom, un email, un email et un mot de passe. Un user pourra accéder à la partie administrateur du site.
* Surveys : Contient un ID. Permet de lier les questions un à questionnaire.
* Questions : Contient un contenu, un type, des choix et l’ID d’un questionnaire. Permet de savoir à quelle question on répond.
* Answers : Contient un contenu, l’ID d’une question et l’ID d’un profil. Permet de savoir ce qu’à répondu un utilisateur à une question.
* Profiles : Contient un email et un UID. Permet de savoir quel utilisateur répond aux questions, et grâce à son UID, l’utilisateur peut consulter ses réponses.

**Functions :**

* UserController[Login] (POST) : Permet à un utilisateur de se connecter à l’administration du site.
* QuestionController[getQuestions] (GET) : Permet de recevoir les questions stockées dans la base de données
* AnswerController[getOneAnswer] (GET) : Permet d’accéder aux réponses d’un seul utilisateur
* AnswerController[getAllAnswers] (GET) : Permet d’accéder aux réponses de tous les utilisateurs.
* AnswerController[postAnswer] (POST) : Permet à un utilisateur d’enregistrer ses réponses dans la base de données.

## 

## Front

**Composants :**

* HomePage : La page d’accueil du site. Accueil l’utilisateur avec une courte présentation de l’entreprise BIGSCREEN et du but du site. L’utilisateur peut soit consulter la page principal de BIGSCREEN ou accéder à la page pour répondre au questionnaire.
* SurveyPage : La page du questionnaire. Lors de l’affichage, fais une requête au back pour obtenir les questions du questionnaire, et les affiches selon le type de la question. Une fois que l’utilisateur à répondu à toutes les questions, une requête est envoyée au back, qui va créer un profil lié à l’email qu’il aura fourni et lier les réponses qu’il aura fournies à ce profil. Une popup s'affiche avec un lien qui lui permettra d’accéder à ses réponses.
* AnswerPage : La page qui permet à l’utilisateur de voir les réponses qu’il aura fourni au questionnaire. Une requête est faite au back avec l’UID qui se trouve dans le lien qu’il a reçu qui permettra de récupérer les réponses liées à son profil.
* LoginPage : La page de connexion de l’administration. Un utilisateur enregistré peut s’authentifier pour accéder à l’interface utilisateur.
* AdminPage : La page d’accueil de l’administration. Cette page contient trois graphiques “pie” qui contiennent les réponses des questions 6, 7 et 10. Un quatrième graphique “radar” permet de visualiser la moyenne pour les réponses onze à quinze.
* AdminSurvey : Une page administrateur qui permet de consulter les questions du questionnaire.
* AdminAnswer : Lors du chargement de cette page, une requête est faite au back pour récupérer les réponses de tous les utilisateurs, les données sont ensuite affichées sous forme de tableau par utilisateur.
* NavBar : Correspond au header du site. Permet de revenir à la page d’accueil.
* AdminNavBar : Un composant similaire au header qui permet de naviguer entre les différentes pages de l’administration. Il est aussi possible de retourner sur la page d’accueil en cliquant sur le logo.
* SurveyQuestion : Utilisé dans la page permettant de répondre au questionnaire, c’est un composant qui est rendu plusieurs fois grâce à une boucle de la liste des questions du questionnaire. Chaque composant correspond à une question avec son titre et son type. Il contient également divers inputs différents selon le type de la question.
* SurveyAnswer : Un composant utilisé pour afficher les réponses d’un utilisateur.
* Loading : Une animation de chargement qui permet de montrer à l’utilisateur qu’une requête est en cours et qu’il a besoin d’attendre quelques instants.

**Stores Pinia :**

* question : Un état qui permet de stocker les questions dans un tableau afin de les réutiliser plus tard, et un booléen qui permet de savoir si les questions ont déjà été stockées.
* Answer : Il est composé d’un tableau comportant les réponses de tous les utilisateurs, un tableau avec les réponses d’un utilisateur lorsqu’il répond au questionnaire, un booléen qui permet de savoir si les réponses de tous les utilisateurs ont déjà été reçu par le back, et un booléen qui permet de savoir si l’utilisateur à répondu à toutes les questions du questionnaire.
* Navbar : Un booléen qui permet de savoir si la navbar administrateur doit être affiché ou non.
* admin : Permet de savoir si un administrateur est connecté.