Projet Semestre 6

PolyQuiz

Team.js

DEFENDINI Lara
RAFFIN Louis
SAVORNIN Guillaume
VAN DER TUIJN Anton

Rapport de projet

Sommaire

Ce projet a pour but de mettre à disposition pour des résidents d'une EHPAD ou d'un institut pour les personnes dépendantes une application intuitive et fonctionnelle afin de leur permettre de se divertir. Cette dernière doit permettre de réaliser des quiz tout en s'adaptant à des handicaps visuels ou moteur des utilisateurs. Ce site web utilise la technologie de Google : Angular. Il permet aussi au personnel soignant de disposer de fonctionnalités supplémentaires comme par exemple la gestion des comptes des résidents, mais aussi la possibilité de superviser les statistiques ou encore créer de nouveaux quiz. Nous allons voir au travers de ce rapport la démarche d'ingénieur qui nous a amenés à identifier les besoins des utilisateurs, puis à créer une maquette pour y répondre, et enfin implémenter cette dernière en la modifiant si besoin.

1 Personas et scénarios — version finale

1.1 Persona n°1

P1. Jean, 74 ans est atteint d'une maladie des articulations : l'arthrose. Il a des douleurs lorsqu'il fait des activités manuelles, il ne souhaite pas que les autres résidents se rendent compte de ses incapacités donc il refuse de participer à certaines activités.

P2. Bernard, 29 ans est l'un des membres du personnel médical du centre, il est chargé d'organiser des animations thérapeutiques pour des résidents durant les journées. Il a remarqué que certain des résidents refusaient de participer aux activités manuelles car ils ont peur d'être mis en échec.

Scénario d'usage — Aide-soignant
Bernard souhaite proposer aux résidents une nouvelle activité : PolyQuiz, il espère que cette nouvelle activité plaira à un grand nombre. Il crée les compte résidents pour chaque résident dont il s'occupe. Grâce à l'outil qui permet de partager des paramètres, il gagne du temps en copiant les paramètres entre plusieurs résidents ayant les mêmes difficultés. Comme il connaît bien le groupe de résident avec le quel il travaille, il décide de créer un nouveau quiz.
Les résidents testent le nouveau quiz de Bernard.
Bernard retourne sur l'interface administrateur afin de visualiser les statistiques de chacun des membres du groupe dont il s'occupe. Il remarque qu'un de ses résidents n'a répondu à aucune question correctement, il en conclu que quelque chose va mal...

1.2 Persona n°2

P3. Anne, 76 ans est atteinte de la maladie de Parkinson, elle a beaucoup de mal à être précise avec ses mains ce qui lui complique l'interaction avec des objets de petite taille.

P4. Julie, 32 ans est aide-soignante et s'occupe de de plusieurs résidents, et entre autres de Anne. Elle aimerait proposer aux résidents des activités qui utilisent l'outil informatique.

Scénario d'usage — Résident
Anne, pensionnaire, est intriguée par l'outil informatique et souhaite apprendre à s'en servir. Julie, aide-soignante, a choisi d'utiliser PolyQuiz pour permettre aux résidents d'être initiés à l'outil informatique tout en s'amusant et faisant travailler leur mémoire. Anne a des problèmes moteurs et souffre de la maladie de Parkinson, elle a donc du mal à être précise quand elle appuie sur un bouton et a besoin de plus gros boutons. Julie choisit donc le handicap moteur dans les paramètres de l'application et choisit un thème puis un quiz avec Anne, en regardant les vidéos d'aide. Si besoin en cours de partie, Anne peut demander à Julie d'ajuster à nouveau les paramètres puis reprendre le quiz là où elle s'est arrêtée.
Pour plus d'aisance, il est possible de répondre au quiz avec le clavier et/ou la souris.

1.3 Persona n°3

P5. Marie, 89 ans est une personne très dynamique, elle est atteinte de problèmes de vision (malvoyante). Elle a des difficultés dues à son handicap pour lire les textes de petite taille et voir les détails sur les images.

P6. Laure, 18 ans est l'aide soignante qui s'occupe de Marie, elles font de temps en temps des activités ensemble.

Scénario d'usage — Résident
Aujourd'hui Laure souhaite faire découvrir à Marie, une de ses patientes, les quiz PolyQuiz. Elle crée un compte pour Marie et paramètre le handicap sur "visuel" (grande police) car elle souffre de la cataracte et ne voit pas bien. Elle souhaite aussi changer le contraste et la police. Marie, ancienne professeure de Français, choisit un thème grammaire seule. Lors d'une prochaine connexion, Laure sait que les paramètres resteront enregistrés, ce qui permettra à Marie d'être autonome pour la sélection et la réalisation d'un quiz. En cas de problème et si Laure n'est pas dans les parages pour l'aider, Marie peut utiliser les vidéos d'aide.

2 L'architecture Client Serveur

Il existe plusieurs moyens de communication entre deux machines. Le modèle le plus connu est le client-serveur. Une première machine dite "client" va faire des requêtes vers une seconde machine dite "serveur". Cette dernière sert le client (d'où son nom). Par exemple le protocole HTTP (web) repose sur ce modèle. Mais il existe un autre moyen de communications entre des machines, le pair à pair (peer to peer). La machine endosse alors à la fois le rôle de client et de serveur. Elle va alors faire des réponses à des requêtes quand elle le peut, et en émettre dans le cas contraire. Un exemple de protocole basé sur le pair à pair est le torrent, il permet de télécharger et partager des fichiers de manière décentralisé. Pour simplifier son fonctionnement, le fichier est découpé et les machines peuvent alors partager les parties qu'elles possèdent et télécharger chez les autres celles qu'elles n'ont pas encore.

Dans notre cas nous sommes en présence d'une architecture client-serveur dite de trois niveaux (three-tier). Notre client web va faire des requêtes pour accéder à des ressources. Un serveur intermédiaire appelé serveur "d'application" va répondre à la requête du client. Mais dans certains cas ce dernier va faire à son tour des requêtes web sur un autre serveur dit serveur de "données" pour récupérer des ressources. Différencier le serveur d'application et le serveur de données présente plusieurs avantages. Tout d'abord, si l'on souhaite créer une application mobile ou autre, on peut conserver le serveur de données tel quel. Un autre avantage est d'améliorer la sécurité, car c'est le serveur d'application qui décider de faire appel au serveur de données et non pas le client, cette couche supplémentaire éloigne les données du client. La liaison entre le serveur d'application et le serveur de données se fait en suivant un ensemble de normes définies par ce que l'on appelle une API REST. Nous reviendrons plus en détails sur l'API REST dans la suite du rapport.

2.1 Front-end

2.1.1 Les composants principaux

Nous avons essayé d'utiliser au mieux les particularités d'Angular pour pouvoir réutiliser certaines parties de notre code en créant notamment de nombreux composants. Certains d'entre eux sont composés de sous composants que nous n'avons pas affiché dans le schéma ci-dessous. Cette architecture nous a aussi permis de travailler en équipe plus simplement, car nous pouvions chacun travailler sur une partie du code isolée des autres.

Composants principaux

Parmi les composants les plus importants, nous retrouvons notamment le "play" qui permet de jouer à un quiz. Ce composant est accessible via une route dans laquelle on indique l'ID unique du quiz auquel on souhaite jouer. (/play/IDQUIZ)

Le “header” et le “menu” illustrent parfaitement le fait de pourvoir réutiliser un code, ils sont appelés une fois à la racine de la page et est visible partout.

Exemple de découpage d’une page avec ses composants

2.1.2 Les services

Nous utilisons différents services pour communiquer avec notre API, et d’autres afin de gérer les différentes parties de notre site.

Chacun de nos services utilisent des ”Observables”. Ils nous permettent d’échanger des informations entre nos composants et de les garder à jour au fur et à mesure que les données changent.

Principaux services de notre application

Parmi eux, on retrouve des services qui nous permettent de communiquer avec notre API : Games, Quiz, Result, Settings, Statistic, Themes, Auth. Le service ”Auth” sert à l’Authentification des utilisateurs. C’est grâce à lui que nous pouvons nous connecter.

Les services du Toaster

Nous avons aussi créé des services pour d'autres raisons que la communication entre le front-end et le back-end. Par exemple l'un d'entre eux nous permet d'afficher un "Toaster" qui nous sert à améliorer le feedback donné aux utilisateurs du site.

Un Toaster est une petite fenêtre qui nous donne des informations pendant quelques secondes puis qui disparait.

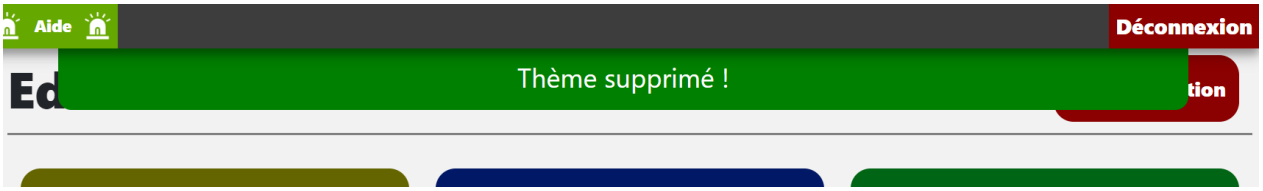
Ils sont très communs, nous aurions pu utiliser ceux d'un module npm déjà existant ou directement ceux de Bootstrap, mais ils n'auraient pas ressemblé à ce que nous avions prévu initialement dans notre maquette.

C'est pourquoi nous avons décidé d'en faire nous-même, pour les utiliser c'est très simple, il suffit d'appeler la fonction ci-dessous du service et le toaster s'affichera :

```
this.toasterService.activateToaster( isErrorMessage: false,  message: 'Contraste enregistré !',  ms: 1000);
```

Message d'erreur

Message de succès



Les services auth-guard

Notre site a besoin de comptes utilisateurs pour être utilisé, certaines parties ne doivent pas être accessible à tout le monde. On a alors voulu sécurisé un minimum les routes de notre site web.

Pour cela, on utilise des services Auth-Guard qui nous permettent de bloquer l'accès à une route si l'utilisateur n'a pas les autorisations nécessaires.

Ces services sont appelés dans le module de Routing, de cette façon ils sont exécutés au chargement de la page. Puis si un utilisateur n'a pas les droits d'aller sur une page alors il sera redirigé, dans le cas contraire l'accès lui sera accordé et la page sera chargée.

Exemple de route sécurisée par nos différents Auth-Guard

```
{path: 'welcome', canActivate: [NotAuthGuardService], component: WelcomeComponent},
{path: 'themes', canActivate: [AuthGuardService], component: ThemeComponent},
{path: 'residents', canActivate: [AuthGuardAdminService], component: ResidentsComponent},
```

Notre projet en comporte trois différents, un pour chacun des cas possibles : soit l'utilisateur n'est pas connecté, soit il est connecté en utilisateur normal ou alors il est connecté en utilisateur admin.

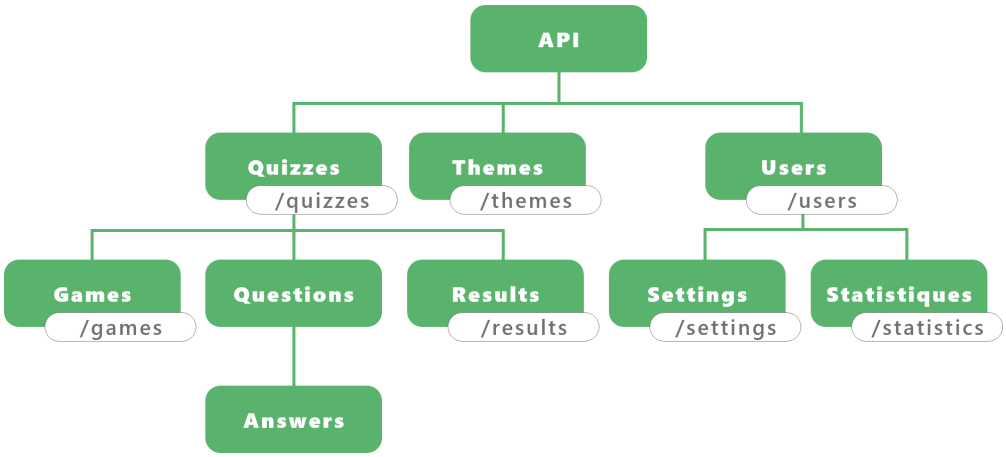
2.2 Back-end

2.2.1 Structure de notre API

L'API REST (REpresentational State Transfer) est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web. Dans l'architecture REST une ressource est égale à une URL et pour interagir avec les ressources nous avons à disposition quatre méthodes : POST, PUT, GET, et DELETE. POST permet la création d'une ressource, PUT permet de mettre à jour une ressource, GET récupère une ressource et DELETE supprime une ressource. L'API REST répond à chaque demande par un code de réponse HTTP, par exemple 200 signifie que tout s'est bien passé et 500 signifie qu'il y a une erreur interne au serveur.

Nous avons utilisé ces codes de réponse dans notre projet, notamment pour donner un feedback a l'utilisateur sur l'état de l'action qu'il souhaite réaliser. Par exemple, lors de la ,*création d'un quiz, si le code de résultat vaut 201 nous indiquons à l'utilisateur dans un toast que le quiz a bien été créé, mais si nous avons un autre code de résultat nous indiquerons qu'une erreur est survenue.

Nous avons continué à utiliser l'API REST que nous avions commencé à mettre en place lors des TD d'initiation. La structure des Quiz ainsi que les Questions et Réponses reste relativement similaire à celle qui nous a été donnée dans le Template de base, nous y avons tout de même ajouté/modifié diverses choses, comme par exemple la suppression d'un Quiz, Question ou Réponse qui ne supprime plus définitivement la ressource mais ajoute une étiquette "DELETED" (Cela permet de pouvoir continuer à visualiser les statistiques d'un résident sur un quiz qui "n'existe plus "). Nous avons ajouté d'autre structures de ressources visibles sur le schéma si dessous.



Structure générale de l'API

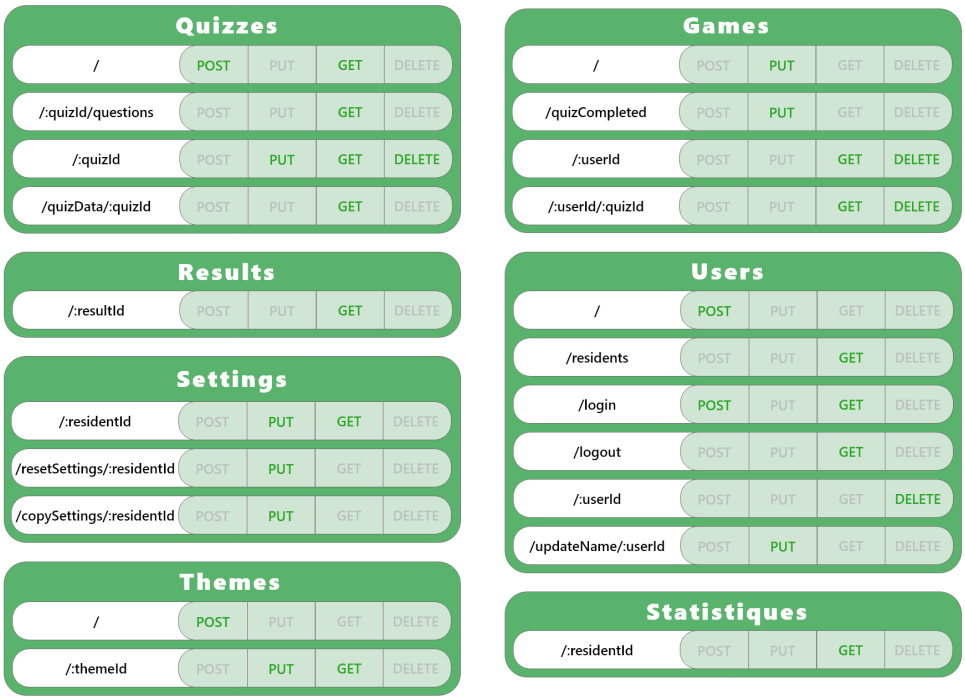
- **Games** permet la gestion de la progression des utilisateurs sur les quiz, cela permet à l'utilisateur de pouvoir quitter un quiz en cours de route pour par exemple changer les paramètres de vision, puis il peut revenir et reprendre le quiz au niveau de la question où il s'est arrêté.
- **Results** est utilisé pour la sauvegarde des réponses d'un utilisateur sur un quiz. A chaque fois qu'un utilisateur fini un quiz une ressource **Result** est créée.
- **Users** permet la gestion de tout ce qui se rapporte aux comptes résidents et administrateur.
- **Settings** sauvegarde les paramètres de Vision et Moteur de chaque **User**. De cette manière, sur un même appareil partagé par plusieurs utilisateurs, chacun des utilisateurs aura accès à ses propres paramètres qu'il pourra modifier à sa guise.

- **Statistiques** a été mis en place pour que les comptes administrateurs des aides-soignants puissent visualiser les statistiques de chacun des résidents. Il est notamment possible de voir le nombre de quiz réalisés (au total et lors de la semaine courante), le pourcentage de bonnes réponses et l'historique des réponses à tous les quiz réalisés.

2.2.2 Les ressources

Dans notre architecture il est possible d'accéder directement aux sept ressources listées ci-dessous. Questions et Answers ne sont pas listées car nous n'avons pas l'utilité d'avoir un accès direct, nous y accédons indirectement par le biais de Quizzes par exemple. Pour la création d'un quiz nous utilisons le POST sur la racine (/), dans le cas où l'utilisateur souhaite éditer son quiz en ajoutant une/des question(s)/réponse(s) on utilise le PUT sur la route /:quizID.

Comme expliqué précédemment (cf. partie 2.2.1) nous avons choisi de modifier le DELETE en mettant en place une étiquette "DELETED" qui indique que la ressource ne doit pas être récupérée lors d'un appel du Front-end. Néanmoins il y a une exception à cette règle, il y a des endroits où nous avons besoins des ressources "DELETED" par exemple lorsqu'un aide-soignant souhaite revoir les résultats d'un résident sur un quiz qui contient des questions qui n'existent plus (étiquetées "DELETED").



Ressources

- **Quizzes** permet de créer, éditer, récupérer et supprimer un quiz. Un quiz est construit grâce à une méthode qui récupère les questions et réponses associées au quiz. La route /quizData/:quizId permet la récupération d'un quiz avec les ressources étiquetées "DELETED".
- **Games** : A chaque fois qu'un utilisateur répond à une question sa réponse est envoyé pour sauvegarder sa progression. Lorsque on arrive à la dernière question du quiz, la réponse est envoyée sur /quizCompleted. Cela crée une nouvelle ressource **Result** (résultat d'un quiz) et met à jour ou crée, si elles n'existent pas, les statistiques de l'utilisateur. Lorsque l'utilisateur abandonne sa progression sur un quiz, nous utilisons la méthode DELETE qui ici supprime réellement la ressource **Game**.

- **Users** permet la création, récupération, mise à jour et suppression d'un utilisateur. A partir du /login on vérifie l'authentification (GET pour vérifier si un cookie existe déjà et le POST pour se connecter). Lors de la création d'un nouvel utilisateur on crée aussi une ressource Settings qui est construite en fonction des assistances (Visuel et/ou Moteur) et indique lors de la création du compte.
- **Results** est utilisé exclusivement pour récupérer le résultat d'un quiz pour un utilisateur, et utilise les références des questions et réponses données par l'utilisateur.
- **Settings**: A partir de l'identifiant de l'utilisateur on peut récupérer ou mettre à jour ses paramètres. Grâce au resetSettings l'utilisateur peut réinitialiser ses paramètres avec les valeurs par défaut (dépendant de ses assistances : par exemple si l'utilisateur a une assistance Moteur il aura par défaut une zone de sélection de grande taille). Le copySettings quant à lui a été implémenté pour les aides-soignants. Il leur permet de copier les paramètres d'un résident vers un/des autres résidents, afin de lui faire gagner du temps lors du paramétrage des comptes.
- **Themes** : Il est possible de créer, mettre à jour et supprimer un thème. Pour la suppression il est nécessaire qu'aucun quiz n'ait une référence vers ce thème. Si cette condition est respectée le thème est alors étiqueté "DELETED".
- **Statistiques** est une ressource qui est surtout utile pour les aides-soignants, mais aussi accessible par les résidents en version réduite pour ne pas les perturber. Il est uniquement possible de récupérer les statistiques d'un résident à partir de son identifiant.

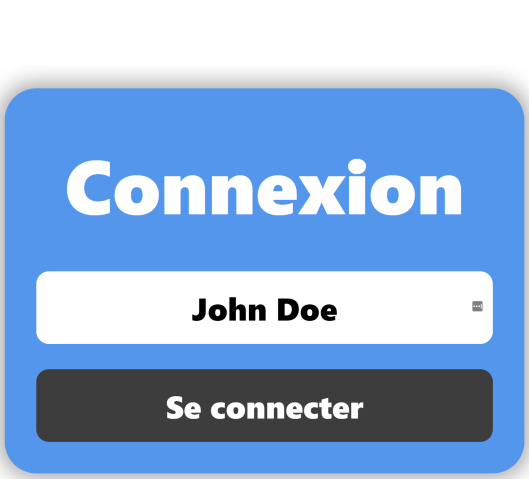
2.2.3 Authentification des utilisateurs

Pour pouvoir utiliser notre site, chaque utilisateur a besoin d'avoir un compte et d'être connecté. Cela nous demande alors de stocker ces comptes et de garder une trace des utilisateurs connectés pour qu'ils restent connectés sur une longue durée.

Stockage des comptes

Pour les résidents, il est demandé (au résident lui-même ou à l'aide-soignant) de rentrer juste leur nom et prénom, pas besoin de mot de passe pour se connecter. Mais en réalité un mot de passe est quand même créé pour qu'ils aient une base de données partagée avec les utilisateurs administrateurs et que les comptes soit standardisés. Cependant les utilisateurs administrateurs eux ont besoin d'un mot de passe.

Interfaces de connexion



Notre site web est destiné à des EHPAD, mais cela ne veut pas dire qu'il ne doit pas être un minimum

sécurisé. Nous devons alors stocker ces mots de passe de manière sécurisé. Pour cela nous utilisons le module **Bcrypt** qui va nous permettre de hacher les mots de passe avant de le stocker.

Le fait stocker simplement un hash va renforcer la sécurité puisque même si une personne malveillante tombe dessus elle ne pourra pas les reconvertir en mot de passe pour se connecter.

De plus ce qui rend ces hash si sécurisés est qu'un "sel" est ajouté ce qui le rend unique même si on hash plusieurs fois le même mot de passe.

Sauvegarde des sessions utilisateurs

Lorsque les résidents se connectent nous voulons qu'ils le restent pendant une longue durée, même s'ils quittent le site web ou s'ils ferment le navigateur pour revenir une semaine plus tard. Cela leur éviterait de devoir se connecter à chaque fois qu'ils ouvrent le site.

Pour cela il existe plusieurs méthodes, nous avons choisi d'utiliser la méthode d'enregistrement des **Session utilisateurs**. Nous envoyons alors un cookie contenant une valeur de session utilisateur qui va être enregistré sur l'appareil de l'utilisateur et sur le serveur. Puis, lorsque la personne revients sur le site, une comparaison est effectuée pour voir si le serveur et l'utilisateur ont bien les mêmes valeurs. SI c'est le cas, alors l'utilisateur est authentifié.

D'autres méthodes comme l'utilisation de token avec JWT (JSON Web Token) sont plus courantes. Ce système permet une meilleure "scalability" (évolutivité) car rien n'est sauvegardé sur le serveur. Mais dans notre cas cela n'est pas nécessaire, cela est dû au faible nombre d'utilisateur.

3 Evaluation croisée et résultats obtenus

Nous avons choisi de réaliser la première version d'évaluation croisée, dans laquelle les équipes vont mutuellement installer et tester l'application réalisée par l'autre équipe, sous la supervision de cette dernière. C'est avec l'équipe "OK Google" que nous avons pu mettre en place cette évaluation qui a duré quasiment trois heures.

Afin de simplifier l'installation de notre application pour l'autre équipe, nous avons déployé notre application sur une machine virtuelle hébergée chez AWS (Amazon Web Service). Ainsi, il a suffi aux membres de l'autre équipe de se rendre sur http://polyquiz.me/ pour avoir accès à notre application. Nous avons aussi écrit une notice d'installation pour les personnes souhaitant l'installer.

Pour permettre de mieux guider les utilisateurs, nous avions préalablement repris nos personas avec des scénarios (cf. partie 1) dans lequel sont décrit différentes tâches à réaliser. Ainsi les membres de OK Google devaient pouvoir juger la facilité et l'intuitivité avec laquelle ils pouvaient utiliser notre application. Ces scénarios nous ont aussi permis de ne pas oublier certaines fonctionnalités.

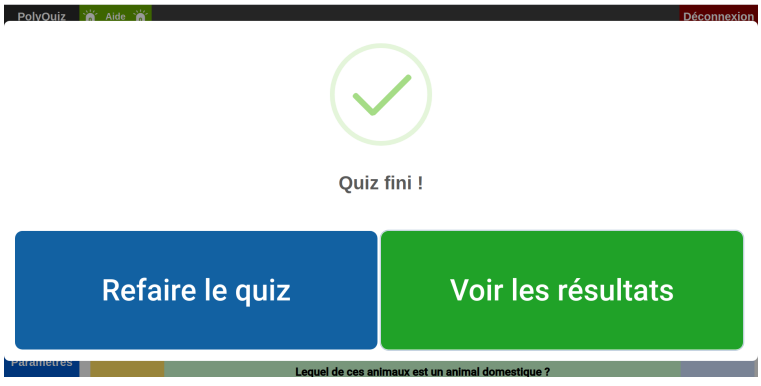
Nous avons préparé un questionnaire en trois parties (partie résidents, partie administration, questions générales) afin de faciliter les retours de l'équipe "OK Google" concernant notre projet.

3.1 La partie résidents

Nous avons commencé l'évaluation avec les scénarios concernant la partie résidents. L'équipe qui nous évaluait a apprécié nos choix concernant le système de connexion pour les résidents (nom + prénom, en ne tenant pas compte de la casse ni du nombre d'espaces pouvant être ajoutés par inadvertance par l'utilisateur). Ils ont également apprécié les paramètres adaptés aux utilisateurs et les ont jugés suffisamment accentués. Ils ont également apprécié le menu d'aide (comportant les vidéos d'aide).

Concernant les améliorations que nous pourrions apporter à notre projet, ils nous ont suggéré plusieurs modifications que nous avons mis en place.

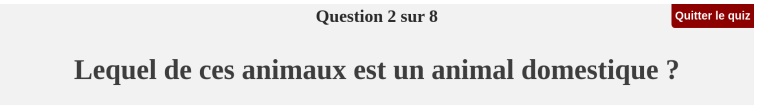
Dans un premier temps, ils ont remarqué que l'utilisateur n'avait aucun retour sur le fait qu'il ait répondu à une question, ni lorsqu'il cliquait sur certains boutons qui ont un impact sur sa navigation. Ainsi, nous avons utilisé la bibliothèque "SweetAlert2" pour ajouter des demandes de confirmation à l'utilisateur lorsqu'il répond à une question, mais aussi lorsqu'il veut quitter un quiz en cours de route ou l'abandonner totalement. Nous avons aussi ajouté une notification pop-up à la fin d'un quiz permettant à l'utilisateur de choisir entre voir ses résultats et refaire le quiz.



Demande de confirmation

Dans un second temps, l'utilisateur ne savait pas réellement son avancée dans le quiz, c'est-à-dire qu'il ne connaissait pas en temps réel le nombre de question effectuées et le nombre de questions totales. Nous avons

donc fait en sorte que lorsque l'utilisateur effectue un quiz, le numéro de la question soit affiché en haut de l'écran (par exemple "Question 3/5").



Enfin, suite aux remarques reçues lors de cette évaluation, nous avons fait en sorte de rendre plus claire la navigation de l'utilisateur en modifiant plusieurs détails. Nous avons par exemple modifié le système de filtres pour qu'on ne puisse choisir qu'une seule difficulté à la fois. L'utilisateur est maintenant informé qu'il est devant une liste de quiz lorsqu'il sélectionne un thème, et ajouté un message informant qu'il n'y a pas de quiz disponible lorsqu'on filtre selon une difficulté et qu'aucun résultat n'est disponible.

3.2 La partie administration

Nous avons ensuite continué l'évaluation croisée avec les scénarios concernant la partie administrateurs. L'équipe "OK Google" a apprécié l'interface permettant aux aides-soignants de créer et de paramétrer un compte utilisateur, et notamment la possibilité de partager les paramètres entre plusieurs résidents. Ils ont également trouvé que la création de quiz était rapide et intuitive.

De même que pour la partie résidents, ils nous ont suggéré plusieurs améliorations que nous avons mis en place.

D'abord, ils ont remarqué que si l'administrateur souhaitait supprimer un quiz, l'opération était plutôt longue (sélection du quiz, clic sur le bouton d'édition, onglet modification des questions, clic sur le bouton de suppression). Nous avons donc facilité cette tâche en mettant un bouton de suppression à côté du bouton d'édition du quiz (sélection du quiz, clic sur le bouton de suppression).

De plus, lors de l'évaluation, une confusion était souvent faite dans le menu latéral entre "création de quiz" et "création d'un résident". Le titre de cet onglet a alors été modifié pour éviter cette confusion à l'avenir.

Enfin, concernant la création de quiz, en suivant les conseils de nos camarades nous avons ajouté un message prévenant l'administrateur qu'il doit créer au minimum deux questions dans un quiz, mais aussi mis en valeur le bouton d'ajout des questions, et ajouté plusieurs demandes de confirmation avant de faire certains choix importants ou irréversibles (validation des questions, suppression des questions, abandon de la création de quiz avant de l'avoir enregistré).

L'évaluation croisée nous a donc permis non seulement de connaître les points forts de notre site, mais également d'améliorer des détails souvent essentiels qu'on ne remarque pas forcément lorsqu'on développe un projet. Le point de vue extérieur qui nous a été donné a donc été essentiel en termes d'ergonomie.

4 Conclusion perspective

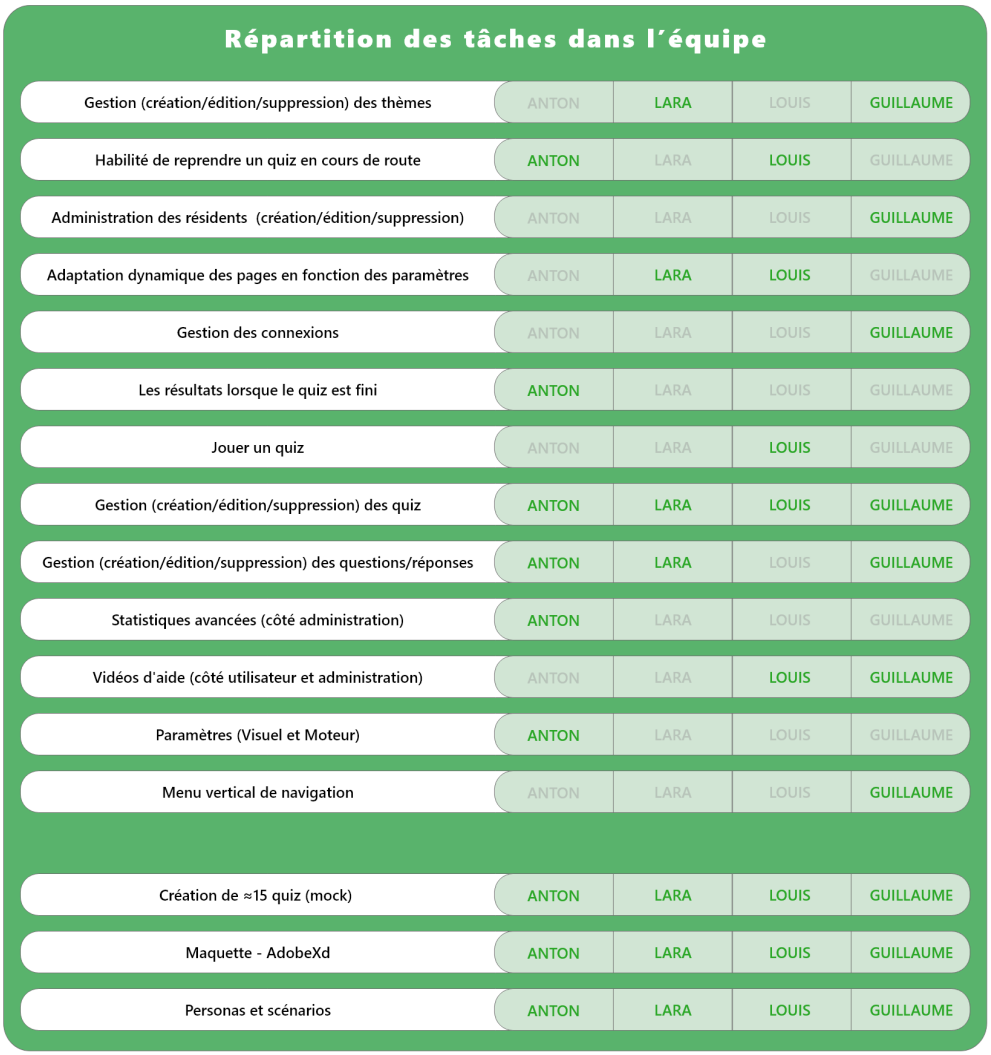
Pour conclure, nous avons tous appris beaucoup de choses durant ce projet, tant sur le plan technique, avec la découverte d'Angular, que sur le plan organisationnel. C'est le premier projet que nous avons réalisé dans lequel nous avons dû définir les besoins de l'utilisateur et créer des solutions adéquates. Nous avons tous les quatre apprécié ce travail d'ingénieur et cette liberté que nous avons eu lors de l'élaboration de la maquette. C'est d'ailleurs en respectant scrupuleusement cette dernière que nous avons pu développer convenablement notre site web.

C'est auprès de personnes âgées de notre entourage et à distance (COVID19) que nous avons pu vérifier si nous avions correctement identifier les besoins. Certaines choses comme les couleurs ou la taille de la police ont ainsi évolué grâce à cela.

Nous avons rapidement compris l'enjeu d'utiliser Angular dans le cadre de ce projet ainsi que les avantages de ce dernier. C'est une technologie qui comporte de nombreux avantages et qui nous a permis par exemple d'éviter la redondance de code, mais aussi de nous faciliter la répartition du travail grâce à la distinction entre les composants, les services et le serveur de données. De plus, c'est une technologie innovante et relativement récente qui possède une communauté très active avec beaucoup de bibliothèques gratuites qui ont facilité notre travail.

Après chaque soutenance, nous avons tenu compte des remarques qui nous ont été faites et nous avons modifié le projet en conséquence. Aujourd'hui notre projet répond à tous les besoins des utilisateurs que nous avons identifiés et c'est pourquoi nous espérons que notre projet sera retenu pour être utilisé dans un EHPAD. Nous y avons passé beaucoup de temps, et nous avons essayé de respecter au mieux les "normes de code" imposées par Angular. En essayant par exemple d'utiliser les composants et les services de manière consciencieuse, cela nous a permis d'obtenir un code évolutif. Nous pouvons rajouter d'autres fonctionnalités simplement sans toucher aux composants actuels pour permettre par exemple de gérer d'autres type de handicaps.

5 Annexes



Bibliographie

Documentation Angular : https://angular.io/docs

- Directives
- Pipes
- Services
- Requêtes HTTP avec Angular
- NgOnInit
- @Input et @Output

SweetAlert2 : https://sweetalert2.github.io/