Vannes
iut
Université Bretagne Sud

Spécifications fonctionnelles de l'application MathsALaMaison

TABLE DES MATIERES

CADRE DU PROJET

RESUME DU PROJET

Le projet MathsALaMaison consiste à développer une application ludique compatible avec tous les smartphones. L'objectif est que les personnes apprennent les mathématiques de façon ludique et partout. L'application MathsALaMaison a pour but d'être publié sur les différents store mobile (Apple Store et Google Play Store).

PROFIL CIBLE DE L'APPLICATION

L'application MathsALaMaison vise toutes personnes voulant apprendre les mathématiques de façon ludique et compétitive. Le public ciblé est tout de même les collégiens entre le niveau 4ème et 3ème.

LIVRABLE

A la fin de ce projet nous allons délivrer les fichiers sources de l'application avec la liste des démarches pour pouvoir mettre en ligne l'application MathsALaMaison. Lors de grande avancée, nous allons communiqué une version testable à la cliente via la plateforme Expo Go.

EQUIPE

Le projet est réalisé par une équipe de 4 étudiants encadrés par des professeurs.

Les étudiants :

- **GÜNES Omer** : Scrum Master, développeur full-stack et responsable du développement du serveur de jeu.
- **MELLAH Rayanne** : Développeur back-end, responsable de la logique et des mécaniques de jeu.
- **PARCOLLET Noé** : Développeur front-end.
- **PÉRON Romain** : Responsable du développement front-end.

Les professeurs :

- **EVENAS Hélène** : Tutrice
- **BAUDONT Pascal** : Responsable SAE
- **LE SOMMER Nicolas** : Responsable SAE

CHARTE GRAPHIQUE

Pour la charte graphique de l'application, on s'est inspiré des jeux déjà existants, notamment comme : Brawl Stars, Fall Guys, Clash Royale, Fortnite...

Notre principale idée en nous inspirant des grands éditeurs de jeux était que, eux, ils ont de l'expérience, ils ont des équipes entières pour optimiser et offrir la meilleure expérience utilisateur possible (exemple d'inspiration : le bouton "jouer" en bas à droite sur tous les jeux, le profil en haut, les paramètres en haut, etc...).

Le développement de la charte graphique a pris du temps, on a d'abord réalisé tout nos pages sur Figma, on a passé du temps en équipe pour avoir une interface qui nous plaît tous d'abord, car on ne veut pas livrer une application avec une interface qui ne nous plaît pas.

On a surtout hésité sur les différentes couleurs dominantes de l'application, surtout les couleurs pour les fonds des différentes pages.

Voici quelques captures d'écrans des différentes versions de la maquette :

Figure 2 - Première version de la page inscription

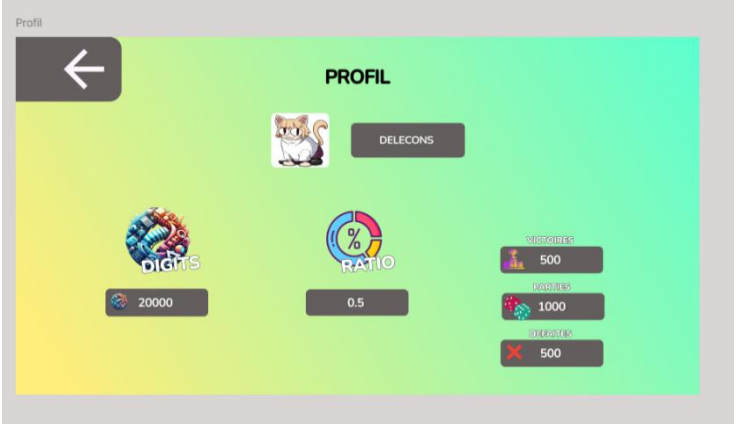


Figure 1 - Maquette de la page de profil

FONCTIONNALITES DE L'APPLICATION

AUTHENTIFICATION ET GESTION DES UTILISATEURS

- **Création de compte utilisateur :** Permet aux utilisateurs de créer un compte avec un email, pseudonyme et mot de passe.
    - **Contrainte :** Chaque compte doit avoir un email et un pseudonyme différent.
- **Connexion / Déconnexion :** Accès sécurisé à l'application via un email ou pseudonyme et un mot de passe. Et déconnexion de l'application si l'utilisateur le souhaite.
- **Suppression de compte :** Suppression du compte possible à n'importe quel moment.
- **Réinitialisation de mot de passe :** Si l'utilisateur oublie son mot de passe il peut le réinitialiser depuis Connexion > Mot de passe oublié. Un code va alors lui être envoyé par mail.
- **Gestion de compte :** L'utilisateur peut à tout moment changer son mot de passe, son email et son pseudonyme.
- **Photo de profil :** L'utilisateur peut à tout moment changer sa photo de profil avec une autre qu'il a au préalable déjà débloquées.

GAMIFICATION DE L'APPRENTISSAGE

- **Digits :** Système de points pour que les utilisateurs se dépasse.
- **Déblocage de cosmétique :** A chaque palier de points, les utilisateurs ont la possibilité de débloquer des photos de profils. Les différents paliers pour débloquer les avatars sont : (100, 125, 150, 190, 230, 280, 350, 452, 520, 640, 790, 960, 1200, 1450, 1790, 2190, 2700, 3300, 4070, 4500, 5000 et 6000 Digits)
- **Classement Générale :** Classement générale permettant d'avoir de la compétition entre les utilisateurs.
- **Musique :** La musique ajoute un aspect plus ludique dans le jeu, avec différents choix de musique.
- **Profil Utilisateur :** Permet de voir ses statistiques globales depuis la création de son compte. Il comprend les données tel que le nombre de parties jouée, le nombre de victoires et de défaites, un ratio sur l'ensemble des parties et le nombre de digits.

ACCESSIBILITE

- **Multiplateforme :** L'application est accessible via la majorité des plateformes mobiles, il est supporté par Android et iOS ce qui permet d'être lancé via n'importe quel appareil récent.
- **Retour Haptique :** Les retours haptiques surviennent lorsque l'utilisateur fait une action ou alors qu'il doit réagir (son tour est venu, confirmation d'une action…)

JEUX

- **Matchmaking :** Ecran d'attente dans laquelle l'utilisateur va attendre jusqu'à qu'assez de joueur rejoigne pour qu'une partie puisse se créer. Dans le futur nous prévoyons d'ajouter un SBMM dans le matchmaking.
- **Mode en ligne :** Le mode en ligne permet de jouer une partie avec d'autre personne depuis n'importe où. Vous lancez une partie, vous attendez que suffisamment de joueur arrive et la partie se lance.
- **Mode local :** Le mode local permet de jouer à plusieurs avec un seul appareil.
- **Mode entrainement :** Le mode entrainement permet de s'entrainer sur les 600 questions présentes dans la base de données, une question va apparaître et l'utilisateur devra y répondre correctement. Une alerte s'affichera en disant si la réponse est bonne ou non.

TECHNOLOGIES UTILISES

Nous utilisons différentes technologies pour le développement de ce projet. Les principaux langages et cadriciels sont :

- **React Native :** Utilisée dans le développement front-end de l'application mobile.
- **TypeScript :** Utilisée dans le développement back-end de l'application mobile.
- **NodeJS :** Utilisée pour la création des APIs pour communiquer avec la base de données mais également pour le serveur de jeu.
- **MySQL :** Utilisée pour stocker de façon permanente les utilisateurs de l'application et les questions.

PARCOURS UTILISATEUR (UX)

Tous les parcours qu'un utilisateur peut prendre est dans le diagramme ci-dessous. Le PDF est également disponible sur notre Notion.

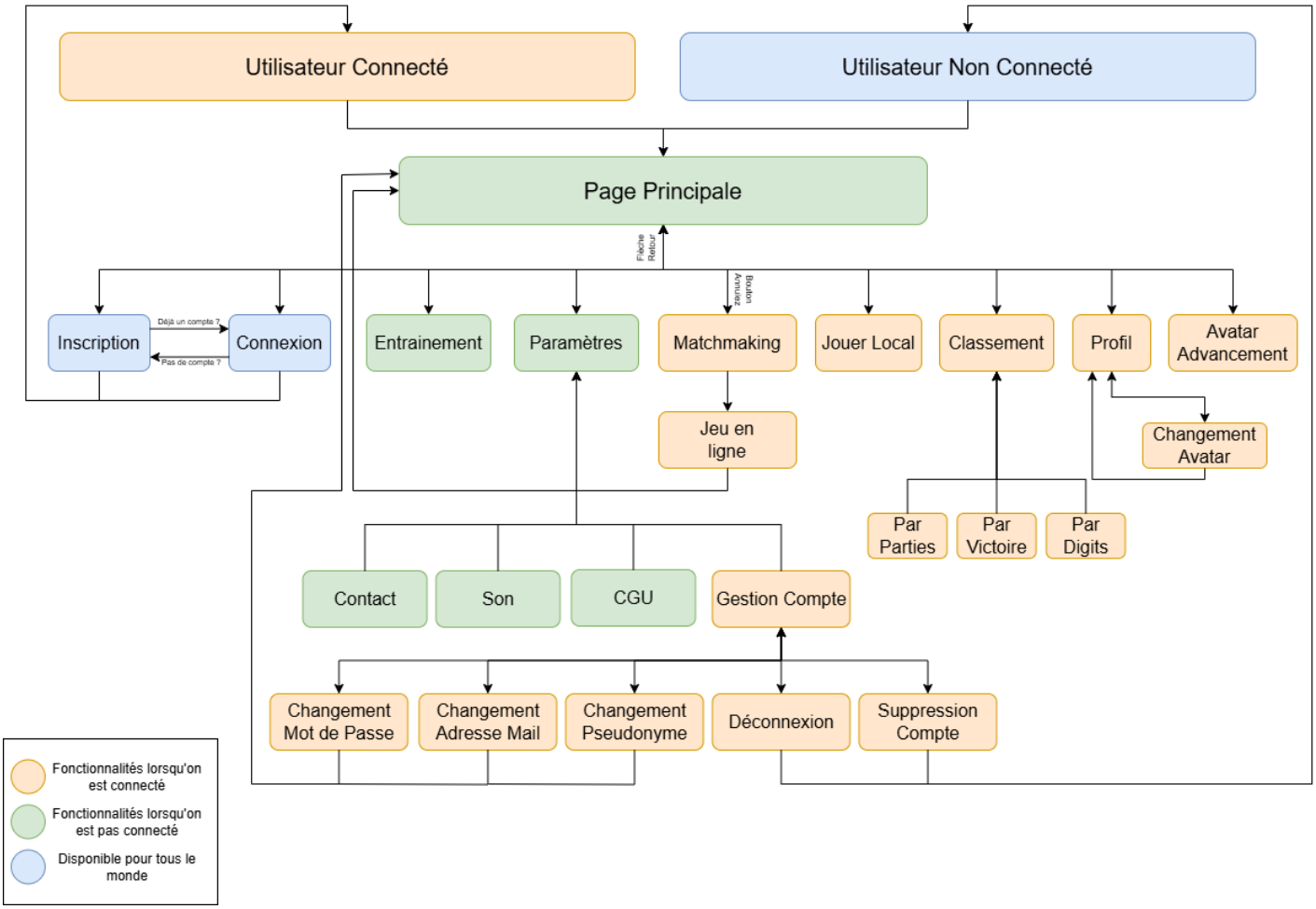


Figure 3 - Schéma parcours utilisateur

BACK-END

FONCTIONNEMENT GENERALE

L'application fonctionne en appelant un **serveur distant**. Ce serveur est chargé de gérer plusieurs aspects essentiels. Il gère notamment, la connexion à la base de données, la création de salle de jeu, la synchronisation client, génération de fichier de journalisation pour conserver un historique détaillé des actions.

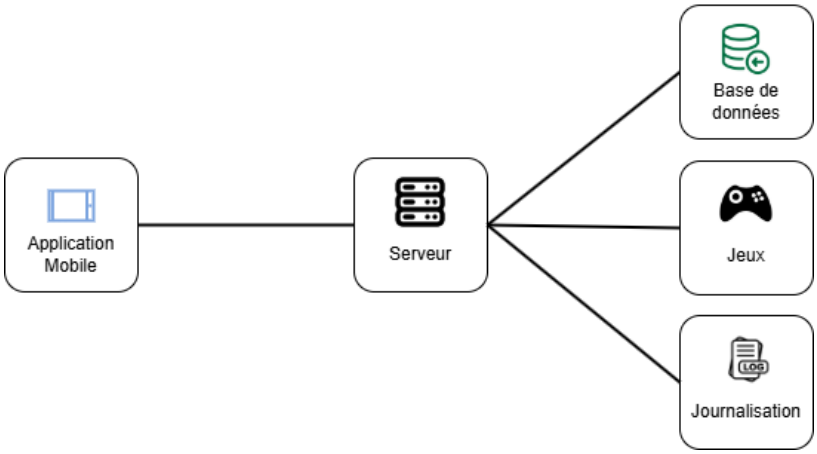


Figure 4 - Fonctionnement générale de l'application

GESTION DES DONNEES

Nous gérons les données de deux façons différentes : localement sur l'appareil de l'utilisateur et via un stockage permanent sur un serveur distant.

1. **Stockage local : « variable de session » :** Les variables de session sont utilisés pour conserver les données de l'utilisateur sur son appareil. Ils permettent de recharger ces données lorsque l'utilisateur quitte puis revient dans l'application.

   Les données stocker sont : « pseudo, email, nombre de point, nombre de partie, nombre de victoire, photo de profil, musique ». La **photo de profil** est stockée sous forme d'un nombre entre 1 et 30 inclus. La variable **musique** contient l'état de la musique, si il est à vrai, alors la musique jouera lorsque l'utilisateur se reconnecte, si il est à faux alors la musique ne jouera pas.

2. **Stockage permanent : Base de données MySQL :** Nous utilisons une base de données MySQL hébergée sur un serveur distant pour stocker de manière permanente les informations importantes. L'application

communique avec cette base de données via un serveur Node.js, qui agit comme un pont entre l'application et la base de données. La base de données est composée de deux tables, une table **User** et une table **Questions**.

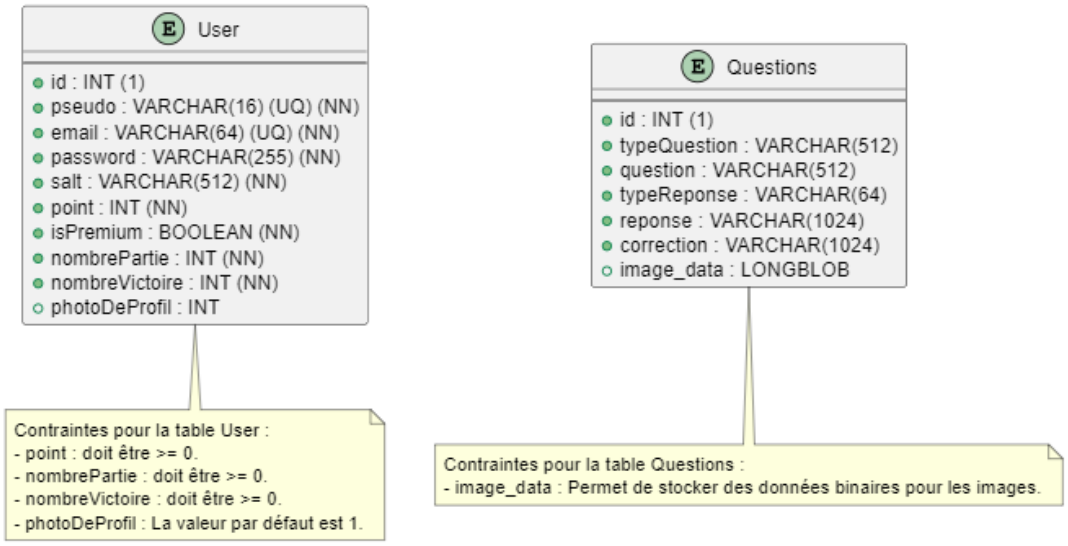


Figure 5 - Diagramme de la base de données

SERVEUR

Le serveur est le noyau central du projet MathsALaMaison. Il assure la gestion des interactions entre les utilisateurs, la base de données et différents services.

Le serveur est composé en 9 parties.

1. Configuration

La partie configuration contient quatre fichiers, ils permettent de changer les configurations du serveur. Les quatre fichiers sont :

- **database.js** : Contient la configuration de connexion à la base de données MySQL.
- **logger.js** : Contient la configuration pour le système de journalisation (log).
- **gameAttribute.js** : Contient les attributs liés au jeu, comme le nombre de point que le gagnant va gagné, le nombre de joueur par partie…
- **globalAttribute.js** : Contient les paramètres globaux de l'application comme le temps de validité d'un code de validation.

2. Controller

La partie controller contient 3 fichiers qui sont utiles pour faire des traitements. Les trois fichiers sont :

- **gameController** : Gère toute la logique concernant le jeu (les cliques reçue par les utilisateurs, génération d'une nouvelle partie, lorsqu'un joueur quitte, stockage des différentes salles de jeu)
- **matchmakingController** : Gère toute la partie avant le jeu, mettre en relation des joueurs cherchant une partie et puis envoyer les joueurs dans gameController pour qu'il créer une partie.
- **playerController** : Gère tout ce qui est post-partie, l'insertion des points gagnés, victoires pour le vainqueur, partie joué pour tous le monde dans la base de données.

3. Logs

La partie logs contient tous les logs du serveur. Conformément au fichier de configuration. Avec notre configuration actuelle un fichier reste sous la forme d'un fichier, puis à la fin de la journée le fichier est archivé pour prendre moins de place puis un autre fichier **.log** à la nouvelle date du jour est créer.

```
[2024-12-18 13:55:08] INFO: Romain ajouté à la liste d'attente.
[2024-12-18 13:55:09] INFO: Olimer a arrêté de chercher une partie.
[2024-12-18 13:55:09] INFO: [18/12/2024 13:55:09] GET request from ::ffff:37.165.113.17 to /users
[2024-12-18 13:55:09] INFO: [18/12/2024 13:55:09] POST request from ::ffff:37.165.113.17 to /checkOnlinePlayer
[2024-12-18 13:55:09] INFO: [18/12/2024 13:55:09] GET request from ::ffff:37.165.113.17 to /users
[2024-12-18 13:55:09] INFO: [18/12/2024 13:55:09] POST request from ::ffff:37.165.113.17 to /checkOnlinePlayer
[2024-12-18 13:55:13] INFO: Olimer ajouté à la liste d'attente.
[2024-12-18 13:55:13] INFO: Création de la partie avec l'ID game-1734526513606 et les joueurs : Romain, Olimer
[2024-12-18 13:55:13] INFO: [18/12/2024 13:55:13] GET request from ::ffff:37.165.113.17 to /numberOfPlayer
[2024-12-18 13:55:20] INFO: Le nombre de partie joué de Olimer a été mis à jour.
[2024-12-18 13:55:20] INFO: Le score du joueur Romain a été mis à jour.
[2024-12-18 13:55:20] INFO: Le nombre de partie du joueur Romain a été mis à jour.
[2024-12-18 13:55:20] INFO: Le nombre de partie joué de Romain a été mis à jour.
[2024-12-18 13:55:20] INFO: La partie game-1734526513606 est terminée. Romain a gagné !
[2024-12-18 13:55:20] WARN: La partie game-1734526513606 n'existe pas.
[2024-12-18 13:55:21] INFO: [18/12/2024 13:55:21] POST request from ::ffff:37.165.113.17 to /checkOnlinePlayer
[2024-12-18 13:55:21] INFO: [18/12/2024 13:55:21] GET request from ::ffff:37.165.113.17 to /users
```

Figure 6 - Exemple d'un fichier de log

4. Middlewares

La partie middlewares contient un seul fichier, le fichier **apiKeyMiddleware** il permet de vérifier les connexions entrantes et de vérifier l'entête de chaque requêtes, si la requêtes contient un champ « **x-api-key** » avec la bonne clé le serveur va y répondre à la requêtes mais si il ne contient pas la bonne clé API ou

que le nom de champs (**x-api-key**) est erroné alors le serveur va répondre :
« **Accès refusé : clé API invalide** »

5. **Routes**

Pour avoir des fichiers plus lisible et maintenable nous avons opté pour créer des fichiers différents pour chaque type de routes. Il y a quatre différents fichiers de routes :

- **userRoutes** : Gère toutes les routes liées à la gestion des utilisateurs (inscription, connexion, suppression compte…)
- **questionRoutes** : Gère toutes les routes relatives aux questions.
- **validationCodeRoutes** : Contient toutes les routes qui permettent de générer et de valider les codes de validations
- **miscelaniousRoutes** : Regroupe différentes routes, qui ne sont pas assez nombreux pour y faire des fichiers distinctes.

6. **Services**

Le serveur contient deux principaux services qui sont :

- **WebSocket** : Permet de créer des connexions avec les différents utilisateurs et de gérer les requêtes de ceux-ci.
- **MailService** : Permet l'envoie de mail de validation aux utilisateurs via le mail indiqué dans le fichier .env

7. **Utilitaire**

L'application compte trois fichiers de type utilitaire :

- **smsAlert** : Permet l'envoie de SMS en cas d'erreur du serveur via l'API de l'opérateur Free Mobile.
- **consoleCommands** : Contient différentes commandes que l'administrateur du serveur peut effectué. Les commandes utilisables sont disponible en écrivant « **help** » dans le terminal.
- **color** : Permet d'afficher des lignes dans le terminal de façon coloré.
- **asciiArt** : Permet de faire de faire du texte de façon ASCII Art cela ajoute un meilleur aspect au serveur.

Figure 7 - Ecran de démarrage du serveur

8. Fichier d'environnement

Le fichier d'environnement permet de stocker des variables qu'on va utiliser dans le code, il est utile pour ne pas écrire les mots de passes ou clé API directement dans le code source. Le fichier d'environnement du serveur est composé comme ci-dessous.

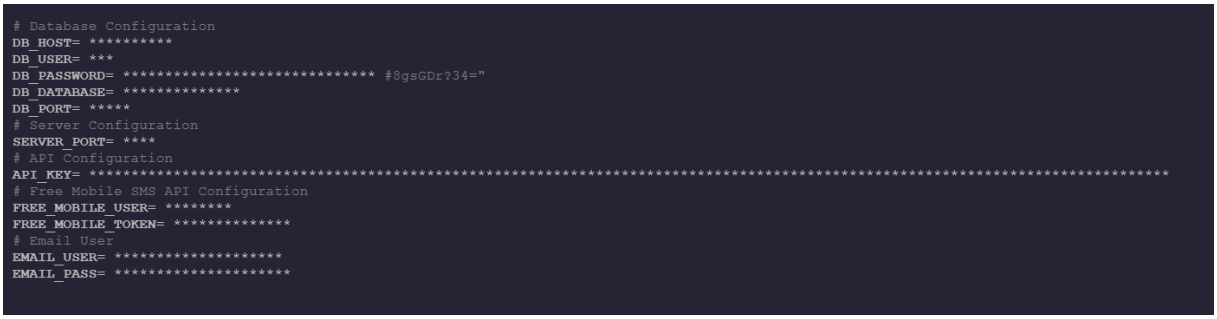


Figure 8 - Fichier d'environnement

DB_HOST : Adresse du serveur où la base de données est hébergée. Cela permet au serveur de savoir où se connecter à la base de données.
(Exemple : apple.com, mathsalamaison.fr, 67.254.25.6...)
DB_USER : Identifiant utilisé pour se connecter à la base de données
DB_PASSWORD : Mot de passe associé à l'utilisateur pour accéder à la base de données.
DB_DATABASE : Nom de la base de données spécifique à utiliser pour l'application.
DB_PORT : Port sur lequel le serveur de base de données écoute les connexions.
SERVER_PORT : Port sur lequel le serveur Node.js va écouter les requêtes.
API_KEY : Clé API utilisée pour sécuriser les requêtes et permettre uniquement à l'application MathsALaMaison de communiquer avec le serveur. La clé API est :
heIVcMzabVMce8Fmbj5QuuoYRmvGgiGqtEhjoZlHPUdRK9fScY7c8A9LGH07wDzx4FEEY8DPaDDA2ZjK3ZI85eEwuvRCRAgqrc3BrUb9YWtiMCKTp3D8n5AM9x0nImfL
Il faut envoyer la clé API dans le header de la requête avec comme balise : [x-api-key].

FREE_MOBILE_USER : Identifiant utilisateur Free Mobile permettant d'envoyer des SMS pour des alertes en cas de problème.
FREE_MOBILE_TOKEN : Token associé à l'utilisateur Free Mobile, utilisé pour l'authentification lors de l'envoi de messages via SMS.
EMAIL_USER : Adresse e-mail utilisée pour envoyer les codes de validation aux utilisateurs.
EMAIL_PASS : Mot de passe associé à l'adresse e-mail utilisée pour envoyer des messages.

UTILISATION DU SERVEUR

Actuellement le serveur de MathsALaMaison tourne sur un serveur sous Ubuntu mais il peut être lancer avec n'importe quelle machine du moment que NodeJS peut y être installé. La partie conseil concerne les personnes installant le serveur sous Linux.

CONSEIL

Une fois que vous avez télécharger le code source du serveur MathsALaMaison, nous vous conseillons de le lancer avec « tmux », tmux va vous permettre de créer un terminal virtuelle, vous pourrez alors quitter le terminal sans que le processus qui s'y trouve s'arrête. Voici donc quelque commandes TMUX :

- **Créer un terminal tmux :** tmux new -s « nomDeLaFenêtre »
- **Voir vos terminal actifs :** tmux ls
- **Retourner dans le dernier terminal actifs :** tmux a
- **Quitter le terminal sans tuer les processus :** CTRL + B puis D
- **Quitter le terminal en le supprimant :** exit
- **Ouvrir un terminal spécifique :** tmux attach-session -t « nomDeLaFenêtre »

COMMANDES

Pour démarrer le serveur vous pouvez utilisé la commande « **./start** » lorsque vous êtes dans le répertoire MathsALaMaison. Si vous avez cette page qui apparait, cela veut dire que le serveur a démarré avec succès.

Figure 9 - Ecran de démarrage du serveur

Chaque appelle qui arrive sur le serveur va alors être affiché sous cette forme :

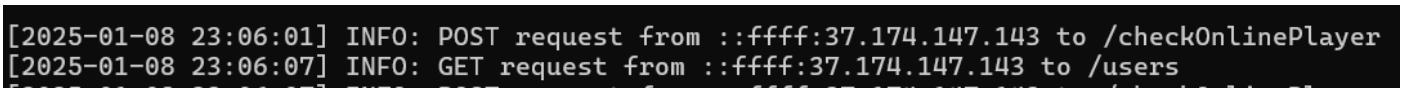
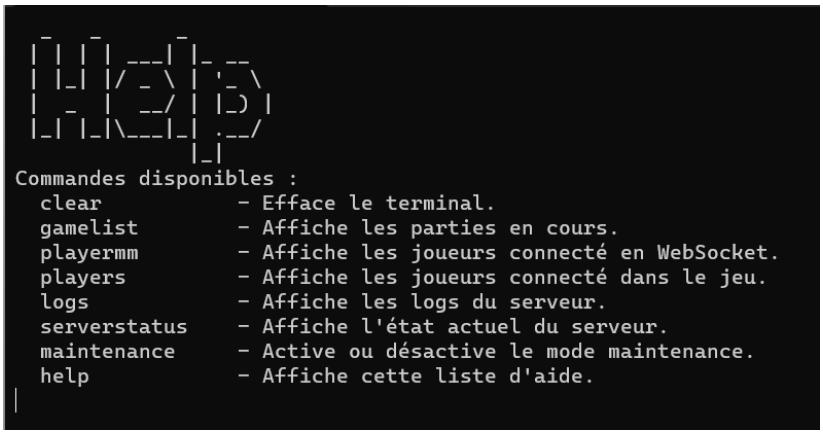


Figure 10 - Exemple de log

[DATE HEURE] [Type du log] : [Type de requête] [provenance de la requête] [routes API]

Vous pouvez également tapez des commandes dans le terminal. Les commandes permettent de contrôler différents aspect du serveur. Les différentes commandes sont :

- **help :** Permet d'afficher l'ensemble des commandes utilisables



- **clear :** Permet d'effacer l'écran du terminal

- **gamelist :** Permet de voir toutes les parties en cours

Figure 11 - Partie en cours

- **playermm :** Permet de voir toutes les personnes connectés via WebSocket

Figure 12 - Liste des joueurs connecté en WebSocket

- **players :** Permet de voir tous les joueurs avec leurs dernières date de connexion. La liste se réinitialise à chaque redémarrage du serveur.

Figure 13 - Liste des joueurs

- **logs :** Permet d'avoir une page vide pour afficher les logs.

Figure 14 - Page de log

- **serverstatus :** Permet d'avoir les différentes informations concernant le serveur comme la durée de fonctionnement, la charge CPU et la mémoire utilisé.

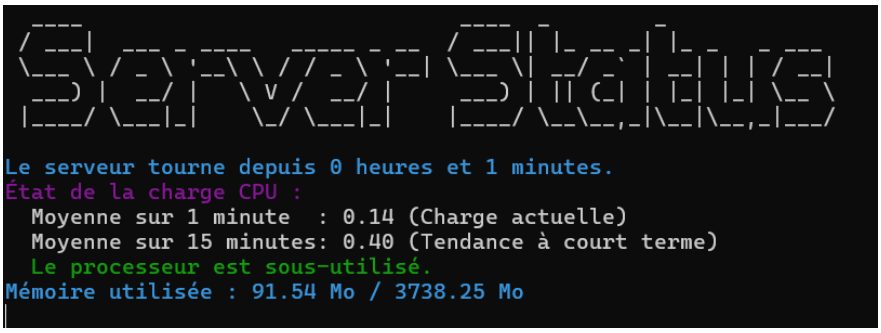


Figure 15 - Informations utiles concernant le serveur

- **maintenance :** Le mode maintenance est une fonctionnalité côté serveur, il permet de couper l'application pour effectuer des maintenances. Pour l'instant il n'a aucun effet sur l'application, il est juste implémenté côté serveur.

Figure 16 - Maintenance activé

Figure 17 - Maintenance désactivé

DOCUMENTATION

Nous documentons le projet MathsALaMaison afin de faciliter sa compréhension et sa gestion. Il y a deux parties de documentation, la première concernant l'application mobile et le second concernant le serveur.

1. **L'application :**
    a. La documentation de l'application MathsALaMaison se trouve directement dans le code source, disponible sur notre GitLab.
    b. Un document expliquant le fonctionnement de l'application se trouve également sur notre Notion.
2. **Le serveur :**
    a. La documentation du code source se trouve sur le site « omergs.com:50000/documentation »
    b. La documentation plus générale des fonctionnalités du serveur se situe sur notre GitLab.

APPLICATION

ORGANISATION

La partie application du projet MathsALaMaison contient différents répertoires et fichier pour le bon fonctionnement de celle-ci.

Les dossiers sont dans l'ordre :

- **app :** Répertoire contenant toute la partie graphique de l'application, le répertoire « **(tabs)** » va contenir toutes les pages, le fichier « **+not-found.tsx** » est la page qui sera affiché si une page redirige vers une page qui n'est pas trouvé. Les deux autres fichiers permettent de gérer correctement l'affichage des pages.
- **assets :** Répertoire qui contient toutes les différentes ressources utilisé par l'application.
    - **fonts :** Répertoire qui contient les polices d'écritures de l'application.
    - **images :** Répertoire qui contient les différentes images.
        - **icones :** Les différentes icônes qui seront utilisées pour symboliser les catégories sur le plateau de jeu.
        - **logo :** Logo de l'application
        - **profile-picture :** Les différentes photos de profils disponibles dans l'application.
            - **default :** Les photos de profils disponibles dès la création d'un compte (par défaut).
            - **exclusive :** Les photos de profil exclusives qui sont délivrées spécialement (à la fin du pass)
            - **road :** Les photos de profils déblocable dans la page avancement des avatars (autrement appelé pass).
    - **music :** Répertoire qui contient les différentes musique qui sont joué dans l'application.
- **components :** Répertoire contenant le code back-end de l'application. Il contient des répertoires et quelque fichiers. Les fichiers sont « **ActionPlaySet.ts** » et « **BoardType.ts** » ils sont utiles pour la gestion des cartes actions et du jeu. Les autres répertoires présent sont :
    - **__tests__ :** Contient tous les fichiers relatives aux tests.
    - **controller :** Contient des fichiers permettant de faire le lien entre la vue et le model.

    - **model :** Contient des fichiers dédiées à la gestion de données. On les utilise pour créer des instances.
    - **navigation :** Dossier par défault d'un projet React-Native
    - **util :** Répertoire contenant des fichiers utilitaires (comme password-util)
- **constant :** Dossier par défault de ReactNative, contient les couleurs des modes claires et sombres.
- **scripts :** Contient un fichier pour réinitialiser le projet.
- **properties :** Répertoire contenant des propriétés utile au bon fonctionnement de l'application
    - **API_KEY.ts :** Contient la clé API qui permet de communiquer avec le serveur distant.
    - **BACKEND_API.ts :** Contient l'url pour effectuer une connexion HTTP avec serveur distant.
    - **WEBSOCKET_API.ts :** Contient l'url pour effectuer une connexion Websocket avec le serveur distant.

INTERFACE UTILISATEUR

PAGE D'INSCRIPTION

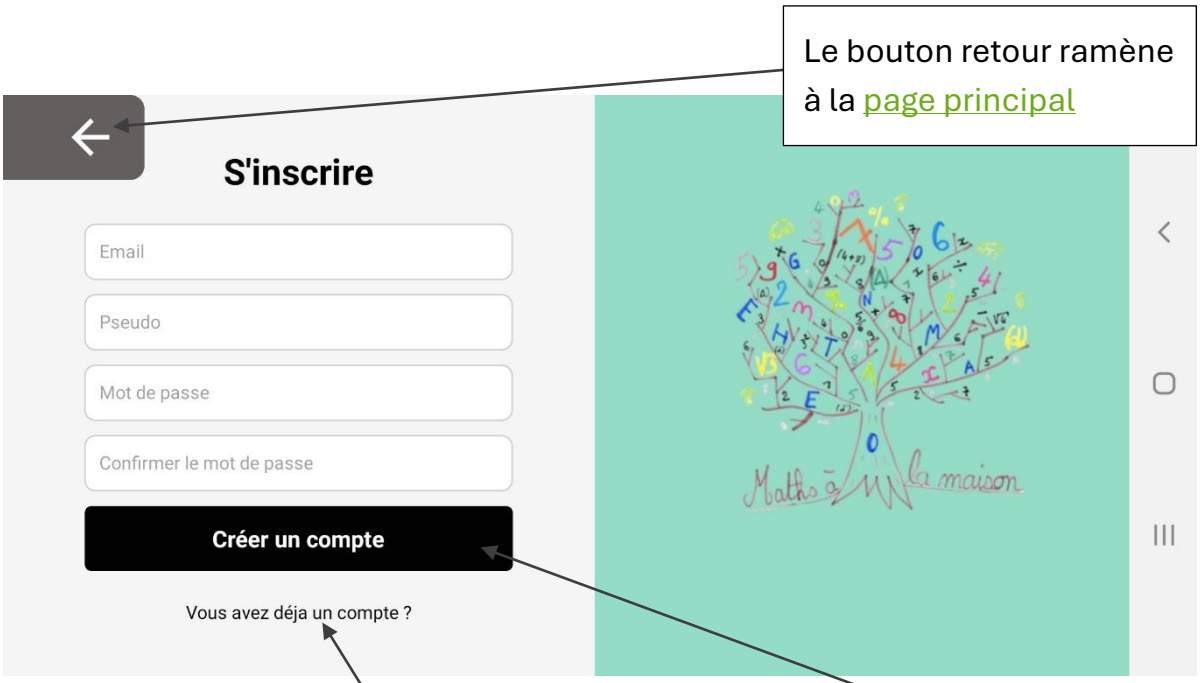


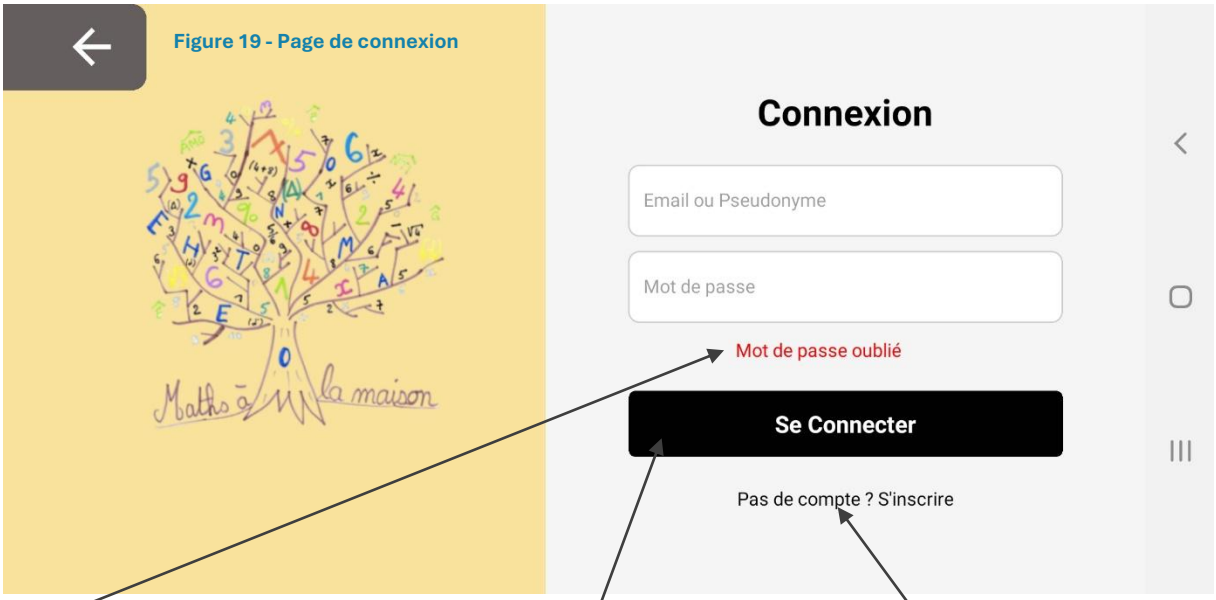
Figure 18 - Page d'inscription

La page inscription, permet de se créer un compte dans l'application. Pour cela il est demandé à l'utilisateur des informations basique comme son adresse email, son pseudonyme et un mot de passe, l'utilisateur doit ensuite confirmer son mot de passe afin de pouvoir finaliser la création de son compte. Chaque compte utilisateur doit avoir un email unique et valide dans le format « test@exemple.fr » et un pseudonyme unique (appartenant à aucun autre compte utilisateur).

Le pseudo de l'utilisateur doit faire entre 4 et 16 caractères **inclus.** Une fois que les conditions sont rempli alors on envoie au serveur le nouvel utilisateur via la route **« /register ».** Puis on enregistre toutes les valeurs dans les variables de sessions. Une fois que ces étapes sont finis, on redirige l'utilisateur vers la page principale.

L'utilisateur peut toujours revenir dans la page principale avec la flèche de retour présent en haut à gauche de la page. L'utilisateur peut également se rendre dans la page de connexion avec le bouton « **Vous avez déjà un compte ?** »

PAGE DE CONNEXION



Ce bouton permet de recréer un mot de passe

Ce bouton permet de confirmer la connexion

Ce bouton ramène à la page inscription

La page de connexion permet à l'utilisateur de se connecter à un compte déjà existant. Pour cela, l'utilisateur a le choix entre se connecter avec son adresse e-mail ou avec son pseudonyme puis d'entrer son mot de passe.

Une fois que l'utilisateur appuie sur « Se Connecter » on va récupérer tous les utilisateurs depuis la base de données, on va vérifier que l'identifiant entré par l'utilisateur correspond à un email ou à un pseudonyme, si cela ne correspond à aucun des deux on va renvoyé l'alerte « **L'email ou le pseudo n'existe pas.** » Si l'identifiant est connu, on va envoyé l'identifiant et le mot de passe au serveur qui va se charger de vérifier si le mail et le mot de passe sont correct, si cela c'est correct on renvoie un booléen égale à vrai. On va ensuite trouver récupérer l'instance de l'utilisateur grâce à son identifiant, puis on va récupérer tous les attributs pour les mettre dans les variables de session. On va ensuite rediriger l'utilisateur vers la page d'accueil.

L'utilisateur peut toujours revenir dans la page principale avec la flèche de retour présent en haut à gauche de la page. L'utilisateur peut également se rendre dans la page d'inscription avec le bouton « **Pas de compte ? S'inscrire** ».

L'utilisateur peut réinitialiser son mot de passe via le bouton « Mot de passe oublié ».

MOT DE PASSE OUBLIE

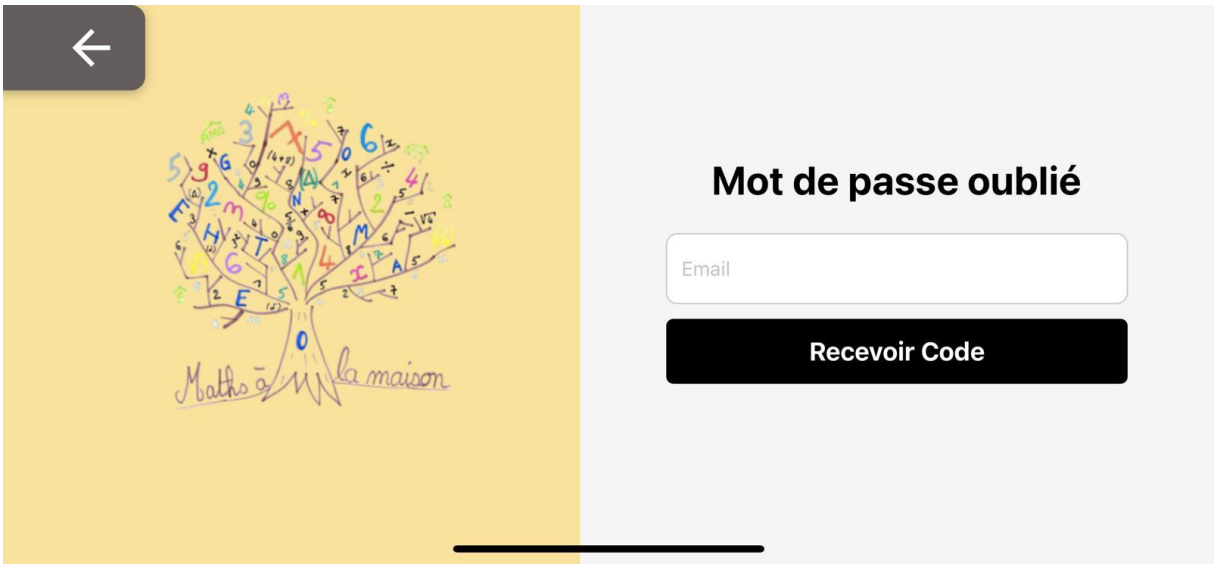


Figure 20 - Mot de passe oublié (demande de l'email)

La page de mot de passe oublié permet à l'utilisateur de réinitialiser le mot de passe associé à son compte. Pour cela il doit d'abord entrer son email.

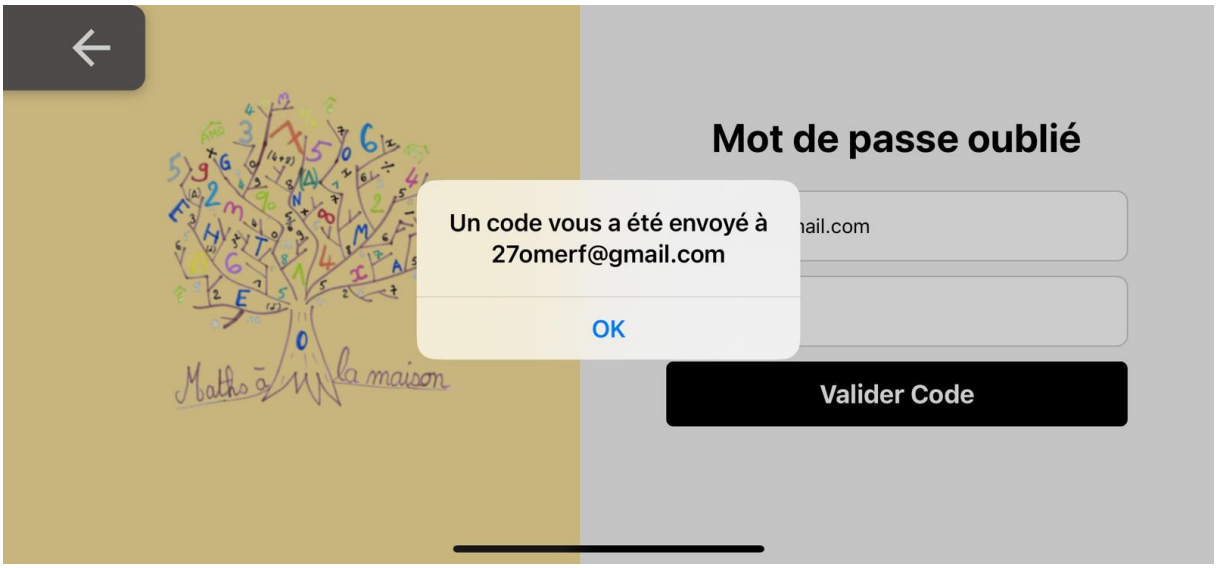


Figure 21 - Mot de passe oublié (code envoyé)

Un code va alors être envoyé à l'adresse électronique de l'utilisateur. Pour cela on va utiliser la route « **/askVerificationCode** » de l'API. On envoie le mail de l'utilisateur à notre serveur en utilisant l'API, puis le serveur va générer aléatoirement un code à 6 chiffres entre 100000 et 999999 **inclus**. On va ensuite haché le code puis on va le stocké dans un dictionnaire avec le timestamp actuelle. On va ensuite envoyé le code par mail à l'utilisateur via **mailService**. On

va ensuite retourner « **success : true** » à l'application si tous c'est passé correctement. Si un problème est survenu on va envoyé « **success : false** », un message d'erreur va alors être envoyé à l'utilisateur.

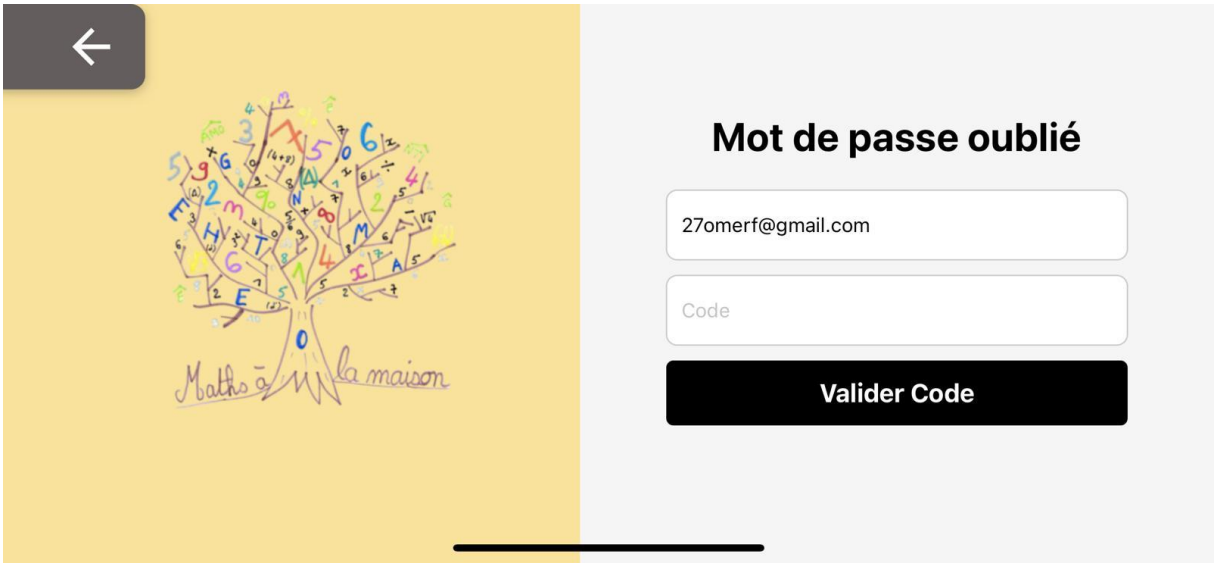


Figure 22 - Mot de passe oublié (validation du code)

L'utilisateur se retrouve alors avec cette page, il doit entrer le code qu'il a reçu par mail dans le champ de texte « **Code** ».

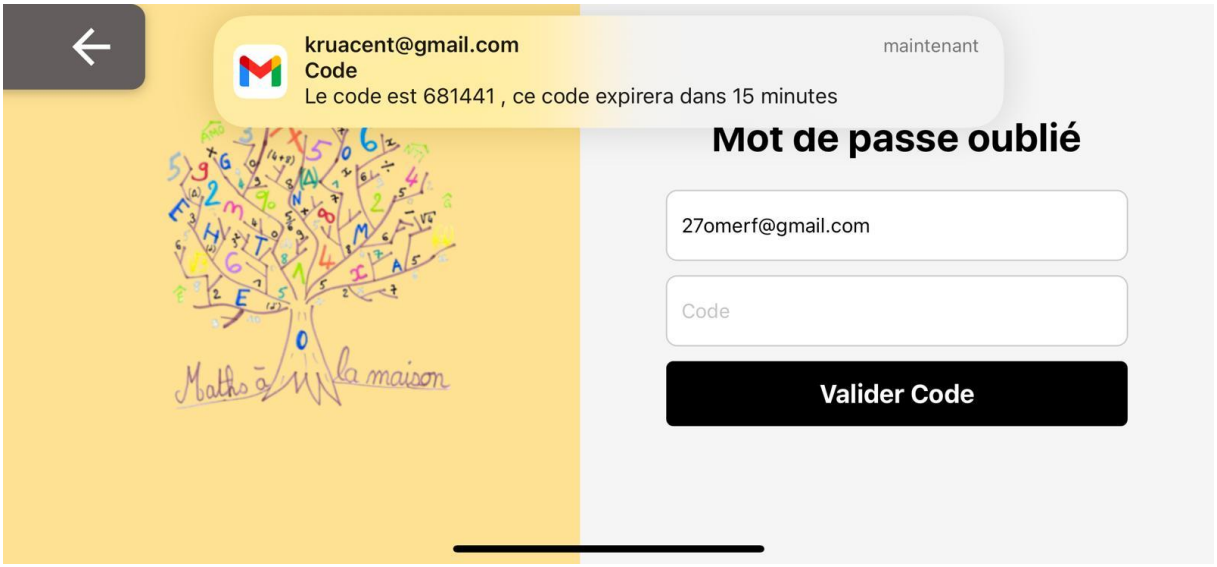


Figure 23 - Mot de passe oublié (réception du code)

On peut voir dans la **Figure 23** que l'utilisateur bien reçu le code par mail. Il doit alors entrer le code « 611441 » dans le champ Code et appuyer sur Valider. Lorsque l'utilisateur appuie sur le bouton « **Valider Code** », l'application va alors envoyé au serveur le mail et le code présent dans les champs « Email » et

« Code » via la route « **/checkVerificationCode** ». Le serveur va alors vérifier pour voir si le mail et le code est bien présent dans la requête sinon il va renvoyé « **{ error: "Email ou code manquant" }**.

Une fois cet étape passé on va cherché dans le dictionnaire des codes de validations appartenant à cet utilisateur. Si on trouve ne trouve pas de code on renvoie **{ error: "Code de validation non trouvé" }**. Si on trouve un code correspondant le serveur va vérifier si le code a expirer ou non. Pour cela il va comparer la différence de temps entre le moment où le code a été sauvegarder dans le dictionnaire et maintenant.

Si il c'est écoulé moins de temps que ce qui est défini dans la configuration du serveur on valide le code en renvoyant **{ success: true, message: "Code valide" })**, sinon on renvoie **{ error: "Code expiré" }**.

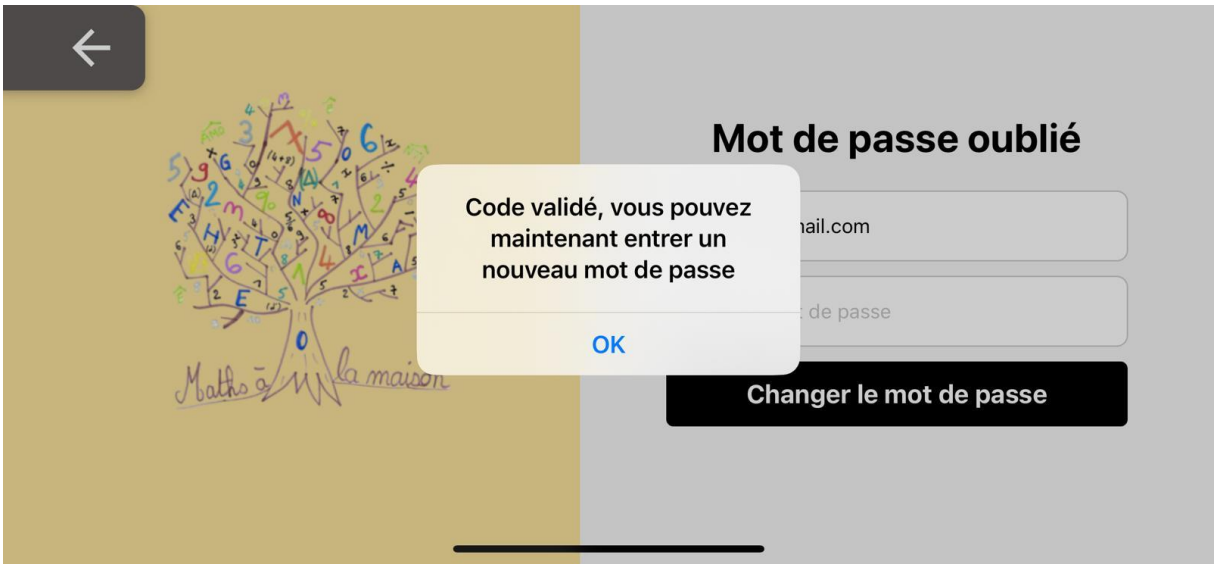


Figure 24 - Mot de passe oublié (code validé)

Si le serveur valide le code, l'utilisateur reçoit une alerte comme dans la **Figure 24**.

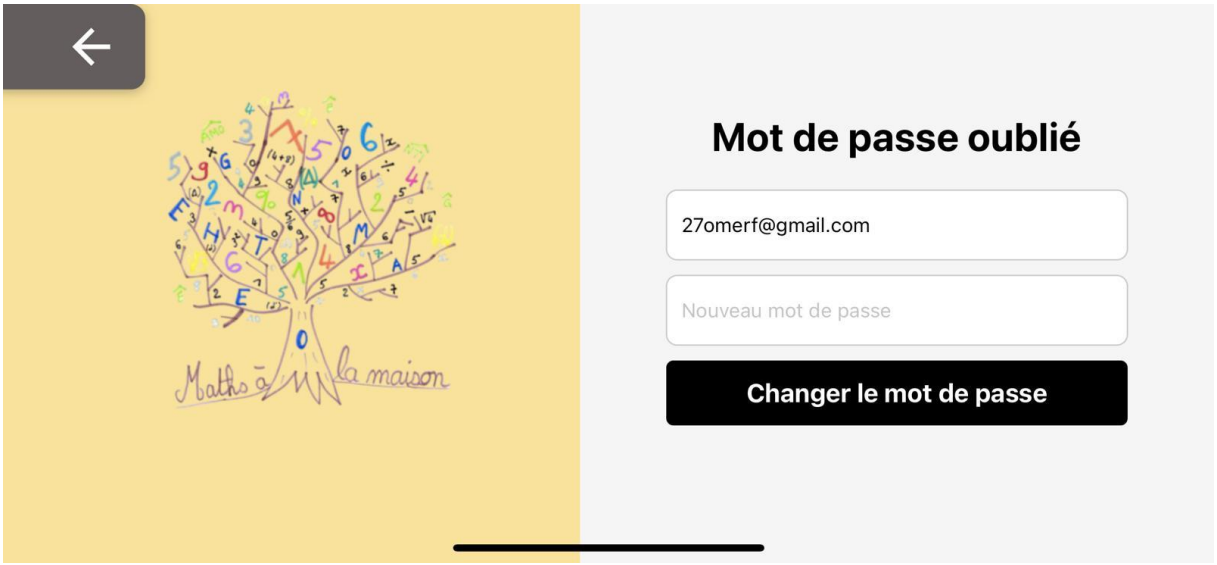


Figure 25 - Mot de passe oublié (création du nouveau mot de passe)

Une fois que le code est validé l'utilisateur peut alors entrer son nouveau mot de passe. Le champ « Email » est alors bloqué pour évité de changé le mot de passe d'un autre compte.

Lorsque l'utilisateur clique sur « Changer le mot de passe », l'application va alors vérifier si le mot de passe est correct. Si le mot de passe est considéré comme valide, on va ensuite hacher le mot de passe avec un sel générer aléatoirement par **password-util.ts**. On va ensuite envoyer le mail, le mot de passe hacher et le sel au serveur via la route « **/updatePassword** ». Le serveur va alors vérifier la présence de tous les champs, si il en manque il va envoyé « **{ error: 'Email, password, and salt are required' }** ». Si tout est correct il va alors enregistrer toutes ces informations dans la base de données, et il va alors renvoyer « **{ success: true }** ».

Si toutes les étapes se sont déroulé sans soucis, l'utilisateur reçoit l'alerte « **"Succès", "Le mot de passe à été changé avec succès"** », puis il est redirigé vers la page principale (pour les utilisateurs non-connecté).

LA PAGE PRINCIPAL :

UTILISATEUR NON CONNECTE

Figure 26 - Page principale (non connecté)

Lorsque l'utilisateur lance l'application, on vérifie si il y a une présence de variable de session. Si il y a des variables de session on affiche la page d'accueil version connecté sinon on affiche la page d'accueil version non-connecté. A chaque fois que l'utilisateur se retrouve sur la page d'accueil, on va récupérer les dernières informations concernant les joueurs depuis le serveur, pour avoir toujours les points à jour.

La page d'accueil pour les utilisateurs non connecté est comme indiqué sur la **Figure 26**. On a donc quelques boutons. L'utilisateur peut s'inscrire ou se connecter, peut accéder aux paramètres, il peut également essayer le mode entraînement.

UTILISATEUR CONNECTE

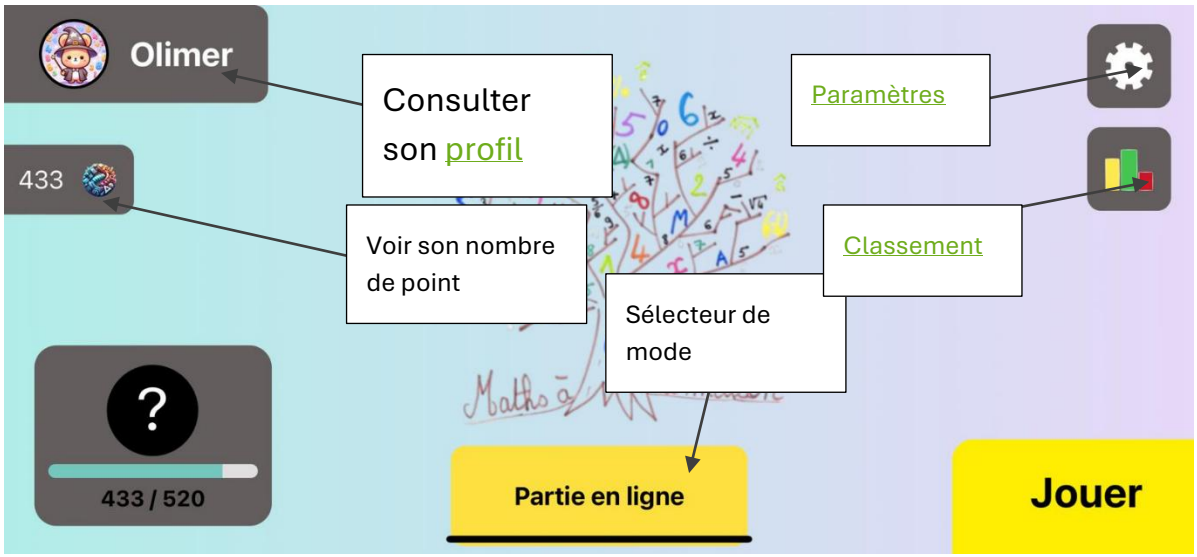


Figure 27 - Page principale (connecté)

Si l'utilisateur est connecté alors la page d'accueil va être comme indiqué sur la **Figure 27**. L'utilisateur peut effectuer des actions basiques comme voir son profil, le classement, accéder aux paramètres et sélectionner différents modes de jeux.

AVANCEMENT DES AVATARS

Figure 28 - Avancement des avatars (déjà débloqués)

Figure 29 - Avancement des avatars (non-débloqués)

La page avancements des avatars contient tous les avatars déblocables par l'utilisateur. Chaque avatar est débloqué à un nombre de point défini, une fois que l'utilisateur atteint un palier il débloque automatique l'avatar du palier, et commence le déblocage du prochain avatar. A la fin de la route des avancements d'avatars on peut gagner une photo de profil spéciales. Cette photo de profil est la photo de profil qui se situe dans le répertoire /assets/images/profile-picture/exlusive

PAGE DE PROFIL

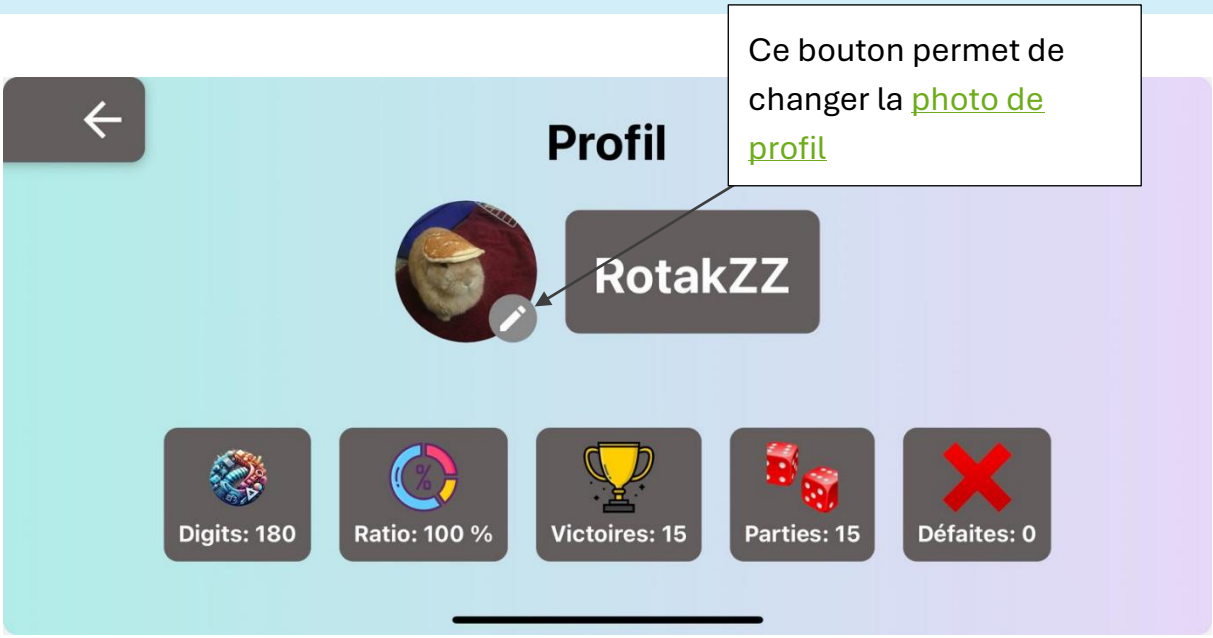


Figure 30 - Page de profil

Les utilisateurs peuvent accéder à leurs statistiques via la page de profil. Dans celle-ci on va y retrouver le nombre de points totaux qu'ils auront gagné depuis la création de leur compte. Leurs ratio parties gagnés/parties totaux. Le nombre de victoire, le nombre de parties totaux, et leurs nombres de défaites. Cette page affiche uniquement les valeurs de la variable de session du joueur, donc on n'appelle pas le serveur.

Dans cette page on peut également changer d'avatar en cliquant sur le bouton sous forme de crayon, on va alors se retrouver dans la page de « changement d'avatar ».

LA PAGE DE CHANGEMENT D'AVATAR (PHOTO DE PROFIL)

Figure 31 - Page de sélection d'avatar (avatars débloqués)

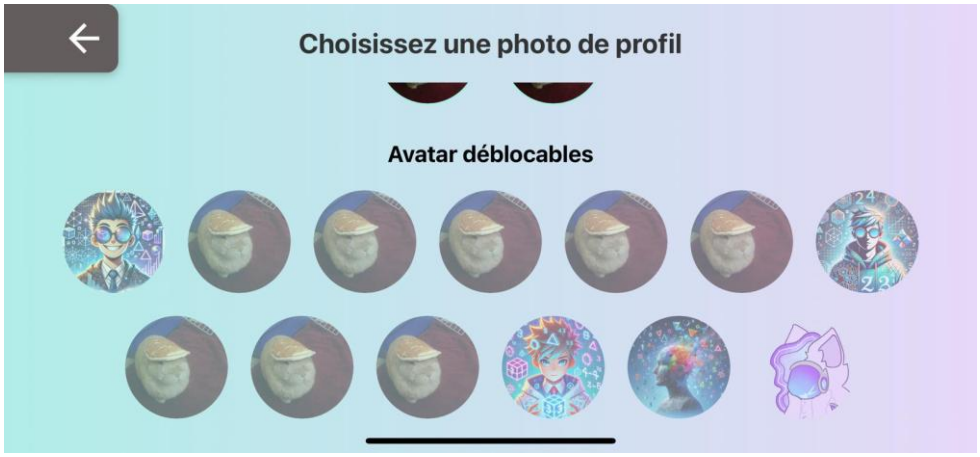


Figure 32 - Page de sélection d'avatar (avatars non-débloqués)

L'utilisateur peut ici choisir un avatar parmi les avatars proposés. Dès qu'il en choisi un, on récupère l'identifiant de l'avatar et on change la valeur de la variable de session « photoDeProfil » avec l'identifiant de l'avatar qu'il a choisi. Après cela nous envoyons une requête contenant l'identifiant de la nouvelle photo de profil et l'email de l'utilisateur au serveur via la route **'/updatePicture'**. Le serveur se charge de mettre à jour la base de données et nous renvoie une réponse 500 si un problème est survenu ou une réponse 200 qui contient vrai, le nombre de ligne affecté et l'email de l'utilisateur concerné.

L'utilisateur reçoit alors l'alerte suivante (**"Succès", "Photo de profil mise à jour !"**). Puis l'utilisateur est redirigé vers la page de profil utilisateur.

CLASSEMENTS :

CLASSEMENT : POINTS

Figure 33 - Classement (points)

La page de classement est divisée en deux parties, la partie de droite représente les statistiques de l'utilisateur (le classement du joueur, le nombre de point, le nombre de victoire et le nombre de partie), les valeurs ici sont récupérer via les variables de session de l'utilisateur.

La partie de gauche représente le classement des meilleurs joueurs en fonction de trois critères, le premier en fonction du nombre de point, le second en fonction du nombre de partie, le troisième en fonction du nombre de victoire. Les valeurs sont récupérées du serveur via la route « **/users** » qui permet de récupérer tous les utilisateurs de la base de données.

CLASSEMENT : PARTIES

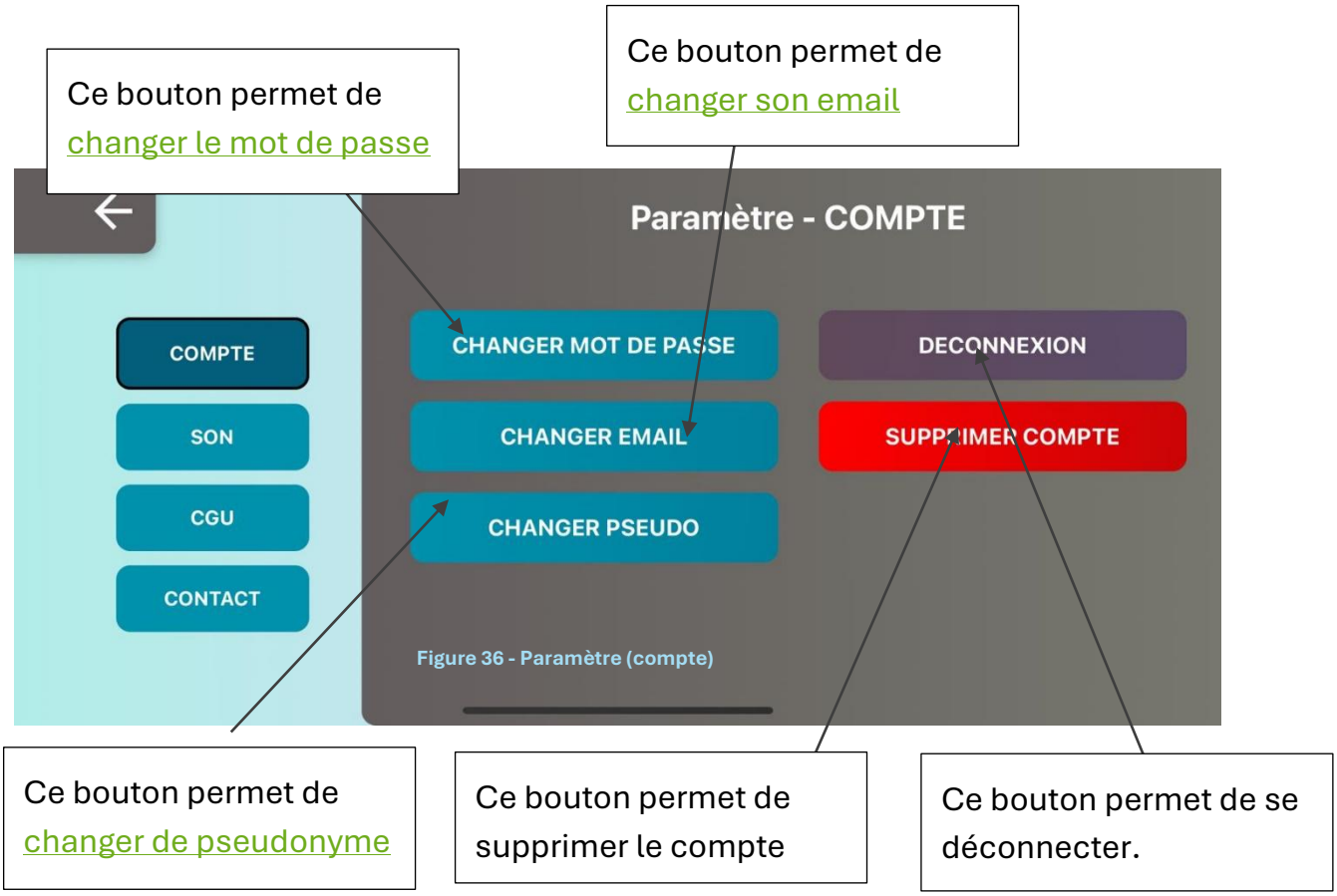
Figure 34 - Classement (parties)

CLASSEMENT : VICTOIRES

Figure 35 - Classement (victoire)

PARAMETRES :

PARAMETRES : COMPTE



Cette page permet de gérer tous les paramètres liés au compte de l'utilisateur, changer le mot de passe, l'email, le pseudonyme, se déconnecter ou supprimer son compte.

PARAMETRE : COMPTE : CHANGER PSEUDONYME

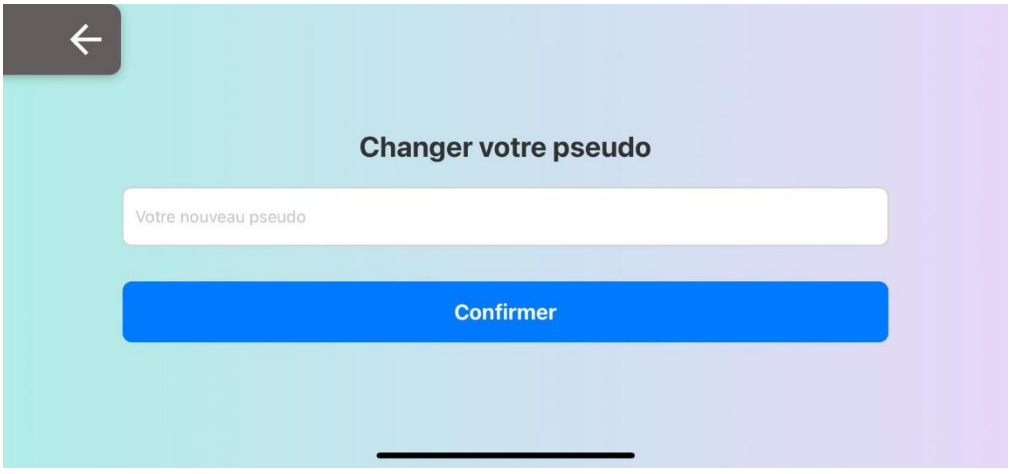


Figure 37 - Paramètre (changement pseudonyme)

Cette page permet de choisir un nouveau pseudonyme, le pseudonyme doit appartenir à aucun autre compte. Une fois que l'utilisateur appuit sur « **Confirmer** », on va tout d'abord vérifier que le pseudonyme est disponible. On va ensuite récupérer l'instance de l'utilisateur via son email, une fois récupérées on va changer le pseudonyme de l'utilisateur puis on va notifier le serveur de ce changement via la route « **/updatePseudo** » en envoyant l'email et le pseudonyme. Le serveur va ensuite mettre à jour le pseudonyme du jour dans la base de données.

PARAMETRE : COMPTE : CHANGER MOT DE PASSE

Figure 38 - Paramètre (changement de mot de passe)

Pour changer de mot de passe, il faut mettre son mot de passe actuel pour être sûr que quelqu'un d'autre ne change pas votre mot de passe par la suite, il faut crée le nouveau mot de passe, le réécrire pour le confirmer et finir par confirmer l'action. Dès que l'utilisateur clique sur « **Confirmer** », on va vérifier que le nouveau mot de passe est sécurisé, puis on va envoyer une requête au serveur avec le mot de passe et le mail de l'utilisateur. Le serveur va se charger de vérifier que mot de passe est correct, il va alors renvoyer « **{ success: true }** » si le mot de passe correspond au compte de l'utilisateur sinon il va renvoyer « **{ error: 'Mot de passe incorrect.' }** ». Si le mot de passe est correct on va alors hashé le nouveau mot de passe et l'envoyer au serveur via la route « **/updatePassword** ». Le fonctionnement de update password est expliqué ici.

PARAMETRE : COMPTE : CHANGER EMAIL

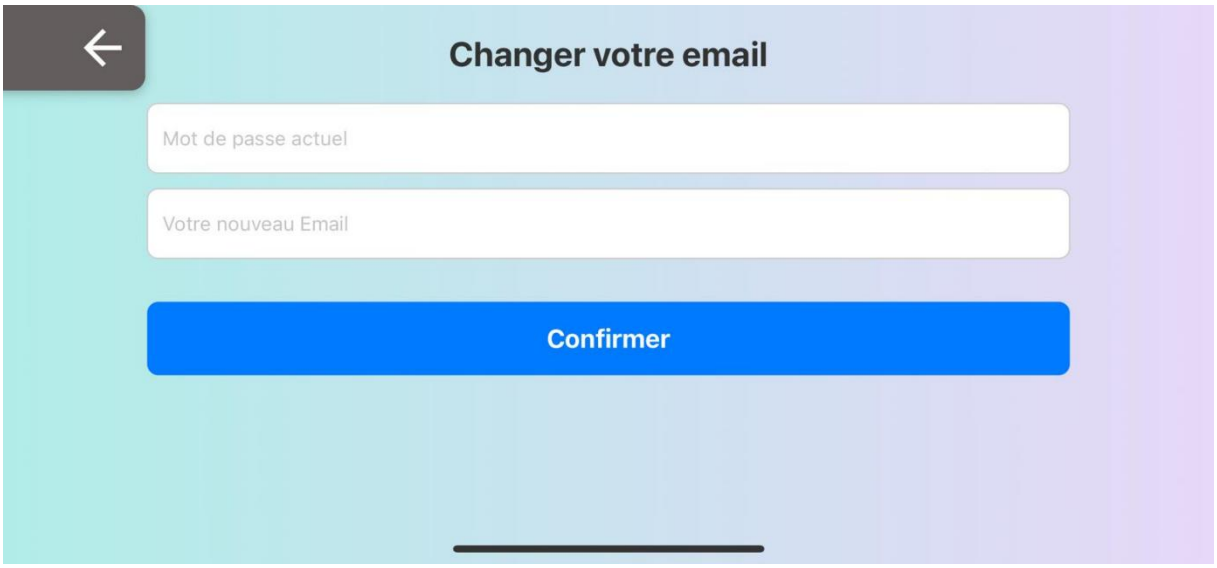


Figure 39 - Paramètre (changement d'email)

Pour changer d'email il faut mettre son mot de passe actuel pour que l'utilisateur prouve qu'il est bien le propriétaire du compte pour changer l'email. Lorsque l'utilisateur va appuyer sur « **Confirmer** », l'application va tout d'abord vérifier que le mot de passe fourni par l'utilisateur est correct. Pour cela il va appeler le serveur via la route « **/login** » en lui envoyant l'email et le mot de passe, dans le cas où l'utilisateur est authentifié on va pouvoir passer à la suite, dans le cas contraire on va afficher une alerte « **Erreur, Le mot de passe est incorrect** ».

Ensuite nous allons vérifier que l'email fourni par l'utilisateur est disponible sinon on va afficher l'alerte : « **Erreur, ce mail est déjà utilisé** », l'email doit également être valide. Pour qu'un email soit considéré comme valide il doit être au format « **test@example.fr** », si l'email n'est pas valide on va alors afficher l'alerte « **Erreur, Ce mail n'est pas valide.** ».

Une fois que les vérifications sont passées, l'application va informer au serveur du changement d'email via la route « **/updateEmail** », en envoyant le pseudonyme et le nouvel email comme paramètre.

Une fois que l'email est mis à jour dans la base de données on affiche l'alerte « **Succès, Votre adresse email a été mis à jour avec succès !** ». On enregistre le nouvel email dans la variable de session puis on redirige l'utilisateur vers la page principale.

PARAMETRE : SON

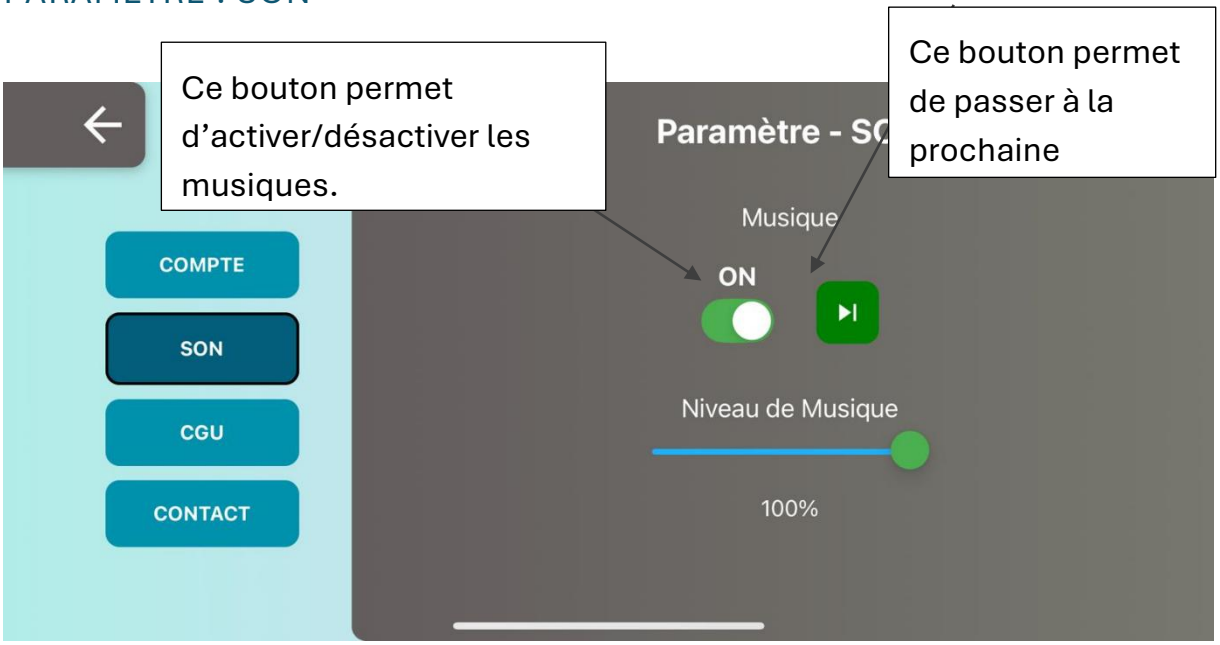


Figure 40 - Paramètre (son)

Cette page permet de gérer tous les paramètres liés à la musique. L'utilisateur peut activer/désactiver la musique. Dans le cas où l'utilisateur est connecté on va stocker l'état de la musique. Si il désactive la musique, la musique ne jouera pas tant que l'utilisateur ne l'aura pas réactivée L'utilisateur peut également changer de piste audio, si l'utilisateur tente de changer de musique alors que celle-ci est désactivé l'application affichera l'alerte « **La musique n'est pas activée.** ». Cette page présente également un curseur pour gérer le volume de la musique, sur **iOS** le niveau de la musique se met à jour instantanément pendant le mouvement de curseur de l'utilisateur, sur **Android** le niveau de musique est pris en compte uniquement quand l'utilisateur arrête d'interagir avec le curseur. Le niveau de la musique n'est pas stocké dans une variable de session, il sera à 100% à chaque lancement de l'application.

PARAMETRES : CONDITION GENERAL D'UTILISATION

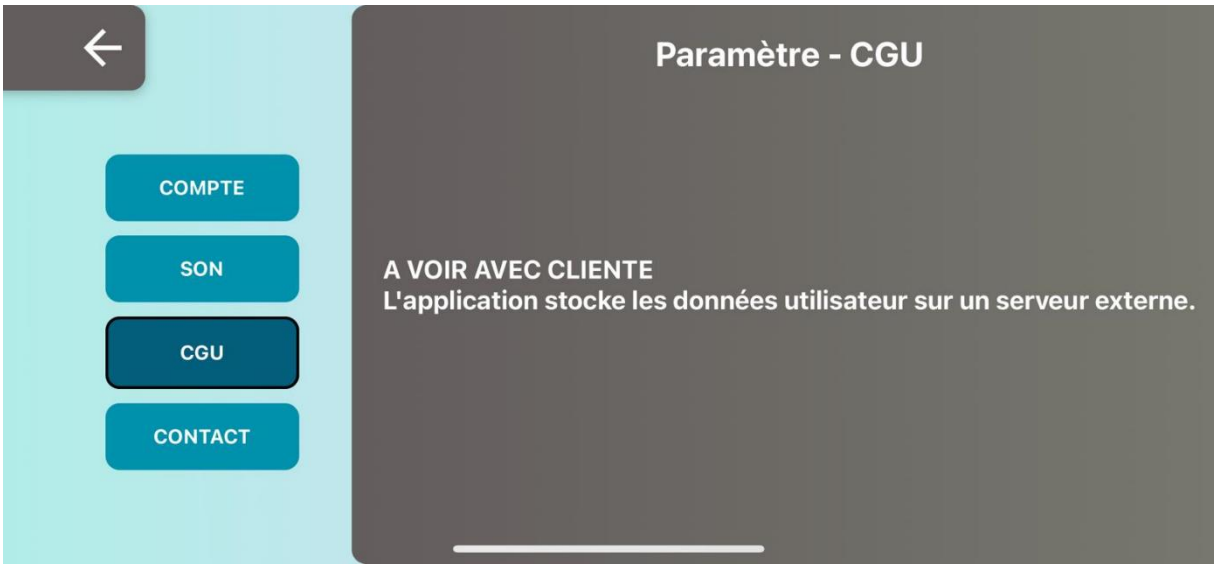


Figure 41 - Paramètre (CGU)

Cette page est une page statique permettant l'affichage des différentes CGU et mentions légales. Il ne comporte aucun appel vers un serveur ou du backend.

PARAMETRE : CONTACT

Figure 42 - Paramètre (contact)

Ceci est une page de contact, elle permet l'affichage de l'email de la cliente avec le site internet « mathsalamaison.fr ».

JOUER : MATCHMAKING

Figure 43 - Jeu (matchmaking)

Cette page est la page de matchmaking, elle s'affiche lorsqu'on clique sur « **Partie en ligne** », puis on clique sur « **Jouer** ». Une fois qu'on arrive sur cette page, l'application va créer une connexion via le protocole WebSocket avec le serveur.

Voici les différents messages possibles :

- **startSearching(ws, parsedMessage.pseudo)** : Permet au joueur d'être ajouté à la liste d'attente.
- **stopSearching(ws, parsedMessage.pseudo)** : Permet au joueur d'être supprimé de la liste d'attente.
- **broadcastPlayersSearching()** : Permet d'envoyer la liste des joueurs cherchant une partie à tout les joueurs cherchant une partie.

A chaque nouveau joueur ajouté dans la liste d'attente le serveur va vérifier si il y a assez de joueur pour commencer une partie, ce nombre est changeable dans « **config > gameAttribute.js > playerPerGame** »

JOUER : LOCAL

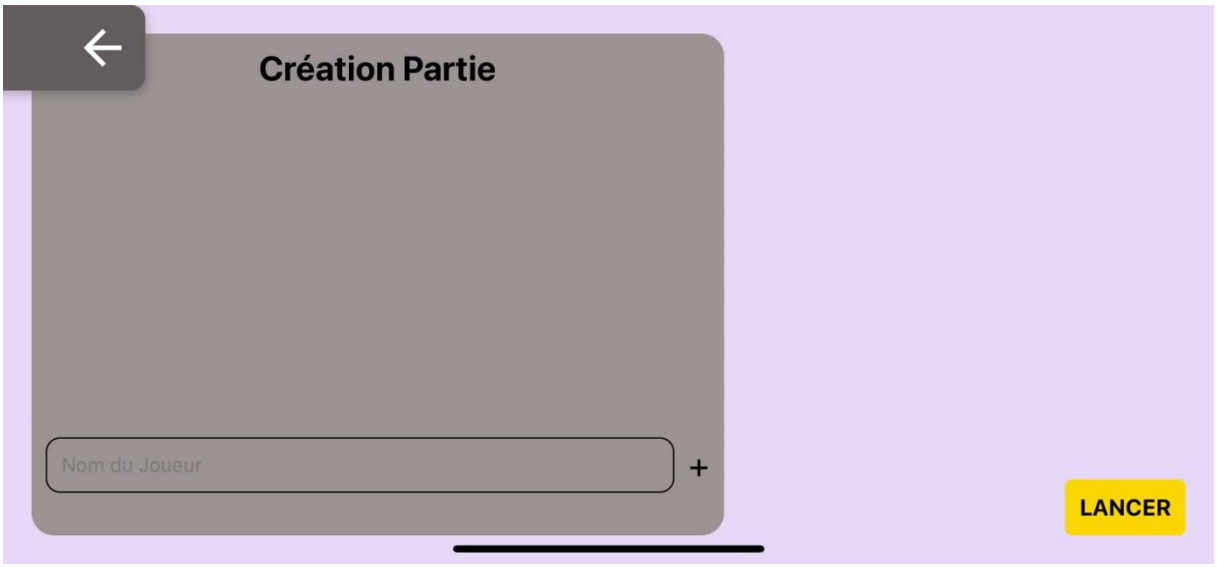


Figure 44 - Jeu (local)

Cette page est encore en développement, elle permet d'ajouter des joueurs grâce au bouton + situé à côté du champ de texte. On peut ajouter au plus 4 joueurs. Lorsqu'on appuie sur « **Lancer »** on a le message « **Fonctionnalité non implementé", "Cette fonctionnalité est en cours de développement** ».

Cette page fait pas encore d'appel vers le serveur ou vers un code backend.

JOUER : JEU DE TEST

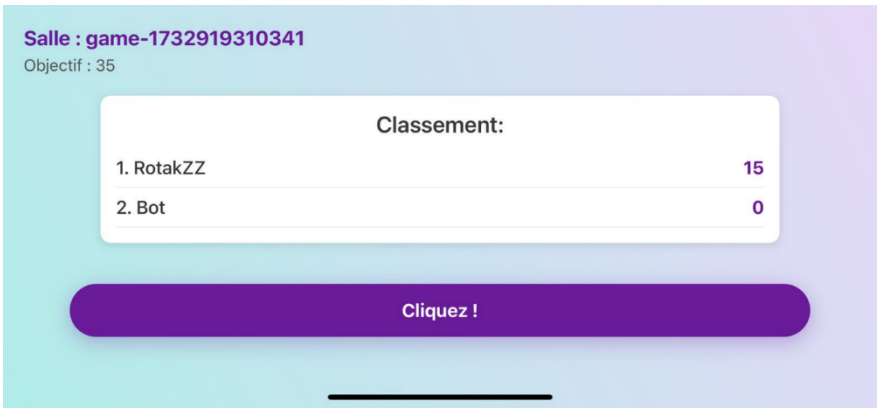


Figure 45 - Jeu (partie en ligne)

Ce jeu est une version de test pour la mise en place d'un multijoueur pour MathsALaMaison. Le but de ce mini-jeu est d'atteindre l'objectif le plus rapidement possible en cliquant sur le bouton « **Cliquez !** ». On accède à cette page lorsqu'on atteint le nombre de joueur suffisant pour créer une partie dans le matchmaking.

Lorsque le serveur détecte qu'il y a assez de joueur pour lancer une partie, il va créer une partie. Une partie est un tableau contenant les valeurs suivantes :

- **id** : L'identifiant unique de la partie (actuellement l'identifiant de la partie est le timestamp à laquelle elle a était créer).
- **players :** Liste contenant tous les joueurs de la partie.
- **scores :** Liste de tous les scores des utilisateurs.
- **clickGoal :** Objectif de clique à atteindre.

On va ensuite stocker cette partie dans une autre liste nommée « **games** » qui va contenir toutes les parties qui sont en cours. Ensuite on va informer qu'une partie est crée aux utilisateurs.

Une fois que les utilisateurs sont informés on les emmènes dans la page de la **Figure 34** pour qu'ils puissent jouer. A chaque clique d'un utilisateur l'application va envoyer au serveur un paquet contenant :

- **type** : userClick (Pour que le serveur sache où renvoyer le paquet reçue)
- **gameId :** L'identifiant de la partie
- **playerId :** L'identifiant du joueur qui envoie son clique.

Lorsqu'un joueur atteint l'objectif de score un message est alors envoyé à tout le monde contenant :

- **type :** gameOver (Pour que l'application sache quoi faire de la requête reçue)
- **gameId :** L'identifiant de la partie
- **winner :** Le vainqueur de la partie
- **message :** Message de fin de partie.

Une fois que le serveur a envoyé ce message il va mettre à jour les statistiques des joueurs dans la base de données, il va ensuite supprimé la liste qui correspondait au jeu depuis **games**.

Si l'objectif de clique n'est pas atteint, à chaque nouveau clique le serveur va envoyer au joueur qu'un joueur à cliquer, pour que l'application de chaque utilisateur incrémente son score (pour avoir une synchronisation entre les clients) .

MODE ENTRAINEMENT :

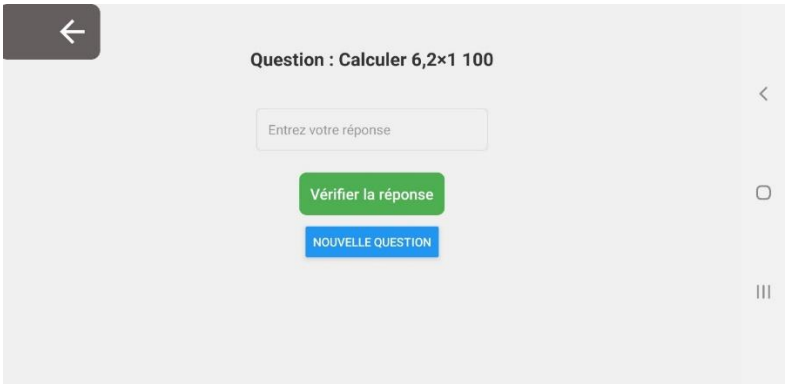


Figure 46 - Jeu (entraînement)

Le mode entraînement permet de s'entraîner aux 600 questions présent dans la base de données de MathsALaMaison. La base de données concernant les questions contient :

- **id :** Identifiant de la question (auto-incrémentée par la base de données)
- **typeQuestion :** Nombre entre 0 et 11, qui précise la catégorie de la question .
- **question :** Texte contenant la question
- **typeReponse :** Acronyme représentant le type de réponse attendu
- **reponse :** Texte contenant la ou les différentes réponse possible.
- **Correction :** Correction de la question
- **image_data :** Image concernant la question si présent.

Pour gérer les questions, nous utilisons les types de questions et les types de réponses.

Les types de questions ont été défini par la cliente, à part la catégorie « **Centre** » qui est une catégorie à part permettant de choisir une question aléatoire parmi n'importe quelles catégories.

Les types de réponses ont été défini par nous, ils nous permettent d'afficher le bon affichage pour les questions. Si la question est de type QCM on doit afficher une interface de type QCM avec plusieurs choix possible. Chaque type de réponse a une signification et permetent de traiter l'attribut réponse de la question différemment.

Voici les différents type de questions :

```
const enum Category{
    Centre = -1,
    Espace ,
    Proportionnalite ,
    Communiquer ,
    Transformation ,
    Geometrie ,
    CalculLitteral ,
    Informatique ,
    Nombres,
    Calculs,
    Logique,
    StatOuProbas,
    Fonctions
}
export default Category;
```

Figure 47 - Catégories

Voici les différents types de réponses et leurs utilisations :

QCM :

Nombre de questions : Il y a 204 questions de type QCM

Signification : Un Question à Choix Multiples avec soit 3/4/5 choix possibles avec une réponse correct.

Utilisation du champ réponse : « **BonneRep|MauvaiseRep|MauvaiseRep|...** » Les choix possibles sont séparées par le caractères « **|** ». Le premier choix est toujours la réponse correct, nous mélangeons les réponses pour l'afficher à l'utilisateur.

RCV

Nombre de questions : Il y a 5 questions de type RCV

Signification : Réponse qui doit contenir tout les INT ou FLOAT indiquer dans la réponse.

Utilisation du champ réponse :
« **Reponse1|Reponse2|Reponse3|Reponse4|Reponse5** ». Les réponses attendues sont séparées par le caractères « **|** ». Pour que l'utilisateur répond correctement sa réponse doit contenir toutes les réponses attendues.

RDS

Nombre de questions : Il y a 223 questions de type RDS

Signification : Réponse standards contenant plusieurs type de réponse attendu

Utilisation du champ réponse :
« **ReponsePossible1|ReponsePossible2|ReponsePossible3** ». Les réponses attendues sont séparées par le caractères « **|** ». L'utilisateur à le point si il entre l'une des réponses possibles.

RLD2*

Nombre de questions : Il y a 12 questions de type RLD2

Signification : Réponse avec 2 choix INT ou FLOAT sous forme de liste déroulante.

Utilisation du champ réponse : « **Reponse1|texte1|Reponse2|texte2** ». Les réponses attendues sont séparées par le caractères « **|** ». Pour que l'utilisateur répond correctement sa réponse doit contenir toutes les réponses attendues au bon emplacements.

RLD3*

Nombre de questions : Il y a 26 questions de type RLD3

Signification : Réponse avec 3 choix INT ou FLOAT sous forme de liste déroulante.

Utilisation du champ réponse :
« **Reponse1|texte1|Reponse2|texte2|Reponse3|texte3** ». Les réponses attendues sont séparées par le caractères « **|** ». Pour que l'utilisateur répond correctement sa réponse doit contenir toutes les réponses attendues au bons emplacement.

RLD4*

Nombre de questions : Il y a 2 questions de type RLD4

Signification : Réponse avec 4 choix INT ou FLOAT sous forme de liste déroulante.

Utilisation du champ réponse :
« **Reponse1|texte1|Reponse2|texte2|Reponse3|texte3|Reponse4|texte4**». Les réponses attendues sont séparées par le caractères « **|** ». Pour que l'utilisateur répond correctement sa réponse doit contenir toutes les réponses attendues aux bons emplacements.

VF

Nombre de questions : Il y a 43 questions de type VF

Signification : Question de type Vrai ou Faux

Utilisation du champ réponse : « **Soit V, Soit F »**

Exemple : « V »

Pour que l'utilisateur répond correctement sa réponse doit être vrai si la question est vrai, faux si la question est fausse.

VFRDS*

Nombre de questions : Il y a 8 questions de type VFRDS

Signification : Vrai ou Faux avec réponse standard

Utilisation du champ réponse : « **Soit V|reponse ou F**».

Exemple : « V|2300 »

Les réponses attendues sont séparées par le caractères « **|** ». Pour que l'utilisateur répond correctement sa réponse doit être vrai si la question est vrai, faux si la question est fausse et il doit contenir la bonne réponse.

VFRLD1*

Nombre de questions : Il y a 13 questions de type VFRLD1

Signification : Vrai ou Faux avec RLD1

Exemple : « V|La solution est : |2,75 »

Utilisation du champ réponse : « **Soit V|texte1|reponse1 ou F**».

Pour que l'utilisateur à les points il doit répondre vrai ou faux correctement puis répondre à la question avec les réponses possibles qui sont après le « | ».

VFRLD2*

Nombre de questions : Il y a 7 questions de type VFRLD2

Signification : Vrai ou Faux avec RLD2

Utilisation du champ réponse : « **Soit V|texte1|reponse1|texte2|reponse2 ou F**».

Exemple : « V|Les 2 solutions sont : |0| et |-1/7| »

Pour que l'utilisateur ait les points il doit répondre vrai ou faux correctement puis répondre à la question avec les réponses possibles qui sont après le « | ».

Les questions marquées d'une * ne sont pas encore implémentée dans l'application

GLOSSAIRE

Expo Go : Plateforme de type bac à sable, permettant de tester et partager du code beaucoup plus simplement. (https://expo.dev/go).

Digits : C'est le nom qu'on a décidé de donner aux points gagnés.

SBMM : Signifie "Skill-Based Matchmaking". C'est un système utilisé pour mettre en relation des joueurs ayant des compétences similaires. Dans notre cas cela permettrai que les joueurs affrontent des joueurs de leurs niveau.

Scrum Master : Le Scrum Master est un facilitateur qui aide l'équipe à adopter et à appliquer la méthodologie Scrum, en supprimant les obstacles et en veillant à la bonne communication et collaboration au sein de l'équipe.

Token : Un token est une chaîne de caractères unique utilisée pour authentifier une application ou un utilisateur, garantissant l'accès sécurisé à des ressources ou services spécifiques.

API : Une API (Interface de Programmation d'Application) est un ensemble de protocoles permettant à plusieurs applications de communiquer entre eux.

TABLE DES FIGURES