

TABLE DES MATIÈRES

1. LISTE DES COMPÉTENCES DU RÉFÉRENTIEL COUVERTES PAR LE PROJET

1.1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

 1.1.1. Maquetter une application

 1.1.2. Développer une interface utilisateur desktop

 1.1.3. Développer des composants d'accès aux données

 1.1.4. Développer la partie front-end d'une interface utilisateur web

 1.1.5. Développer la partie back-end d'une interface utilisateur web

1.2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

 1.2.1. Concevoir une base de données

 1.2.2. Mettre en place une base de données

 1.2.3. Développer des composants dans le langage d'une base de données

1.3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

 1.3.1. Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement

 1.3.2. Concevoir une application

 1.3.3. Développer des composants métier

 1.3.4. Construire une application organisée en couches

 1.3.5. Développer une application mobile

 1.3.6. Préparer et exécuter les plans de tests d'une application

 1.3.7. Préparer et exécuter le déploiement d'une application

2. RÉSUMÉ DU PROJET

3. CAHIER DES CHARGES

3.1 Présentation d'ensemble du projet

 3.1.1 Présentation de l'entreprise / du client

 3.1.2 Description de l'existant

 3.1.3 Objectifs du projet

 3.1.4 Intervenants sur le projet

 3.1.5 Cible adressée par le projet

 3.1.6 Principes de référencement

 3.1.7 Exigences de performances et de volumétrie

- 3.1.8 Multilinguisme & adaptations pour un public spécifique
- 3.1.9 Reprise de l'existant
- 3.2 Description graphique et ergonomique
 - 3.2.1 Composants de la charte graphique
 - 3.2.2 Design, Responsive design et autres exigences liées au design
 - 3.2.3 Maquettes
- 3.3. Besoins fonctionnels « métier »
 - 3.3.1. Utilisateurs du projet
 - 3.3.2. Processus utilisateur impacté
 - 3.3.3. Informations relatives aux contenus
 - 3.3.4. Inventaire des besoins fonctionnels
- 3.4 Prestations attendues
 - 3.4.1. Livrables et prestations
 - 3.4.2. Budget

4. GESTION DE PROJET

- 4.1. Méthodologie
- 4.2 Outils, planning et suivi

5. SPÉCIFICATIONS FONCTIONNELLES

- 5.1. Description de la solution
 - 5.1.1. Caractéristiques de la solution
 - 5.1.2. Acteurs du projet
 - 5.1.3. Applications connexes
 - 5.1.4. Documentation référence
 - 5.1.5. Lexique/glossaire
 - 5.1.6. Aspect métier
- 5.2. Fonctionnalités
 - 5.2.1. Matrice Profil/Droits
 - 5.2.2. Fonctionnalités Générales
 - 5.2.3. Fonctionnalités détaillées
 - 5.2.4. Aide en ligne
 - 5.2.5. Gestion des langues

5.2.6. Accessibilité

6. SPECIFICATIONS TECHNIQUES

- 6.1. Référencement
- 6.2. Environnement technique
- 6.3. Navigation et accessibilité
- 6.4. Services tiers
- 6.5. Sécurité
- 6.6. Gestion du contenu, des données
 - 6.6.1. Modèle Conceptuel de Données (MERISE)
 - 6.6.2. Modèle Logique de Données (MERISE)
 - 6.6.3. Modèle Physique de Données (MERISE)
 - 6.6.4. Diagramme de classes (UML)
 - 6.6.5. Structure JSON de notre base de données MongoDB

7. RÉALISATIONS

- 7.1. Exemple 1 : Validation des entrées du formulaire d'inscription côté front avec affichage des erreurs à l'utilisateur selon les cas échéants.
- 7.2. Exemple 2 : la validation des champs côté back et la sécurisation des données.
- 7.3 Exemple 3 : Récupération, traitement et affichage des scores finaux pour le Podium de fin de partie.

8. PRÉSENTATION DU JEU D'ESSAI DE LA FONCTIONNALITÉ LA PLUS REPRÉSENTATIVE

- 8.1. Fonctionnalité testée
- 8.2. Description des scénarios
- 8.3. Résultats des tests
- 8.4. Conclusion

9. DESCRIPTION DE LA VEILLE SUR LES VULNÉRABILITÉS DE SÉCURITÉ

- 9.1 Comment je me renseigne ?
- 9.2 À quelle fréquence je me renseigne ?
- 9.3 Ce que je mets en œuvre dans ce domaine

10. DESCRIPTION D'UNE SITUATION DE TRAVAIL AYANT NÉCESSITÉ UNE RECHERCHE SUR SITE ANGLOPHONE ou FRANCOPHONE

- 10.1. Description du besoin

10.2. Description de la recherche

10.3. Exploitation du résultat

11. ANNEXES

1. LISTE DES COMPÉTENCES DU RÉFÉRENTIEL COUVERTES PAR LE PROJET

1.1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

1.1.1. Maquetter une application

Les maquettes de notre projet d'application mobile ont été dessinées sur Canva d'après le cahier des charges élaboré en amont. Les maîtres mots qui ont guidé notre maquettage sont ergonomie et sobriété. Nous faisons un jeu. Il se doit d'être intuitif et accessible à tous. Nous avons fait valider l'UI par un web designer.

1.1.2. Développer une interface utilisateur desktop

Notre projet est une application mobile, il n'y a donc pas de version desktop.

1.1.3. Développer des composants d'accès aux données

Notre API a en charge trois endpoints d'importance égale pour la gestion du jeu et la sécurisation des données :

- User
- Themes
- Games

Nous avons mis en place des composants d'accès aux données qui permettent de gérer les interactions entre les utilisateurs, les thèmes et les parties tout en assurant la sécurité et la performance de l'application.

1.1.4. Développer la partie front-end d'une interface utilisateur web

Nous avons utilisé le langage React Native et son framework Expo Go pour développer l'interface front-end de notre application.

1.1.5. Développer la partie back-end d'une interface utilisateur web

Pour le back-end, nous avons utilisé Node.js et MongoDB (base de données NoSQL) qui disposent ensemble d'une très bonne synergie.

1.2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

1.2.1. Concevoir une base de données

Conception de la base de données sur draw.io.

Trois tables : Games, Themes et Users. Qui deviendront trois collections dans notre base de données MongoDB.

1.2.2. Mettre en place une base de données

Base de données non relationnelle, MongoDB, utilisée avec l'ORM Mongoose.

1.2.3. Développer des composants dans le langage d'une base de données

Conception des models des objets de notre base de données.

1.3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

1.3.1. Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement

Utilisation de Trello, Discord et Notion pour la gestion du travail collaboratif.

1.3.2. Concevoir une application

L'application Dash a été conçue en suivant une méthodologie centrée sur l'utilisateur, en partant du cahier des charges jusqu'à l'élaboration des maquettes fonctionnelles. Nous avons mis l'accent sur l'expérience utilisateur (UX) pour garantir une prise en main simple et rapide, adaptée à un jeu casual. L'architecture de l'application a été définie pour séparer les responsabilités entre le front-end, le back-end et la base de données, en respectant le modèle MVC (Model-View-Controller). Chaque décision de conception a été validée au regard des objectifs de performance, de sécurité et de maintenabilité.

1.3.3. Développer des composants métier

Les composants métier de l'application Dash gèrent les règles spécifiques du jeu, telles que le calcul des points par équipe, la gestion des sessions de jeu, et l'attribution aléatoire des thèmes. Ces composants sont encapsulés dans le back-end et respectent une architecture modulaire, facilitant l'évolution des règles ou l'ajout de nouvelles fonctionnalités.

1.3.4. Construire une application organisée en couches

Notre serveur est organisé en MVC.

1.3.5. Développer une application mobile

Notre front est une application mobile développée en React Native

1.3.6. Préparer et exécuter les plans de tests d'une application

Nous avons réalisé des tests fonctionnels et des tests utilisateurs pour s'assurer du bon fonctionnement de notre application.

1.3.7. Préparer et exécuter le déploiement d'une application

Le déploiement de l'application Dash sera réalisé en utilisant Google Play Console pour distribuer l'application aux utilisateurs. Avant cela, nous aurons configuré et signé l'APK, suivi par des phases de tests internes et de tests en version bêta pour assurer la stabilité de l'application avant la mise en production.

Pour le back-end, nous avons opté pour un serveur AWS, offrant une grande flexibilité et scalabilité. La base de données MongoDB est hébergée sur MongoDB Atlas, permettant une gestion simplifiée et sécurisée des données avec des sauvegardes automatiques et des outils de monitoring. Le déploiement

du back-end a été automatisé avec des scripts qui assurent la mise à jour du serveur sans temps d'arrêt, en respectant les meilleures pratiques de sécurité et de gestion des clés API.

2. RESUMES DU PROJET

DASH est une application mobile de jeu d'ambiance où les joueurs devinent des cartes en équipe. Le but de cette application, en plus d'être divertissante, est également d'avoir une excellente rejouabilité et de remplacer le jeu de cartes papier que l'on a oublié d'amener lors d'une soirée entre amis ou en famille. Dans un premier temps, les joueurs choisissent leurs thèmes, le nombre d'équipes (de 2 à 6) et le nombre de joueurs par équipe (de 2 à 6). Ils choisissent également la durée d'un tour d'équipe, 30 - 45 - 60 secondes.

Le jeu se déroule en 3 manches :

Manche 1 : il y a 42 cartes à deviner, sur des thèmes divers (géographie, célébrités, histoire, anime...). Au total, le jeu comprend 20 thèmes plus un thème personnalisable). Les équipes peuvent durant cette première manche s'exprimer librement pour faire deviner leurs cartes. *Exemple : C'est une célèbre souris de Disney.*

Manche 2 : La manche 2 commence lorsque les 42 cartes ont été devinées. Les équipes essaient à nouveau de faire deviner les mêmes cartes, mais cette fois-ci en utilisant un seul mot. *Exemple : souris.*

Manche 3 : La manche 3 débute lorsque les 42 cartes ont été devinées. Dorénavant, il est interdit de parler pour faire deviner les cartes. Les joueurs doivent mimer. *Exemple : mime très réussi de la souris, "squick, squik"(bruitages autorisés).*

DASH is a mobile party game where players guess cards in teams. The goal of this app, in addition to being entertaining, is to offer excellent replayability and to replace the physical card game that one might forget to bring to a gathering with friends or family. To start, players choose their themes, the number of teams (from 2 to 6), and the number of players per team (from 2 to 6). They also select the duration of each team's turn: 30, 45, or 60 seconds.

The game is played over 3 rounds:

Round 1: There are 42 cards to guess, with various themes (geography, celebrities, history, anime...). In total, the game includes 20 themes plus one customizable theme). In this first round, teams can freely describe their cards to make their teammates guess. *Example: "It's a famous Disney mouse."*

Round 2: Round 2 begins when all 42 cards have been guessed. Teams try to guess the same cards again, but this time using only one word. *Example: "mouse."*

Round 3: Round 3 starts when all 42 cards have been guessed. From now on, talking is not allowed to make the cards guessed. Players must mime. *Example: a well-executed mime of the mouse, "squeak, squeak" (sound effects are allowed).*

3. CAHIER DES CHARGES

3.1 Présentation d'ensemble du projet

3.1.1 Présentation de l'entreprise / du client

Il n'y a pas d'entreprise à proprement parler. L'équipe de DASH se compose de trois développeurs (J s) qui conçoivent cette application sur leur temps libre. Ils ont au préalable fait une version 1 au format desktop en tant que Proof of Concept.

L'équipe croit fortement au potentiel de ce jeu, c'est pourquoi ils ont décidé d'en faire une seconde version, mobile cette fois, qui sera publiée sur le Play Store et de la présenter au titre de Concepteur Développeur d'Applications. Si succès il y a, un partenariat commercial sera envisagé entre les concepteurs.

3.1.2 Description de l'existant

Pourquoi un jeu de société ?

Les jeux de société ont connu une croissance continue sur les 30 dernières années, avec un taux de croissance à deux chiffres. Cette croissance a touché un large public, allant bien au-delà d'une activité réservée aux enfants.

En 2022, 87% des Français ont déclaré jouer régulièrement à des jeux de société, positionnant la France en tête du marché européen des jeux de société.

Plus de 33 millions de boîtes de jeux de société ont été vendues en France en 2022.

Parmi les différentes catégories de jeux de société, les petits jeux ou jeux "apéritifs" ont connu un succès considérable ces dernières années. Ces jeux sont sociaux, abordables et basés sur des concepts simples, ce qui les rend accessibles à tous, y compris aux joueurs occasionnels. Ils se jouent en quelques minutes et sont parfaits pour animer une soirée entre amis ou en famille.

Time's up, le jeu de cartes sur lequel est basé notre application, a été vendu à plus de 500 000 copies à travers le monde. Ce n'est pas une création originale, en effet, ce jeu est inspiré d'un jeu plus ancien appelé Celebrities. Le jeu est du domaine public, nous pouvons donc le remodeler en application sans risque d'atteinte au droit d'auteur.

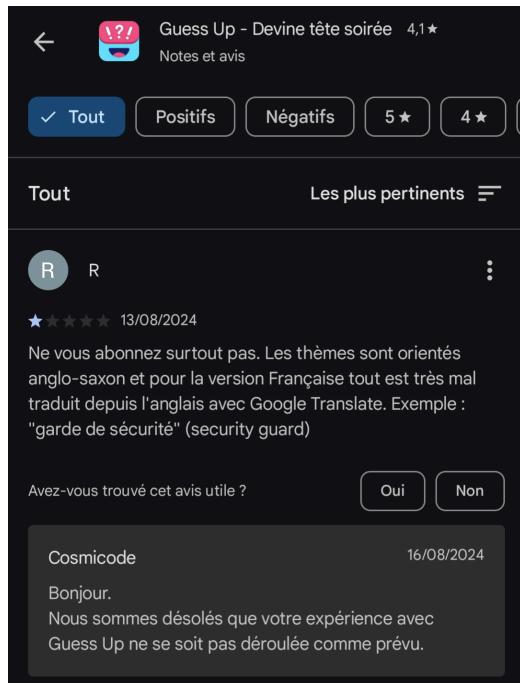
Qu'en est-il de la concurrence ?

Malgré le succès immense de leur jeu de cartes, Asmodée, la boîte-mère n'a pas pris le parti du numérique et n'a pas développé d'application pérenne et qualitative du Time's Up.

Il existe bien une version pour les enfants, mais sa base de données est pauvre et la rejouabilité est donc inexistante. De plus, elle ne semble plus maintenue, comme l'indiquent les commentaires dans le Play Store.



Depuis peu, une application nommée **Guess Up** est arrivée sur le Play Store. Développée par un studio de développement portugais, elle a de nombreux points forts : interface agréable et ergonomique, diversité de thèmes, mises à jour régulières. Cependant, elle est, comme beaucoup d'applications, polluée par les publicités intempestives et une fois le contenu gratuit exploité, les utilisateurs sont frustrés par le fait que trop d'éléments soient payants. De plus, bien que le jeu soit traduit en français, il est mal traduit, ou ne tient pas compte de la localisation (processus d'adapter un produit, un texte ou un contenu pour qu'il soit parfaitement adapté à une région spécifique, en tenant compte des différences culturelles, linguistiques).



Cette application, lancée en 2023, a été téléchargée plus de 5 millions de fois, malgré ses défauts.

C'est très encourageant pour Dash, où l'accent a été mis sur la qualité de la base de données, et son adaptation au marché français.

3.1.3 Objectifs du projet

L'objectif de Dash est d'être ce petit jeu que tout le monde a sur son téléphone et que l'on va sortir en soirée, pour une heure ou deux. Un divertissement renouvelé par des mises à jour régulières, du contenu additionnel et une ergonomie toujours plus optimisée.

D'un point de vue commercial, il y aura une partie payante mais le but premier est d'éviter les publicités intempestives qui gâcheraient l'expérience de jeu. Nous explorons l'idée des packs de thèmes payants.

3.1.4 Intervenants sur le projet

Il y a trois personnes impliquées dans ce projet, qui sont de base, développeurs.

Mais il nous faut également endosser d'autres responsabilités, puisque le code n'est qu'un élément parmi d'autres, dans la création d'une application.

Nous sommes donc webdesigners, comptables, juristes, devOps, commerciaux, dans la mesure de nos moyens et de nos préférences.

3.1.5 Cible adressée par le projet

DASH vise un public très divers de particuliers, c'est un jeu familial.

Néanmoins, le cœur de cible de notre application est la génération des "Millenials". Ce groupe est généralement familier avec les applications mobiles et les jeux de société. Les segments de ce public peuvent être analysés comme suit :

- **Hommes et femmes** : Le jeu est conçu pour être accessible et attrayant pour tous les genres, avec un design cartoon et une interface intuitive.
- **CSP moyennes à supérieures** : Ce segment comprend des individus ayant un intérêt pour les loisirs numériques et les jeux de société, souvent avec un pouvoir d'achat suffisant pour acheter des contenus premium.
- **Joueurs occasionnels et réguliers** : DASH attire à la fois ceux qui jouent occasionnellement lors de soirées entre amis ou en famille, ainsi que les joueurs réguliers qui recherchent une expérience de jeu renouvelée.

En résumé, le public cible de DASH est varié, mais se concentre sur les jeunes adultes technologiquement avertis qui apprécient les jeux de société et les interactions sociales.

3.1.6 Principes de référencement

Notre application est mobile, nous appliquerons donc les principes du ASO (App Store Optimization).

Mots-clés pertinents : les utilisateurs à la recherche d'une application comme la nôtre taperont des mots-clés qu'il nous faut avoir ciblés au préalable. Les mots comme "jeu", "game", "ambiance",



“devinettes”, “guess”, “guessing”, “Time’s Up”, “card” doivent renvoyer à notre app. Comme les mots-clés principaux sont déjà saturés par les applications les plus connues, il faudra prêter attention aux mots-clés secondaires comme “team”, “competition”, “quizz”, “trivia”, “celebrities” pour attirer plus d’utilisateurs.

Titre optimisé : Nous avons choisi DASH comme nom pour notre app, car il est court, évocateur (notion de rapidité, d’agilité), et avec son logo, est facilement mémorisable. Son sous-titre comportera un mot-clé principal comme “Devine !”

Description : La description est l’un des éléments clés que l’algorithme de Google Play Store prend en compte pour classer les applications dans les résultats de recherche. Il est donc important d’y intégrer les mots ou expressions clés sur lesquels nous souhaitons nous positionner. En principe, les mots-clés stratégiques doivent être présents entre 3 et 5 fois dans notre description.

Avis et notes : Les notes sont un levier que nous devons exploiter pour améliorer notre référencement. En effet, Google Play Store aime présenter aux internautes des applications qui ont beaucoup de feed-back positifs. Si nous avons de nombreux retours positifs, de plus en plus d’internautes nous feront confiance et téléchargeront notre application. Il est aussi important de répondre aux commentaires des utilisateurs.

Visuels attractifs : Nous ferons des captures d’écrans pour mettre en valeur l’interface attrayante de notre application.

Performances : Assurer une bonne performance de l’application (vitesse, stabilité) pour éviter les désinstallations, ce qui peut impacter le SEO.

Mises à jour régulières : Maintenir l’application à jour pour rester pertinent et satisfaire les critères des algorithmes d’app stores.

3.1.7 Exigences de performances et de volumétrie

Si l’on s’en réfère à la concurrence et à l’attractivité de notre produit, nous pouvons espérer de très bons chiffres de téléchargements de l’application. Un premier objectif serait atteint à partir de 500 000 téléchargements, que nous espérons atteindre en 6 mois maximum. Une application freemium a, en

moyenne, un taux de transformation de 3%¹. Notre application proposerait du contenu additionnel pour 2€ en moyenne. Nous pouvons donc tabler sur un CA de 30 000€ à 6 mois.

3.1.8 Multilinguisme & adaptations pour un public spécifique

Nous voulons nous concentrer sur la version française en premier lieu pour offrir une expérience qualitative en matière de contenu adapté à un public francophone.

Par la suite, nous envisageons une traduction en anglais avec une localisation pour les thèmes (remplacer les acteurs français par des acteurs britanniques ou américains par exemple). La traduction sera assurée par nos soins, nous sommes deux développeurs bilingues sur 3. Le résultat sera révisé par une traductrice certifiée.

3.1.9 Reprise de l'existant

Il n'y aucun existant, nous avons démarré le projet de 0.

3.2 Description graphique et ergonomique

3.2.1 Composants de la charte graphique

¹ Source : <https://adapty.io/blog/freemium-to-premium-conversion-techniques/>



Logo :

Le logo est une tête de renard stylisée, avec un design cartoon.

- **Couleurs** : Le logo utilise principalement des tons de rouge, orange, et blanc pour représenter le renard.
- **Forme** : Le renard a une expression malicieuse, avec des yeux grands et expressifs qui attirent immédiatement l'attention. Ses oreilles sont pointues et bien définies.
- **Style** : Le design est très moderne et épuré, avec des lignes douces et un style qui rappelle les avatars ou les mascottes utilisés dans les jeux vidéo ou les applications mobiles.
- **Texte** : Sous l'image du renard, le nom "DASH" est écrit en lettres majuscules, ajoutant une touche de dynamisme à l'ensemble du logo.

Ce logo est simple, mais efficace pour attirer l'attention, et il transmet une impression de rapidité et d'agilité, ce qui est en accord avec le nom "DASH".

Police et taille de caractères

Nous utilisons la police de caractères Roboto de Google. Elle a de nombreux avantages :

Lisibilité et clarté

Roboto a été spécialement conçue pour offrir une lisibilité optimale sur les écrans de petite taille. Ses formes simples et épurées garantissent que le texte reste lisible même sur des interfaces compactes ou

en conditions de faible luminosité, ce qui est essentiel pour les jeux mobiles où l'information doit être rapidement assimilée.

Esthétique moderne

Le design de Roboto est à la fois moderne et neutre, ce qui permet de l'intégrer harmonieusement dans une interface utilisateur sans détourner l'attention du contenu principal du jeu. Elle est particulièrement adaptée pour des jeux comme DASH, qui ont une esthétique contemporaine.

Polyvalence

Roboto est disponible en plusieurs épaisseurs (light, regular, bold, etc.), ce qui permet une grande flexibilité dans la hiérarchisation des informations à l'écran. Cette polyvalence est utile pour différencier les éléments importants, comme les scores, les titres de menus, et les boutons interactifs.

Compatibilité et performance

En tant que police native sur Android, Roboto est optimisée pour les performances sur les appareils mobiles, ce qui contribue à une expérience utilisateur fluide sans ralentissements. De plus, son adoption large assure une cohérence visuelle sur différents systèmes d'exploitation et appareils.

Palette de couleurs



1. #FF8D1A (Orange vif)

Cette teinte d'orange est dynamique et énergique. Cette couleur peut être utilisée pour les éléments interactifs ou les appels à l'action, attirant immédiatement l'attention sans être agressive.

2. #FF5733 (Rouge orangé)

Le rouge orangé est une couleur qui dégage une impression d'urgence et d'excitation. Utilisée de manière judicieuse, elle peut accentuer les éléments clés de l'interface, comme les boutons importants ou les notifications. Elle évoque également une sensation de passion et de détermination, ce qui renforce l'engagement des joueurs.

3. #2196F3 (Bleu vif)

Le bleu vif apporte une touche de fraîcheur et de confiance à la palette. Il équilibre les tons chauds avec une sensation de calme et de fiabilité.

4. #FFC300 (Jaune vif)

Le jaune attire l'attention et peut être utilisé pour mettre en évidence des informations importantes sans être trop envahissant.

Ensemble, la combinaison des teintes chaudes (orange et rouge orangé) avec le bleu froid crée un contraste visuel fort qui attire l'attention et maintient l'intérêt des utilisateurs.

3.2.2 Design, Responsive design et autres exigences liées au design

Non concerné, nous faisons une application mobile pour téléphones et tablettes. Le design est donc naturellement adapté à ces écrans.

3.2.3 Maquettes



La page d'accueil de l'application. Deux boutons principaux, l'un pour jouer sans être authentifié, l'autre pour s'authentifier au préalable. Un bouton hamburger donne accès aux réglages et au profil utilisateur.





La page “Profil Utilisateur” comporte plusieurs éléments comme la gestion du thème personnalisé, l'accès à l'historique des parties, aux statistiques du compte et à la liste d'amis. Un bouton JOUER bien visible est présent ainsi qu'un bouton de déconnexion.



La page “Themes” propose les 20 thèmes + le thème personnalisable. On sélectionne de 1 à 21 thèmes avant de pouvoir passer à la page suivante.



La page “Options” permet de choisir le nombre d’équipes, leur nom, le nombre de joueurs par équipes et le chronomètre.



La page “Game” est la page de jeu. En haut, on affiche la manche et l’équipe qui est en train de jouer. Puis les scores des différentes équipes sont incrémentés en direct. La carte à faire deviner est au centre, au-dessus du chronomètre et de ses boutons lecture/pause ou réinitialiser.

Enfin il y a deux boutons pour valider ou passer une carte, mais il sera proposé un système de swipe pour une meilleure ergonomie.

3.3 Besoins fonctionnels « métier »

3.3.1 Utilisateurs du projet

Voici les différents types d'utilisateurs de notre application.

Joueurs : Utilisateurs finaux de l'application qui participent au jeu, créent des comptes, et interagissent avec les différentes fonctionnalités (choix de thèmes, équipes, chronomètre, etc.).

Administrateurs : Personnes responsables de la gestion des contenus (thèmes, avatars, etc.), de la modération et de la gestion des utilisateurs.

Développeurs : Équipe technique qui maintient et améliore l'application.

Marketeurs : Équipe en charge de la promotion de l'application, des campagnes de fidélisation, et de l'analyse des données utilisateurs pour optimiser l'expérience.

Support client : Personnel chargé de l'assistance aux utilisateurs, gestion des plaintes et des bugs, et remontée des problèmes techniques aux développeurs.

3.3.2 Processus utilisateur impacté

Les processus impactés par l'application DASH incluent :

Comptabilité : Gestion des achats in-app, abonnements, suivi des transactions et reporting financier.

Gestion des données : Suivi et analyse des comportements des utilisateurs pour améliorer l'expérience, et gestion des comptes utilisateurs.

Marketing : Campagnes de marketing digital, gestion de la fidélisation client, et analyse des données pour ajuster les stratégies marketing.

Support client : Assistance et résolution des problèmes rencontrés par les utilisateurs, gestion des retours utilisateurs.

Ressources humaines (RH) : Gestion de l'équipe interne, des tâches assignées et des objectifs de performance.

3.3.3 Informations relatives aux contenus

Types de contenus :

- **Thèmes de jeu** : Listes de thèmes préétablis avec des cartes de jeu.
- **Avatars et illustrations** : Graphismes utilisés pour personnaliser l'expérience utilisateur.
- **Articles et tutoriels** : Guides et articles expliquant les règles du jeu, les fonctionnalités de l'application, et d'autres ressources pour les utilisateurs.
- **Ressources téléchargeables** : Peut inclure des thèmes supplémentaires, avatars, ou autres contenus premium.

Droit à l'image, copyrights, et DRM :

- **Droit à l'image** : Nous utilisons des créations originales pour nous assurer de ne pas faire d'atteinte au droit à l'image.
- **Copyrights** : S'assurer que tous les contenus (images, textes, vidéos) sont protégés par les droits d'auteur ou utilisés sous licence.
- Se dédouaner de tout ce que l'utilisateur peut créer à travers son thème personnalisé.

Règlement général sur la protection des données (RGPD) :

- **Collecte de données** : Les données des utilisateurs doivent être collectées de manière transparente et avec leur consentement explicite. Cela inclut les informations personnelles telles que les noms, adresses e-mail, et données de connexion.
- **Stockage et traitement des données** : Les données doivent être stockées de manière sécurisée, en conformité avec le RGPD, et ne doivent être utilisées que pour les finalités spécifiées (amélioration du service, personnalisation, etc.).
- **Droits des utilisateurs** : Les utilisateurs doivent pouvoir exercer leurs droits, comme le droit à l'accès, à la rectification, et à l'effacement de leurs données. Un processus clair doit être mis en place pour gérer ces demandes.

3.3.4 Inventaire des besoins fonctionnels

Thème	Qui	Quoi	Pourquoi
Authentification	Utilisateur	Me connecter	Accéder au site
Authentification	Utilisateur	Me déconnecter	Quitter le site
Authentification	Utilisateur	Gérer mon compte	Mettre à jour mon profil utilisateur
Démarrer une partie	Utilisateur	Choisir des thèmes	Ajouter des thèmes à la partie
Démarrer une partie	Utilisateur	Sélectionner des options	Modifier les options d'une partie
Jouer	Utilisateur	Démarrer une partie	Jouer
Réglages	Utilisateur	Modifier les thèmes personnalisables	Permettre aux utilisateurs de jouer avec des thèmes dont ils auront écrit les éléments..
Réglages	Utilisateur	L'utilisateur peut modifier les réglages de l'application	Permettre une meilleure ergonomie du jeu et une expérience utilisateur plus satisfaisante.
Inviter des amis	Utilisateur	Inviter des amis à télécharger l'application	Permettre aux utilisateurs de se connecter, se lancer des défis...

Podium	Utilisateur	Consulter le podium à la fin d'une partie	Pour consulter les points en fin de partie
Règles	Utilisateur	Consulter les règles du jeu	Pour apprendre à jouer lorsque l'on télécharge l'application.
Achats	Utilisateur	Achat de nouveau pack de thème	Pour augmenter le nombre de cartes ou de thème

3.4 Prestations attendues

3.4.1 Livrables et prestations

Voici les livrables et les prestations qui sont de notre ressort :

- Cahier des charges
- Spécifications fonctionnelles & techniques
- Maquettage & design
- Développement
- Intégration
- Gestion de la base de données
- Achat du nom de domaine et gestion de l'hébergement
- Maintenance et mises à jour
- Accompagnement marketing : plan marketing, ASO, SMO (Social Media Optimization – Optimisation des Médias Sociaux - développer la visibilité de l'app au travers des médias sociaux)

3.4.2 Budget

Ceci est hypothétique, nous sommes une équipe de trois développeurs qui travaillons sur cette application durant notre temps libre.

Frais de développement technique :

Salaires et honoraires : trois développeurs juniors rémunérés à un taux horaire de 30€, pour une période de 6 mois, travaillant 25 jours par mois, 7 heures par jour : 95 040€.

Frais opérationnels :

Frais de bureau : 12 000€ par an

Coûts d'infrastructure technique et d'hébergement : 800€ par an (susceptible d'augmenter avec l'accroissement du trafic utilisateur)

Total première année : 107 640€

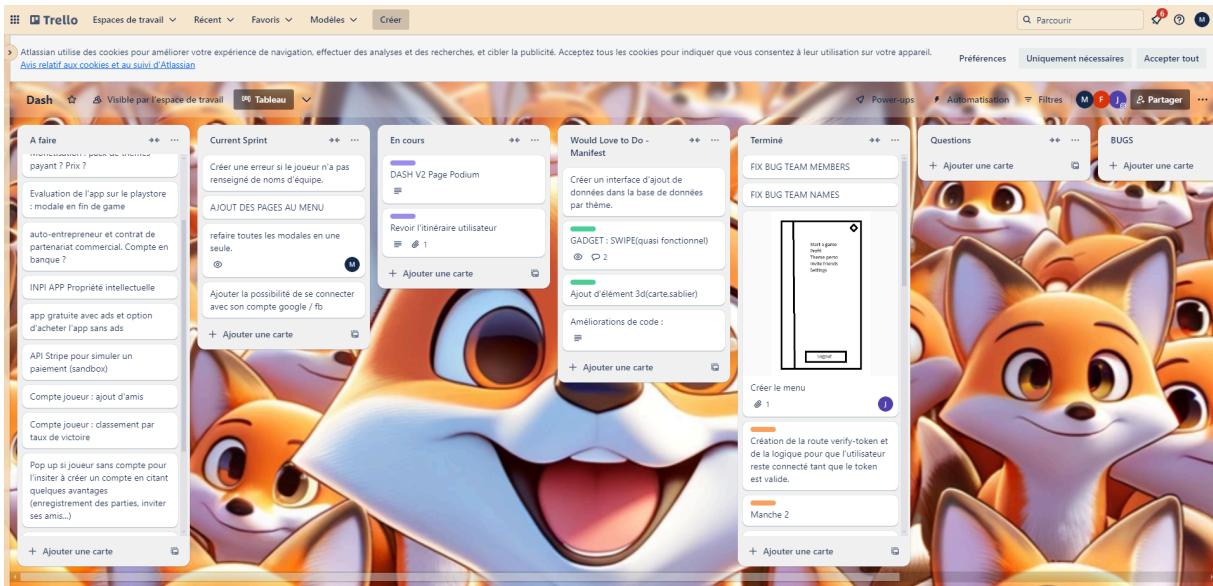
Marge d'erreur : 15% => 124 000€

4. GESTION DE PROJET

4.1 Méthodologie

La méthodologie AGILE est une façon de gérer un projet en restant flexible, en collaborant étroitement avec l'équipe, et en cherchant constamment à s'améliorer. L'idée de base, c'est de travailler par petites étapes, qu'on appelle des "sprints", pour livrer des fonctionnalités du produit au fur et à mesure. L'AGILE met l'accent sur l'importance des personnes et des échanges plutôt que sur des processus rigides, sur le fait de collaborer régulièrement avec le client, de s'adapter rapidement aux changements, et de livrer fréquemment des versions fonctionnelles du logiciel.

Dans le cadre de mon projet, j'ai mis en œuvre des principes AGILE avec des sprints de 2 à 4 semaines. Chaque sprint commençait par une planification des tâches à accomplir, suivie de réunions plusieurs fois par semaine (une mise à jour écrite sur discord pour chacun de nous) pour suivre l'avancement. À la fin de chaque sprint, une revue était effectuée pour présenter les fonctionnalités développées et recueillir les retours, permettant ainsi d'ajuster les priorités pour le sprint suivant. J'ai également utilisé des outils comme Trello pour la gestion des tâches et Discord pour la communication avec l'équipe, assurant ainsi une transparence et une collaboration efficace tout au long du projet.



4.2 Outils, planning et suivi

Nous avons mis au point un rétroplanning découpé en plusieurs phases afin d'organiser notre année de travail :

Phase de planification (Mois 1) :

- Définir les objectifs du projet.
- Identifier les fonctionnalités clés de l'application.
- Élaborer une ébauche du cahier des charges.
- Répartir les rôles et les responsabilités au sein de l'équipe.

Phase de conception (Mois 2-3) :

- Créer des wireframes et des maquettes de l'interface utilisateur.
- Définir l'architecture globale de l'application.
- Choisir les technologies à utiliser.

Phase de développement (Mois 4-10) :

- Développer les fonctionnalités de base de l'application.
- Effectuer des tests unitaires et d'intégration tout au long du processus.

Phase de tests (Mois 11) :

- Effectuer des tests de validation et de performance.
- Corriger les bugs et les problèmes identifiés.

Phase de gestion administrative et juridique (Mois 11-12) :

- CGU, propriété intellectuelle, dépôt de la marque.
- Création d'entreprise.

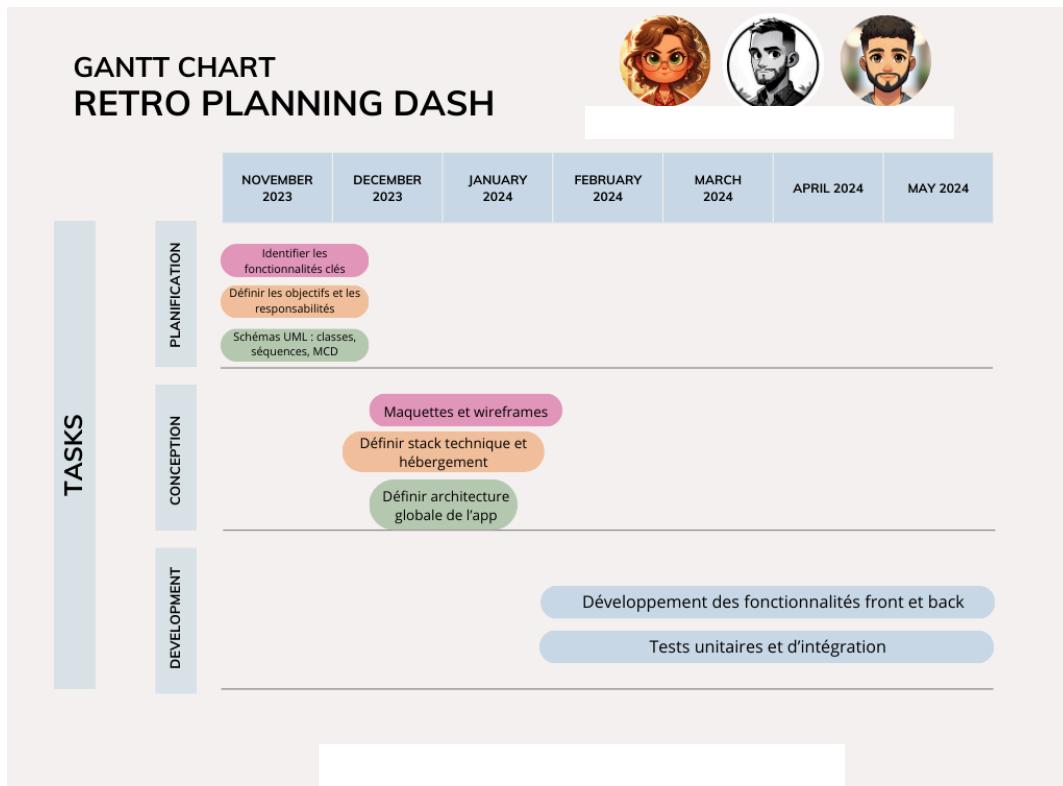
Phase de lancement (Mois 12) :

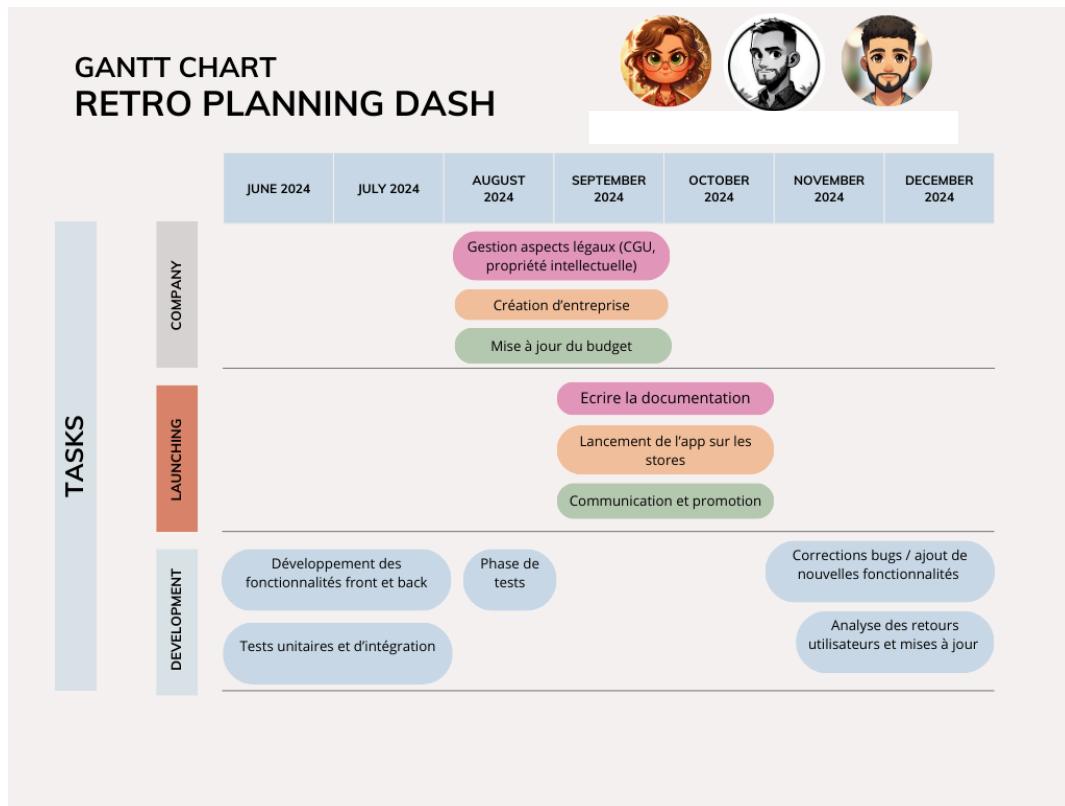
- Préparer la documentation de l'application.
- Soumettre l'application aux boutiques d'applications (App Store, Google Play, etc.).
- Communiquer sur le lancement de l'application.

Phase de maintenance et d'améliorations (après le lancement) :

- Suivre les retours des utilisateurs et les évaluations de l'application.
- Corriger les bugs et apporter des mises à jour régulières.
- Ajouter de nouvelles fonctionnalités en fonction des besoins et des retours.

Visualisation du rétroplanning via un diagramme de Gantt :





5. SPÉCIFICATIONS FONCTIONNELLES

5.1 Description de la solution

5.1.1 Caractéristiques de la solution

Découpage de la solution en 4 parties : les thèmes, les options, le compte utilisateur et le jeu.

Les thèmes

Objectif :

Permettre aux utilisateurs de sélectionner des thèmes de jeu, incluant des thèmes prédéfinis et des thèmes personnalisables.

Entrées :

- Sélection des thèmes dans une liste.
- Possibilité d'ajouter un thème personnalisé (texte saisi par l'utilisateur).

Sorties :

- Les thèmes sélectionnés sont ajoutés à la session de jeu.
- Affichage des thèmes sélectionnés avec une confirmation visuelle.

Règles de gestion :

- L'utilisateur peut sélectionner entre 1 et 21 thèmes.
- Les thèmes payants sont disponibles uniquement après achat.
- Le thème personnalisé doit contenir entre 10 et 80 éléments.
- Si aucun thème n'est sélectionné, un message d'erreur s'affiche demandant à l'utilisateur de choisir au moins un thème.

Contraintes fonctionnelles :

- Les thèmes doivent être stockés dans une base de données et mis à jour dynamiquement selon les choix de l'utilisateur.
- Les modifications apportées aux thèmes (ajout, suppression) doivent être sauvegardées en temps réel.
- Le thème personnalisé ne doit apparaître qu'aux utilisateurs authentifiés.

Les options

Objectif :

Permettre aux utilisateurs de définir leurs options de jeu (nombre d'équipes, noms d'équipes, nombre de joueurs par équipe et chronomètre) en leur proposant une liste d'options prédéfinies et un chemin à suivre.

Entrées :

- Nombre d'équipes (de 2 à 6).
- Noms des équipes saisis par l'utilisateur.
- Nombre de joueurs par équipe (de 2 à 6).
- Durée du tour (30, 45, ou 60 secondes).

Sorties :

- Les options sélectionnées sont enregistrées pour la session de jeu.

- Affichage des options choisies pour confirmation avant le démarrage du jeu.

Règles de gestion :

- Le nombre total de joueurs doit être supérieur ou égal à 2.
- La durée du tour sélectionnée s'applique à toutes les équipes pour la session en cours.
- Les options doivent être modifiables avant le début du jeu, mais verrouillées une fois la partie lancée.

Contraintes fonctionnelles :

- L'interface doit permettre une navigation fluide entre les différentes options.
- Les changements d'options doivent être immédiatement visibles dans l'interface utilisateur.

L'authentification

Objectif :

Gérer la création, la connexion et la déconnexion des comptes utilisateurs.

Entrées :

- Email et mot de passe pour l'inscription et la connexion.
- Informations supplémentaires pour l'inscription (nom d'utilisateur, avatar).

Sorties :

- Confirmation de l'inscription ou de la connexion réussie.
- Redirection vers la page compte utilisateur après la connexion.
- Stockage sécurisé des informations utilisateur.

Règles de gestion :

- Les mots de passe doivent être chiffrés et sécurisés.
- Un utilisateur doit être authentifié pour accéder à certaines fonctionnalités (ex : personnalisation des thèmes).
- La session utilisateur doit expirer après un certain temps d'inactivité (ex : 1 heure).

Contraintes fonctionnelles :



- Utilisation de JWT pour sécuriser les sessions.
- Les erreurs d'authentification doivent être clairement indiquées à l'utilisateur (ex : mot de passe incorrect).

Le jeu

Objectif :

Gérer le déroulement d'une partie, y compris la distribution des cartes, le suivi des scores, et le contrôle du chronomètre.

Entrées :

- Démarrage du jeu après sélection des thèmes et des options.
- Actions des joueurs (valider/passer une carte, répondre, etc.).
- Chronomètre (démarrage, pause, réinitialisation).

Sorties :

- Affichage de la carte à deviner pour chaque équipe.
- Mise à jour en temps réel des scores et du temps restant.
- Affichage du podium et des résultats à la fin de la partie.

Règles de gestion :

- Chaque équipe doit deviner un ensemble de 42 cartes, réparties sur 3 manches.
- Les cartes non devinées sont remises dans le paquet pour les tours d'équipes suivants.
- Les scores doivent être calculés en fonction des cartes devinées..

Contraintes fonctionnelles :

- L'interface doit être réactive pour permettre un jeu fluide sans latence.
- Le chronomètre doit être précis et visible en permanence pendant le tour d'une équipe.
- Les transitions entre les manches doivent être fluides et clairement indiquées aux joueurs.

5.1.2 Acteurs du projet

1. Direction métier :

- **Rôle :** Supervise l'alignement du projet avec les objectifs stratégiques de l'entreprise (ou du groupe de développeurs dans le cas de DASH). S'assure que l'application répond bien aux besoins du marché cible.
- **Responsabilités :** Valider les grandes orientations du projet, superviser l'avancement général, et intervenir en cas de besoin pour réorienter le projet.

2. Utilisateurs clés (joueurs) :

- **Rôle :** Les utilisateurs finaux de l'application, qui interagissent directement avec les fonctionnalités. Leur retour d'expérience est crucial pour les phases de test et d'amélioration continue.
- **Responsabilités :** Participer aux tests utilisateurs, fournir du feedback pour améliorer l'ergonomie et la jouabilité, et signaler les bugs ou incohérences.

3. Responsable de l'exploitation :

- **Rôle :** Assure la bonne marche de l'application une fois mise en production. Gère les serveurs, les bases de données, et s'assure que l'infrastructure technique supporte la charge des utilisateurs.
- **Responsabilités :** Suivi des performances, gestion des mises à jour, monitoring des systèmes, et résolution des incidents techniques.

4. Responsable du support :

- **Rôle :** Gère les relations avec les utilisateurs en cas de problème, en offrant une assistance technique et en remontant les problèmes critiques à l'équipe de développement.
- **Responsabilités :** Répondre aux demandes d'assistance des utilisateurs, documenter les problèmes récurrents, et travailler avec les développeurs pour résoudre les bugs et améliorer l'application.

5.1.3 Applications connexes

Play Store :

- **Lien avec notre app :** C'est la plateforme de distribution de l'application. Les mises à jour de l'application, les avis des utilisateurs, et la visibilité de l'application dépendent du Play Store.
- **Conséquences de modifications :** Toute modification du Play Store (comme les règles de soumission d'applications, les changements dans l'algorithme de référencement, etc.) peut affecter la distribution et la visibilité de DASH. Les bugs liés à la compatibilité avec les nouvelles



versions du Play Store peuvent également impacter les téléchargements ou l'expérience utilisateur.

Systèmes de paiement intégrés (Google Pay) :

- **Lien avec notre app** : Gère les achats in-app pour les thèmes payants ou autres contenus premium.
- **Conséquences de modifications** : Des changements dans les systèmes de paiement peuvent affecter les transactions des utilisateurs, nécessitant des mises à jour pour garantir que les paiements sont traités correctement.

AWS (hébergeur du serveur) :

- **Lien avec notre app** : AWS fournit l'hébergement pour les serveurs, les bases de données, et autres infrastructures nécessaires pour faire fonctionner l'application.
- **Conséquences de modifications** :
 - **Mises à jour de services** : Si AWS change ou met à jour ses services, cela peut affecter le fonctionnement de l'application. Par exemple, un changement dans le service de base de données pourrait nécessiter des ajustements dans la manière dont l'application interagit avec cette base.
 - **Disponibilité et scalabilité** : AWS est crucial pour garantir que notre application puisse gérer des pics de trafic et rester disponible en tout temps. Toute modification de la configuration AWS (comme le redimensionnement automatique des serveurs) peut avoir un impact direct sur les performances de DASH.
 - **Sécurité** : AWS offre des services de sécurité, mais toute modification dans les configurations de sécurité, comme les politiques IAM (Identity and Access Management), pourrait affecter l'accès et la sécurité des données dans notre application.

5.1.4 Documentation référence

Manuel utilisateur :

Ce document détaille les fonctionnalités de l'application du point de vue de l'utilisateur final. Il explique comment naviguer dans l'application, comment jouer, gérer les paramètres, etc.

Guide d'installation :

Document qui explique comment installer l'application, configurer l'environnement (pour les développeurs ou les utilisateurs avancés), et résoudre les problèmes d'installation courants.

Cahier des charges :

Document initial qui décrit les besoins et les attentes du client pour l'application. Il sert de base pour toutes les spécifications et le développement ultérieur.

5.1.5 Lexique/glossaire

API (Application Programming Interface) : Ensemble de fonctions et de procédures permettant la création d'applications qui accèdent aux fonctionnalités ou aux données d'un autre service.

ASO (App Store Optimization) : Ensemble de techniques visant à optimiser la visibilité d'une application mobile dans les magasins d'applications (ex : Play Store, App Store).

Back-end : Partie d'une application qui gère la logique métier, les calculs, les bases de données, et les API, souvent invisible pour l'utilisateur final.

Freemium : Modèle économique où l'application est gratuite avec des fonctionnalités payantes additionnelles.

Front-end : Partie visible de l'application avec laquelle l'utilisateur interagit directement.

IHM (Interface Homme-Machine) : Interface utilisateur permettant l'interaction entre l'utilisateur et le système.

JWT (JSON Web Token) : Standard ouvert pour la création de jetons d'accès, souvent utilisé pour gérer l'authentification dans les applications web.

Manche : Dans le cadre de l'application DASH, phase spécifique du jeu où les joueurs doivent deviner les cartes selon des règles différentes (ex : manche 1 - devinette libre, manche 2 - un mot, manche 3 - mime).

RGPD (Règlement Général sur la Protection des Données) : Règlement européen qui encadre le traitement des données personnelles au sein de l'Union européenne.

SFD (Spécifications Fonctionnelles Détaillées) : Document décrivant en détail les fonctionnalités d'une application, comment elles doivent être mises en œuvre techniquement.

Thème : Dans le cadre de l'application DASH, un thème est un regroupement d'éléments qui partagent une même catégorie (histoire, géographie, anime, acteurs/actrices etc.)



UX (User Experience) : Ensemble des éléments relatifs à l'expérience globale vécue par l'utilisateur lors de l'utilisation d'une application ou d'un site web.

5.1.6 Aspect métier

5.1.6.1 Criticité de l'application

Il s'agit de déterminer à quel point l'application est critique pour les utilisateurs. Pour DASH, bien que ce soit une application de divertissement, elle peut être considérée comme critique pour les utilisateurs lors de moments spécifiques, comme des soirées ou des événements sociaux où l'application est le centre de l'activité.

L'indisponibilité de l'application pendant une période clé (par exemple, une soirée entre amis) pourrait fortement impacter l'expérience des utilisateurs, créant de la frustration et potentiellement nuisant à la réputation de l'application.

Criticité de la population qui utilise l'application

La criticité de la population qui utilise l'application est moyenne. L'application sera principalement utilisée par des adultes en soirée, ou en week-end. Il est crucial que DASH soit disponible à ces moments-là, néanmoins, il sera possible de planifier des maintenances en journée quand le jeu sera le moins sollicité par les utilisateurs.

Nombre d'utilisateurs

A l'heure actuelle, nous n'avons pas de données puisque l'application n'est pas encore publiée.

Disponibilité (Key Performance Indicator)

SLA (Service Level Agreement) : Pour une application non critique mais de divertissement comme DASH (où l'expérience utilisateur est la clé de la réussite de l'app), il est courant d'avoir un SLA de 99.9%. Cela signifie que l'application doit être disponible en permanence, et surtout pendant les pics d'utilisation en soirée et week-ends.

Support et maintenance

Le support technique, assuré par les trois développeurs, sera disponible 6 jours sur 7.

Les maintenances seront effectuées le lundi, le jour le moins propice à l'utilisation de l'application par les utilisateurs.

Impacts indisponibilité de service

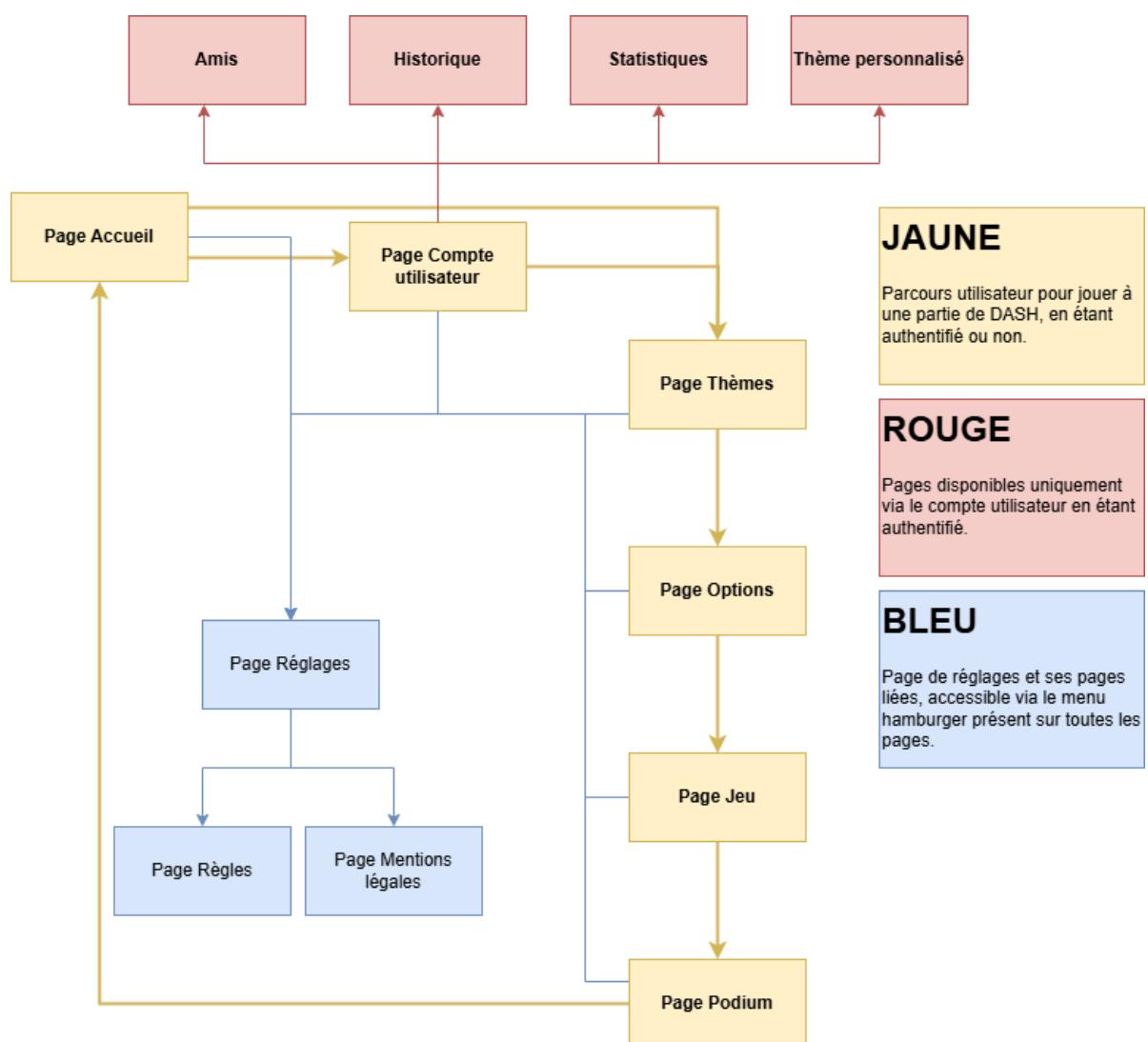
Que se passe-t-il si l'application devient indisponible ? Pour DASH, cela signifie une soirée gâchée pour des utilisateurs, une perte de confiance dans l'application, et potentiellement des avis négatifs.

Si l'application génère des revenus (via des achats in-app, par exemple), l'indisponibilité pourrait entraîner une perte financière directe.

Une indisponibilité prolongée pourrait nuire à la réputation de l'application, réduisant ainsi le taux de rétention des utilisateurs et leur fidélité.

5.1.6.2 Cartographie

Schéma de l'arborescence de l'application



Page d'accueil :

- Boutons principaux (Jouer sans connexion, Connexion)
- Menu Hamburger (réglages, profil, etc.)

Page de connexion/inscription :

- Formulaire de connexion
- Option de récupération de mot de passe

Page de profil utilisateur :

- Gestion des thèmes personnalisés
- Historique des parties
- Statistiques
- Liste d'amis

Page de sélection des thèmes :

- Liste des thèmes disponibles
- Thème personnalisé

Page de configuration des options de jeu :

- Nombre d'équipes
- Noms des équipes
- Nombre de joueurs par équipe
- Chronomètre

Page de jeu active :

- Cartes à deviner
- Chronomètre
- Compteur de points par équipe
- Boutons valider / passer
- Bandeau informatif sur la manche et l'équipe en cours

Page des résultats/podium :

- Classement des équipes



- Boutons rejouer ou retour à l'accueil

Page de réglages :

- Paramètres de l'application
- Mentions légales
- Options d'affichage
- Notifications

5.2 Fonctionnalités

5.2.1 Matrice Profil/Droits

Type d'utilisateur	Utilisateur	Inscrit	Membre
Fonctionnalité			
Authentification			
Créer un compte	X		
Se connecter		X	X
Se déconnecter		X	X
Commencer une partie			
Choisir des thèmes	X	X	X
Sélectionner des options	X	X	X
Créer des thèmes personnalisé		X	X
Jouer			
Jouer une partie	X	X	X
Consulter les résultats	X	X	X
Gestion de compte			
Ajouter des amis		X	X
Laisser un avis		X	X
Accéder à ses historiques de parties		X	X
Accéder au classement DASH		X	X
Pack de cartes additionnel			X

5.2.2 Fonctionnalités Générales

5.2.2.1 Diagramme de cas d'utilisation global



5.3.1 Fonctionnalités détaillées

5.3.1.1 Sélection des thèmes

Description

La fonctionnalité de sélection des thèmes permet aux utilisateurs de choisir parmi une variété de thèmes disponibles au lancement d'une partie. Les joueurs peuvent sélectionner de un à vingt-et-un thèmes, chaque thème contient 80 cartes avec des éléments à deviner. Chaque thème est représenté par une icône, on presse pour sélectionner le thème, on presse à nouveau pour le désélectionner. Puis on presse le bouton vert avec la flèche pour passer à la suite. Si aucun thème n'est sélectionné, un message d'erreur s'affiche.

Diagramme d'activité

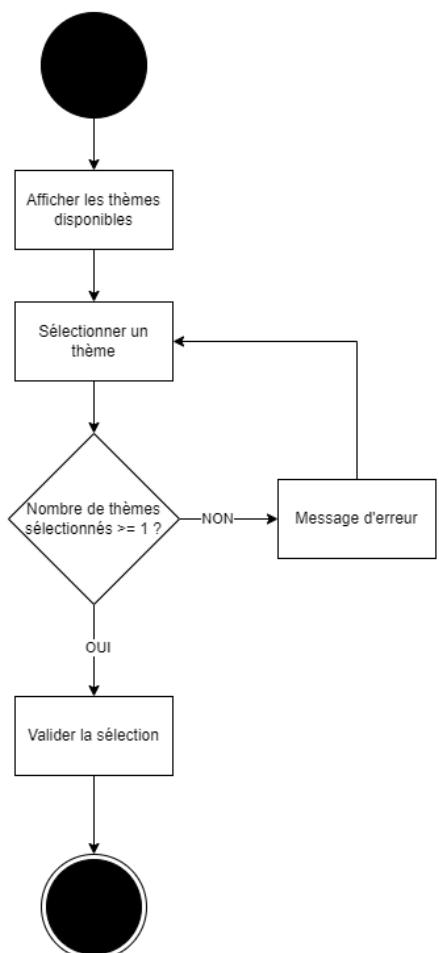
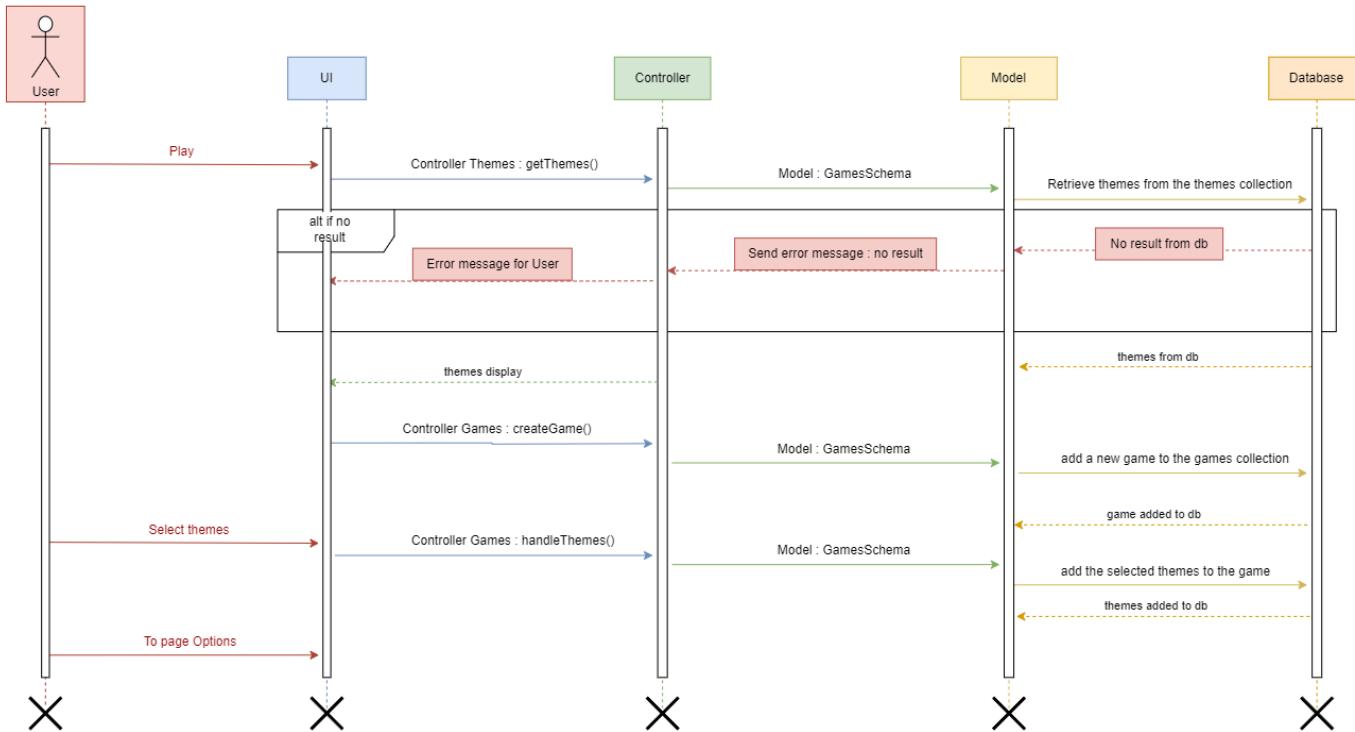


Diagramme de séquence

Diagramme de séquence : fonctionnalité des thèmes au lancement d'une partie



Données/Actions

1. Données en entrée :

- **Description :** Les données fournies à la fonctionnalité sont les ID des thèmes sélectionnés par l'utilisateur.
- **Détail des données :**
 - **Name:** theme_id
 - **Type:** alphanumeric (hexadecimal)
 - **Length:** 24 characters
 - **Semantic:** Identifiant unique généré par MongoDB pour chaque thème.

- **Référence values list:** Tous les ObjectIds valides de la collection des thèmes dans MongoDB.
- **Value limits:** Doit être une chaîne de 24 caractères hexadécimaux (0-9, a-f). Doit être un identifiant existant dans la liste des thèmes disponibles.

2. Données gérées (traitement) :

- **Description :** Une fois que l'utilisateur a sélectionné les thèmes, le système traite ces sélections pour préparer le jeu.
- **Détail des données :**
 - **Name :** selected_themes
 - **Code :** Liste des IDs de thèmes sélectionnés.
 - **Abbreviation/Short description :** Themes
 - **Type :** array of alphanumeric.
 - **Length :** Jusqu'à 21 éléments, chaque élément étant un theme_id.
 - **Semantic :** Représente l'ensemble des thèmes que l'utilisateur a choisis pour la partie en cours.
 - **Référence values list :** Liste des IDs des thèmes sélectionnés.
 - **Value limits :** Entre 1 et 21 IDs.

3. Données générées (en sortie) :

- **Description :** Les données générées sont les tableaux de strings associés aux thèmes sélectionnés, utilisés lors du jeu pour la devinette.
- **Détail des données :**
 - **Name :** theme_elements
 - **Code :** Tableau contenant les éléments à deviner pour chaque thème.
 - **Abbreviation/Short description :** Elements
 - **Type :** array of strings
 - **Length :** 80 strings par thème sélectionné.
 - **Semantic :** Contient les éléments à deviner par les joueurs dans le jeu, associés aux thèmes sélectionnés.
 - **Référence values list :** Liste des strings associées à chaque thème.
 - **Value limits :** 80 éléments par thème, string non vide.

Résumé :

- **Entrée** : Les IDs des thèmes sélectionnés (theme_id).
- **Traitement** : La gestion de la sélection dans un tableau (selected_themes).
- **Sortie** : Les éléments à deviner (theme_elements), organisés par thème.

Ecran/Affichage



5.3.1.2 Fonctionnalité 2 : Les options

Description

Les options, malgré leur nom, sont l'étape obligatoire avant de lancer le jeu, après la sélection des thèmes. Elles sont au nombre de 4.

L'utilisateur sélectionne parmi un choix d'icônes le nombre d'équipes.

Puis il entre manuellement les noms de chaque équipe.

Il précise combien de joueurs chaque équipe comprend en pressant une icône.

Enfin, il choisit une durée de chronomètre toujours en pressant une icône.

Diagramme d'activité

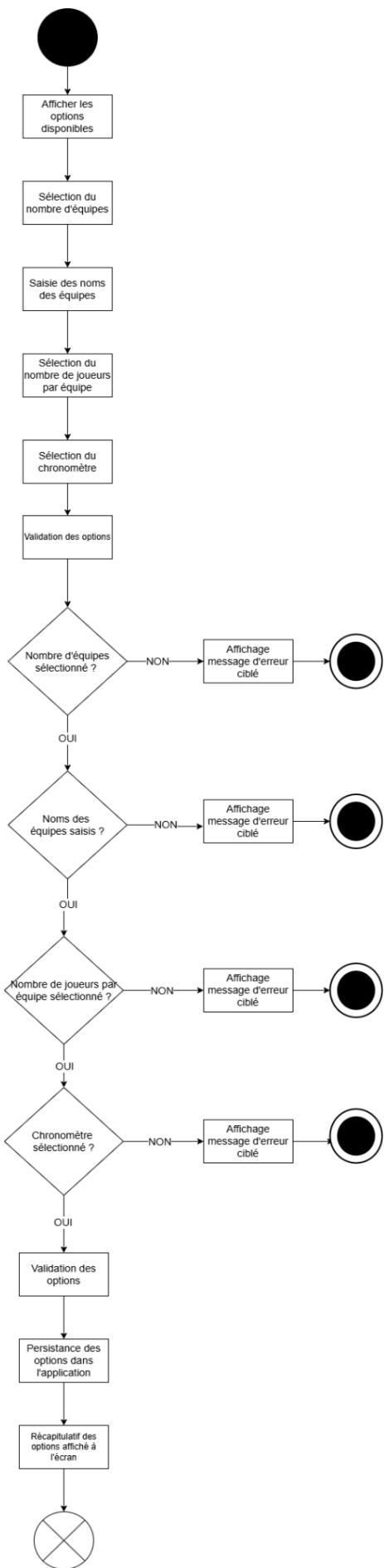
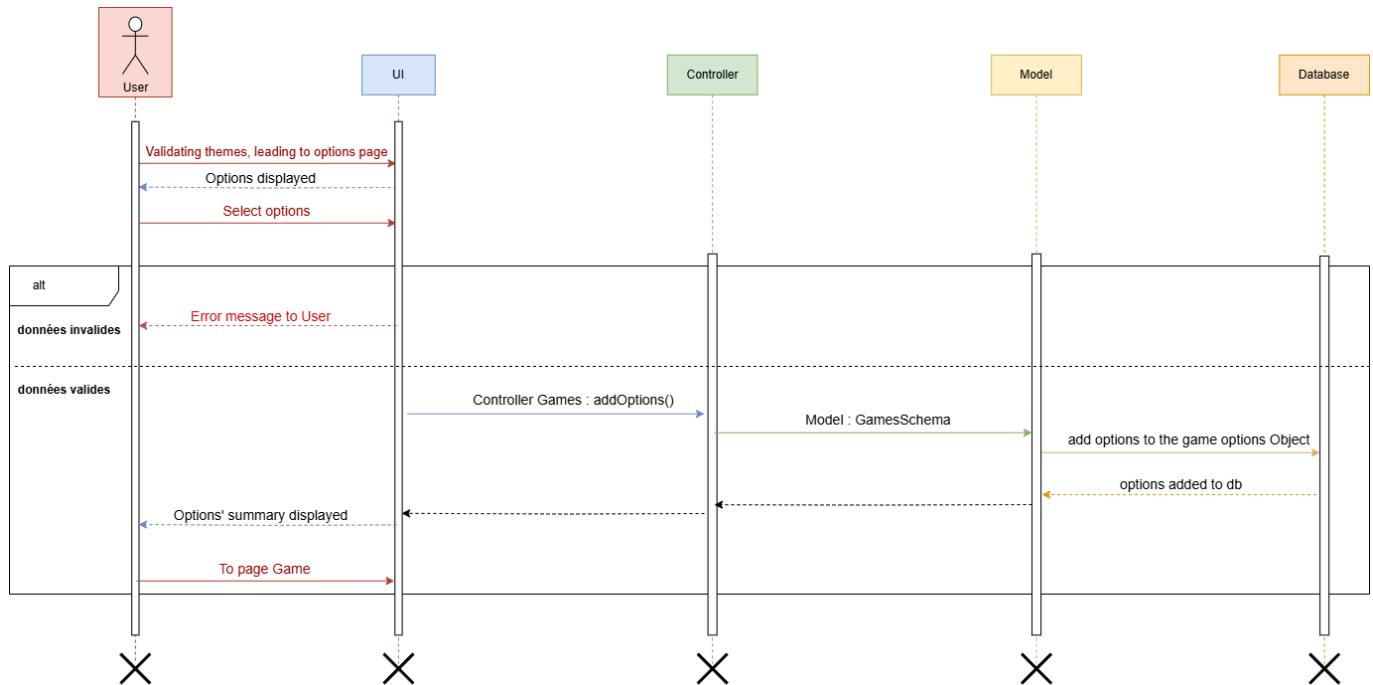


Diagramme de séquence



Données

Description : Les données fournies à la fonctionnalité des options sont les choix faits par l'utilisateur concernant le nombre d'équipes, les noms des équipes, le nombre de joueurs par équipe, et la durée du chronomètre.

Détail des données en entrée :

- Name : nb_teams
 - Type : numeric
 - Length : 1 digit (ex : 2, 6)
 - Semantic : Représente le nombre d'équipes que l'utilisateur souhaite créer pour la partie.
 - Référence values list : 2 à 6 (valeurs possibles pour le nombre d'équipes).
 - Value limits : Doit être un entier compris entre 2 et 6.
- Name : team_names
 - Type : array of strings
 - Length : Variable (selon le nombre d'équipes sélectionnées, chaque string peut avoir une longueur jusqu'à 30 caractères).
 - Semantic : Représente les noms des équipes définis par l'utilisateur.
 - Référence values list : Aucun, dépend des saisies de l'utilisateur.

- Value limits : Doit correspondre au nombre d'équipes sélectionnées (2 à 6 strings).
- Name : team_members
 - Type : numeric
 - Length : 1 digit (ex : 2, 6)
 - Semantic : Représente le nombre de joueurs par équipe.
 - Référence values list : 2 à 6 (valeurs possibles pour le nombre de joueurs par équipe).
 - Value limits : Doit être un entier compris entre 2 et 6.
- Name : chrono
 - Type : numeric
 - Length : 2 digits (ex : 30, 45, 60)
 - Semantic : Représente la durée du chronomètre pour chaque tour de jeu.
 - Référence values list : 30, 45, 60 (valeurs possibles pour le chronomètre en secondes).
 - Value limits : Doit être l'une des valeurs de la liste de référence.

Données gérées (traitement) pour les options :

Description : Une fois que l'utilisateur a configuré les options, le système gère ces choix pour préparer la partie.

Détail des données :

- Name : selected_options
 - Code : Ensemble des options choisies par l'utilisateur.
 - Abbreviation/Short description : Options
 - Type : object
 - Length : Dépend des valeurs sélectionnées pour chaque option.
 - Semantic : Contient toutes les options définies par l'utilisateur, à savoir nb_teams, team_names, team_members, et chrono.
 - Référence values list : Les choix de l'utilisateur basés sur les listes de références pour chaque option.
 - Value limits : Doit contenir des valeurs valides pour toutes les options.

Données générées (en sortie) pour les options :

Description : Les données générées sont les configurations finalisées des options qui seront utilisées tout au long du jeu.

Détail des données :

- Name : game
 - Code : Configuration du jeu basée sur les options sélectionnées.
 - Abbreviation/Short description : game
 - Type : object
 - Length : Variable, en fonction des options choisies.
 - Semantic : Contient la configuration complète du jeu pour la session en cours, basée sur les options fournies et validées par l'utilisateur.
 - Référence values list : Liste des configurations possibles selon les options sélectionnées.
 - Value limits : La configuration doit être cohérente avec les valeurs limites des options choisies.

Résumé :

- Entrée : Les choix de l'utilisateur pour les options (nb_teams, team_names, team_members, chrono).
- Traitement : La gestion de ces choix dans un objet (selected_options) pour préparer le jeu.
- Sortie : La configuration finale du jeu (game), qui sera utilisée pour diriger la partie.

Ecran/Affichage



5.3.1.3 Fonctionnalité 3 : le jeu

Description

Une fois les thèmes et les options choisis, le jeu se lance. Un joueur de la première équipe a {durée du chronomètre} pour faire deviner un maximum de cartes à son équipe. Une fois le temps écoulé, c'est à un joueur de la deuxième équipe de faire de même. Jusqu'à ce que toutes les cartes aient été devinées.

Le score s'incrémente à chaque carte validée en direct pour l'équipe qui joue. Il est possible de mettre en pause ou de réinitialiser le chronomètre. Le jeu passe automatiquement à la manche suivante quand le paquet de cartes est vide. En manche 3, il passe à la page Podium et affiche les scores.

Diagramme d'activité

Diagramme d'activité de la fonctionnalité de jeu

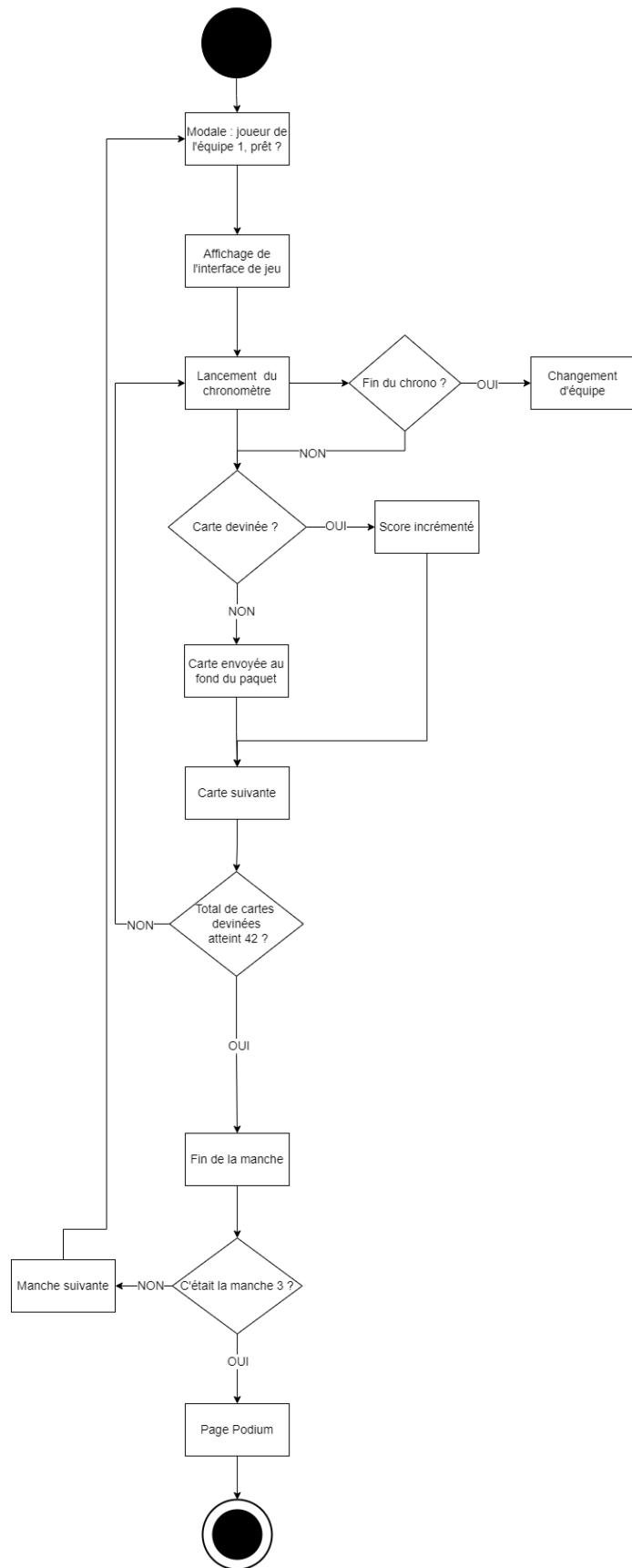


Diagramme de séquence

Données

1. Données en entrée :

Les données fournies à la page de jeu sont principalement liées à la configuration du jeu (options, thèmes) et aux interactions des utilisateurs pendant le déroulement de la partie.

Description :

- Les options de jeu sélectionnées (nombre d'équipes, noms des équipes, durée du chronomètre, etc.) sont récupérées avant le début de la partie.
- Les cartes à deviner, organisées par thèmes, sont sélectionnées pour le jeu à partir de la base de données.

Détail des données :

- Name : gameId
 - Type : Alphanumérique (MongoDB ObjectId)
 - Length : 24 caractères
 - Semantic : Identifiant unique du jeu.
 - Référence values list : Les IDs valides des jeux dans la base MongoDB.
- Name : options
 - Type : Objet contenant les options sélectionnées.
 - Structure :
 - nbTeams : Nombre d'équipes (nombre)
 - nameTeams : Tableau des noms des équipes (array de strings)
 - teamMembers : Nombre de joueurs par équipe (nombre)
 - chrono : Durée du chronomètre (nombre)
 - Value limits : Chaque option doit correspondre aux valeurs sélectionnées par l'utilisateur.
- Name : themes
 - Type : Tableau d'IDs des thèmes (array of alphanumeric)
 - Length : Jusqu'à 21 thèmes sélectionnés
 - Semantic : Représente les thèmes choisis pour la partie.
 - Référence values list : Liste des IDs des thèmes disponibles dans la base MongoDB.

2. Données gérées (traitement) :



Le traitement des données dans cette page de jeu inclut plusieurs actions, telles que la gestion des cartes à deviner, l'incrémentation des scores, et le changement de joueurs entre les équipes.

Description :

- Les cartes sont traitées et affichées une par une, permettant aux joueurs de deviner les éléments.
- Le système gère le chronomètre pour chaque équipe.
- Les scores sont mis à jour en temps réel pour chaque équipe.
- Les transitions entre les manches sont effectuées automatiquement une fois que toutes les cartes ont été jouées dans une manche.

Détail des données :

- Name : currentCard
 - Type : Objet représentant la carte actuelle à deviner (object)
 - Semantic : La carte affichée que les joueurs doivent deviner.
 - Structure :
 - theme_id : ID du thème auquel la carte appartient
 - element : Le mot ou concept à deviner (string)
 - Value limits : Chaque carte appartient à un des thèmes sélectionnés et est choisie aléatoirement dans la liste.
- Name : points
 - Type : Tableau contenant les scores par équipe (array of objects)
 - Structure :
 - teamName : Nom de l'équipe (string)
 - scoreRound1, scoreRound2, scoreRound3 : Scores obtenus dans chaque manche (nombre)
 - Semantic : Le système maintient et met à jour les scores au fur et à mesure que les cartes sont devinées.
 - Value limits : Les scores sont incrémentés pour chaque carte correctement devinée.

3. Données générées (en sortie) :

Les données générées sont principalement liées aux résultats finaux de la partie, les scores des équipes, et l'avancement dans les manches.

Description :



- Les scores finaux sont calculés et affichés à la fin de chaque manche et à la fin de la partie.
- Le podium avec les scores finaux est affiché une fois que toutes les manches sont jouées.

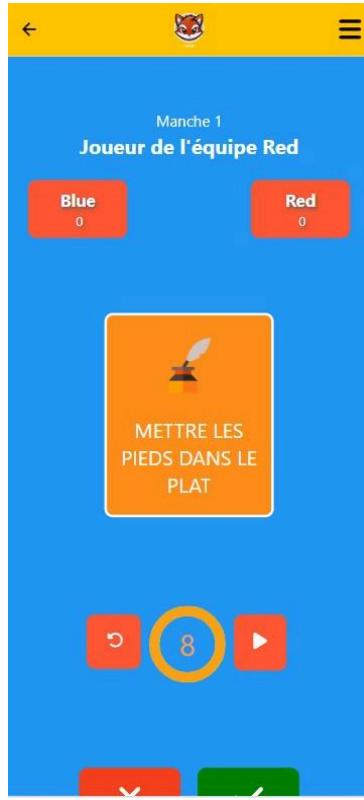
Détail des données :

- Name : finalScores
 - Type : Tableau des scores finaux par équipe (array of objects)
 - Structure :
 - teamName : Nom de l'équipe (string)
 - totalScore : Score total (cumul des scores des 3 manches) (nombre)
 - Semantic : Contient les scores finaux de chaque équipe après les trois manches.
 - Value limits : Score total pour chaque équipe.
- Name : winnerTeam
 - Type : Objet contenant l'équipe gagnante (object)
 - Structure :
 - teamName : Nom de l'équipe gagnante (string)
 - totalScore : Score total de l'équipe (nombre)
 - Semantic : Représente l'équipe qui a obtenu le plus haut score dans la partie.
 - Value limits : Une seule équipe peut avoir le score le plus élevé.

Résumé :

- Entrée : Les options de jeu sélectionnées (gameId, options, themes).
- Traitement : Gestion des cartes, du chronomètre, et des scores pendant la partie.
- Sortie : Scores par équipe, podium final, équipe gagnante.

Ecran/Affichage



2.1.1. Aide en ligne

Le menu hamburger présent sur toutes les pages donnera accès à une page de réglages qui contient un mode d'emploi de l'application et un formulaire de contact du support. Fonctionnalité réalisée ultérieurement.

2.1.2. Gestion des langues

Non concerné. L'application est uniquement en français.

2.1.3. Accessibilité

En raison de la nature même du jeu, il est primordial que les cartes soient facilement lisibles et que les joueurs puissent mimer les éléments à deviner. Cela rend malheureusement l'application peu adaptée aux personnes malvoyantes ou malentendantes.

Toutefois, nous avons conçu une charte graphique avec des couleurs vives et bien contrastées, des polices de grande taille et des boutons suffisamment larges. Ainsi, l'application reste accessible et

intuitive pour tous les types d'utilisateurs, y compris ceux qui sont moins à l'aise avec la technologie, tels que les jeunes enfants ou les grands-parents, afin de garantir une expérience familiale inclusive.

Utilisable dans des conditions de luminosité faible, ce qui est primordial pour un jeu de soirée.

6. SPECIFICATIONS TECHNIQUES

6.1. Référencement

1. Optimisation du titre et du sous-titre :

- **Titre** : Il est **court, clair, et mémorable**, tout en incluant des mots-clés pertinents liés à l'application. "DASH - Jeu de devinettes en équipe".
- **Sous-titre** : Il précise des aspects importants du jeu en utilisant des **mots-clés** qui peuvent attirer des utilisateurs potentiels, comme "jeu d'ambiance", "mimes".

2. Mots-clés optimisés :

- Sur le **Google Play Store**, les mots-clés sont extraits directement de la **description**, il faut donc inclure les mots-clés principaux de manière naturelle dans cette dernière.

3. Description optimisée :

- **Description longue** : Google Play utilise l'intégralité de la description pour améliorer le référencement. Il est essentiel d'utiliser les mots-clés principaux (ex. : "jeu de devinettes", "multijoueur", "en famille") dans les premières lignes de la description, car les utilisateurs voient d'abord ces lignes.
- **Limite de répétition** : Bien que les mots-clés doivent être présents, il est recommandé de ne pas les répéter de manière excessive, car cela peut être pénalisé.

4. Icône et screenshots :

- **Icône** : Elle est visuellement **attrayante et reconnaissable**. L'icône de **DASH**, qui représente un renard, doit être claire et se démarquer des autres applications du même genre. Elle peut inclure des couleurs vives pour attirer l'œil.

- **Screenshots** : Ils montrent les moments les plus importants du jeu : l'écran de sélection des équipes, les cartes à deviner, les scores, etc. Nous utilisons des descriptions sur les screenshots pour expliquer les fonctionnalités principales.

5. Vidéo promotionnelle :

- **Google Play Store** permet d'ajouter une vidéo. Une vidéo promotionnelle de 30 à 60 secondes qui montre le gameplay, les interactions entre les équipes, et la simplicité d'utilisation, sera créée.

6. Avis et notation :

- Les **avis des utilisateurs** sont cruciaux pour le référencement. L'application doit encourager les utilisateurs à laisser des **avis positifs**. Plus il y a d'avis positifs, plus l'application sera bien classée.
- **Répondre aux avis** : Répondre activement aux avis, qu'ils soient positifs ou négatifs, améliore également le classement.

7. Mises à jour régulières :

- Les **mises à jour fréquentes** sont vues de manière positive par les algorithmes des stores. Cela montre que l'application est maintenue, et elle est alors plus susceptible d'apparaître dans les résultats de recherche.

8. Performances techniques :

- **Temps de chargement rapide** : Assurer un temps de chargement rapide et des transitions fluides dans l'application.
- **Stabilité** : Moins il y a de bugs, moins les utilisateurs désinstallent l'application, ce qui est un facteur pris en compte dans l'ASO.

9. Taille de l'application :

- L'application **ne doit pas être trop lourde**. Les utilisateurs sont plus enclins à télécharger une application légère, et les stores favorisent également les apps plus légères dans les classements.

6.2. Environnement technique

Le front

L'application est développée en **React Native**. Nous avons choisi la version 0.74.5, qui n'est pas la dernière version (0.75) mais qui est suffisamment récente pour bénéficier des améliorations de performance et des corrections de bugs apportées par les versions précédentes. Cette version est également bien documentée, avec une communauté active qui a produit des ressources utiles. Nous n'avons pas jugé utile d'upgrade notre version lorsque la 0.75 est sortie car c'est une mise à jour assez mineure et nous comptons faire cet upgrade pour la version 0.76 qui devrait apporter des changements plus significatifs.

Nous utilisons React Native avec le framework **Expo**. Meta, la maison-mère qui a créé React Native préconise l'utilisation d'Expo pour la création d'applications.

Expo est un framework qui permet de démarrer rapidement avec React Native sans avoir à gérer la configuration native (iOS et Android). Il simplifie l'utilisation des fonctionnalités courantes comme la gestion des assets, les animations, et les autorisations natives, en fournissant des API prêtes à l'emploi.

Expo, et plus précisément la version 51.0.34, est un bon choix pour les projets qui cherchent à bénéficier des dernières fonctionnalités de React Native tout en maintenant une base de code facile à gérer et performante. Cela permet d'optimiser les cycles de développement et de rester compatible avec les futures évolutions du framework. La version 52 devrait sortir en même temps que la version 0.76 de React Native et nous ferons un upgrade de nos configurations à ce moment.

Le back

Nous utilisons la version 20.10.0 de **Node.JS**. Node.js 20 est une version "Current" qui sera probablement convertie en LTS (Long-Term Support) dans le futur, ce qui en fait un bon choix pour la pérennité et la stabilité de notre projet. Une version LTS est maintenue avec des correctifs de bugs critiques et de sécurité pendant une période prolongée, garantissant que nous n'aurons pas besoin de migrer trop souvent.

Cette version offre aussi de meilleures performances et une sécurité accrue.

Nous utilisons le framework **Express** dans sa version 4.18.2.

Express est un framework minimaliste facile à prendre en main qui est immensément populaire. Express repose sur un système de middlewares qui permet d'ajouter facilement des fonctionnalités comme la gestion des requêtes HTTP, les sessions, l'authentification, la gestion des erreurs, etc. Grâce à cette extensibilité, nous pouvons rapidement personnaliser et enrichir notre serveur en fonction des exigences de notre application. C'est particulièrement utile pour la sécurité.

Pour la base de données, nous nous sommes tournés vers **MongoDB** et **Mongoose** (la bibliothèque ODM qui nous sert d'interface entre MongoDB et Node.JS).

MongoDB est conçu pour l'évolutivité horizontale. Si notre application grandit et doit gérer de plus grandes quantités de données ou un plus grand nombre d'utilisateurs, on peut facilement distribuer les données sur plusieurs serveurs, ce qui permet à **MongoDB** de gérer de grandes charges avec des performances fiables.

La combinaison de **MongoDB** et **Mongoose** permet de bénéficier à la fois de la flexibilité et de l'évolutivité d'une base de données NoSQL, tout en ajoutant la structure et la rigueur nécessaires à travers **Mongoose** pour organiser et valider les données. En utilisant ces deux outils ensemble, nous obtenons un environnement puissant pour développer des applications modernes et scalables, tout en restant dans l'écosystème JavaScript/Node.js.

Librairies utilisées :

Back :

JWT (Json Web Token) : Permet de créer des jetons sécurisés pour l'authentification et l'autorisation des utilisateurs.

JOI : Une bibliothèque de validation de données pour s'assurer que les objets et les requêtes respectent un schéma défini.

bcrypt : Utilisée pour le hachage des mots de passe, offrant un moyen sécurisé de stocker des informations sensibles.

Mongoose : Crée une connexion orientée objet entre MongoDB et Node.js, facilitant les opérations avec MongoDB.

Cors : Un middleware qui gère les permissions des requêtes inter-origines pour sécuriser les échanges entre différents domaines.

Helmet : Un middleware qui sécurise les applications Express en définissant divers en-têtes HTTP.

Front :

Dotenv : Charge les variables d'environnement à partir d'un fichier `.env` pour une gestion centralisée des configurations sensibles.



react-native-deck-swiper : Permet de glisser des cartes dans une interface utilisateur, typiquement pour valider ou passer des éléments.

react-native-countdown-circle-timer : Un composant de minuteur circulaire animé avec des couleurs et des animations de progression.

react-native-async-storage : Un système de stockage clé-valeur asynchrone, non chiffré et persistant pour React Native.

expo-av : Un module universel pour gérer la lecture audio et vidéo dans les applications Expo.

react-native-fontawesome : Fournit des icônes FontAwesome pour les interfaces React Native.

react-navigation : Gère la navigation entre les différentes pages de l'application React Native.

6.3.Navigation et accessibilité

Méthodes d'accès : L'application sera téléchargeable via le **Google Play Store**.

Hébergement : Les données utilisateurs (scores, comptes) sont stockées sur une infrastructure **AWS**.

Email de support : teamdash2023@gmail.com

Conception responsive : L'application est optimisée pour **smartphones** et **tablettes** avec des écrans de différentes tailles.

6.4.Services tiers

Google Analytics : L'application intègre Google Analytics pour suivre les statistiques d'utilisation, les événements en jeu, et les données démographiques des utilisateurs. Cela permet d'améliorer l'expérience utilisateur en analysant les interactions et en optimisant les fonctionnalités.

Intégration réseaux sociaux : Les utilisateurs pourront à l'avenir partager leurs résultats sur les réseaux sociaux tels que Facebook et Instagram, augmentant ainsi l'engagement social et la viralité de l'application.

Emailing : Un service d'emailing est utilisé pour la récupération de compte en cas de perte de mot de passe.

6.5.Sécurité

Nous utilisons des frameworks, **Expo** pour le front et **Express** pour le back qui ajoutent déjà leur couche de sécurité supplémentaire à celle déjà présente dans React Native et Node.JS. **Express** limite les risques liés à l'exécution de code malveillant en interdisant certaines pratiques dangereuses. Par exemple, lorsqu'il traite des requêtes POST avec des payloads JSON, **Express** parse automatiquement les données, réduisant le risque d'exécution de code dangereux injecté dans les requêtes HTTP.

Expo intègre des protections pour la sécurité mobile, comme le chiffrement des données sensibles et gère les mises à jour sécurisées via OTA (over-the-air), réduisant les risques liés aux versions obsolètes.

Sécurité :

- La sécurité de l'application Node.js est renforcée via **Helmet**, qui ajoute des protections supplémentaires contre les attaques courantes telles que XSS et les attaques de type clickjacking.
- **CORS** est configuré pour restreindre les accès provenant de domaines non autorisés, garantissant un contrôle strict sur les sources externes pouvant accéder aux données.

Sauvegarde :

- Les données de l'application sont sauvegardées automatiquement sur une base de données **MongoDB**, avec des sauvegardes régulières pour garantir la récupération en cas de défaillance système.
- Les sauvegardes sont hébergées sur des serveurs sécurisés (**MongoDB Atlas**), garantissant la disponibilité des données.

Stockage :

- Les données utilisateur (scores, comptes, configurations) sont stockées dans une base de données **MongoDB**, avec des mots de passe protégés par un système de hachage sécurisé (bcrypt).
- Les fichiers statiques et ressources médias (si applicable) sont hébergés via des services cloud **AWS S3**, garantissant une gestion scalable et sécurisée.

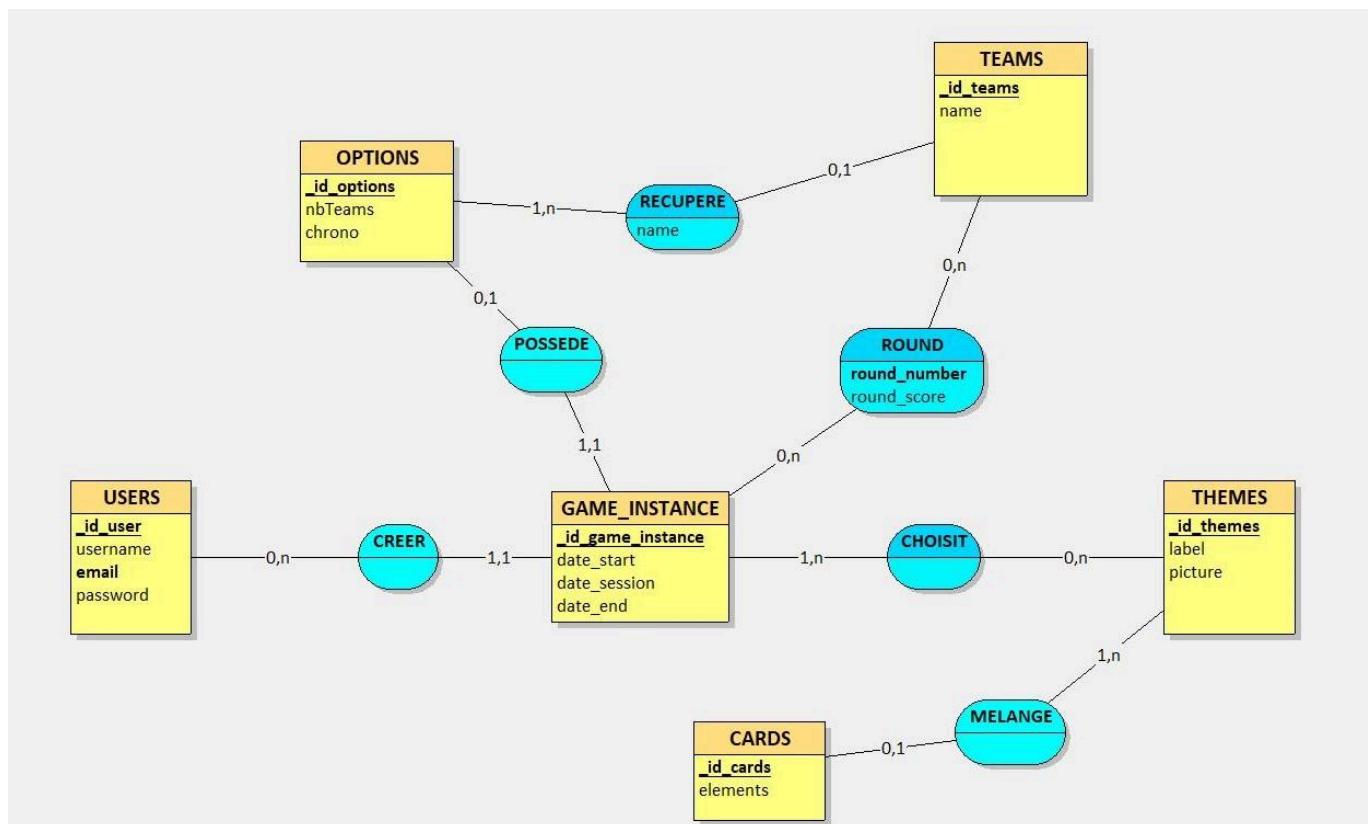
Versioning :

- Le code source est versionné via **GitHub**. Un dépôt Git est utilisé pour assurer le suivi des changements et permettre la collaboration en équipe via des branches pour chaque nouvelle fonctionnalité.

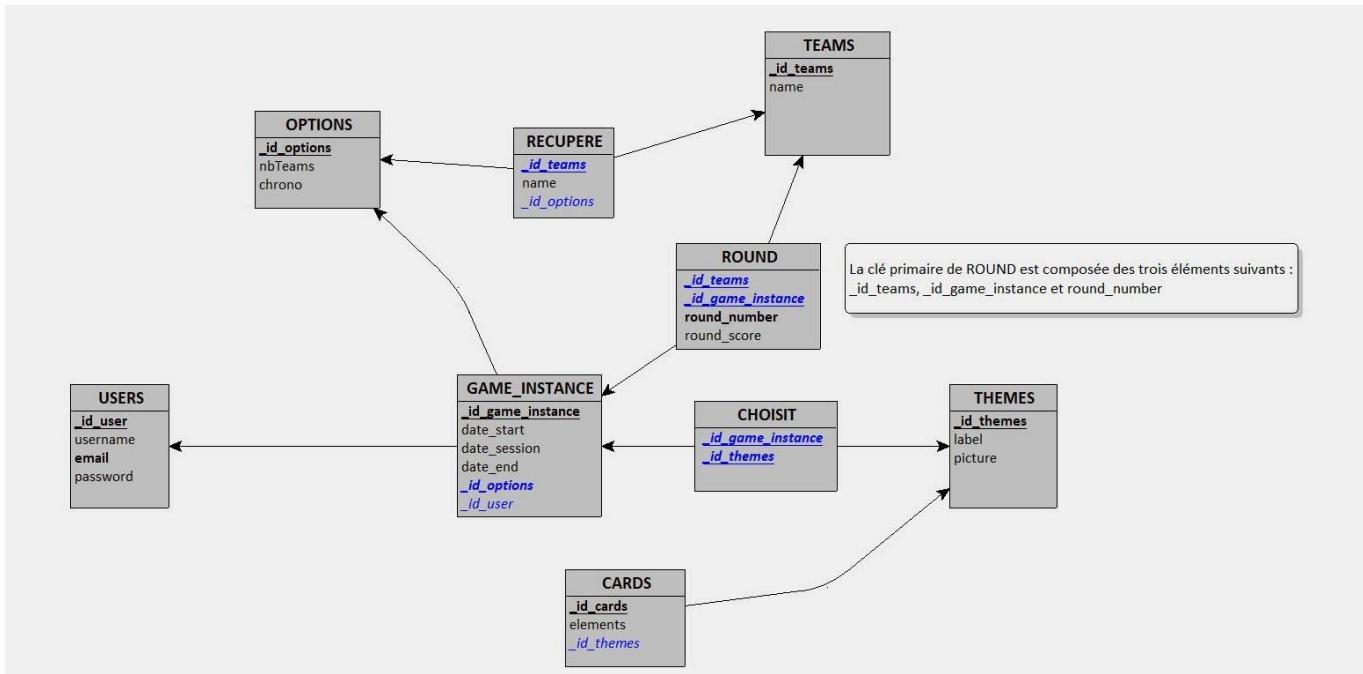
- Des révisions de code et des tests automatisés seront mis en place avant chaque fusion dans la branche principale pour assurer la stabilité des versions.

6.6. Gestion du contenu, des données

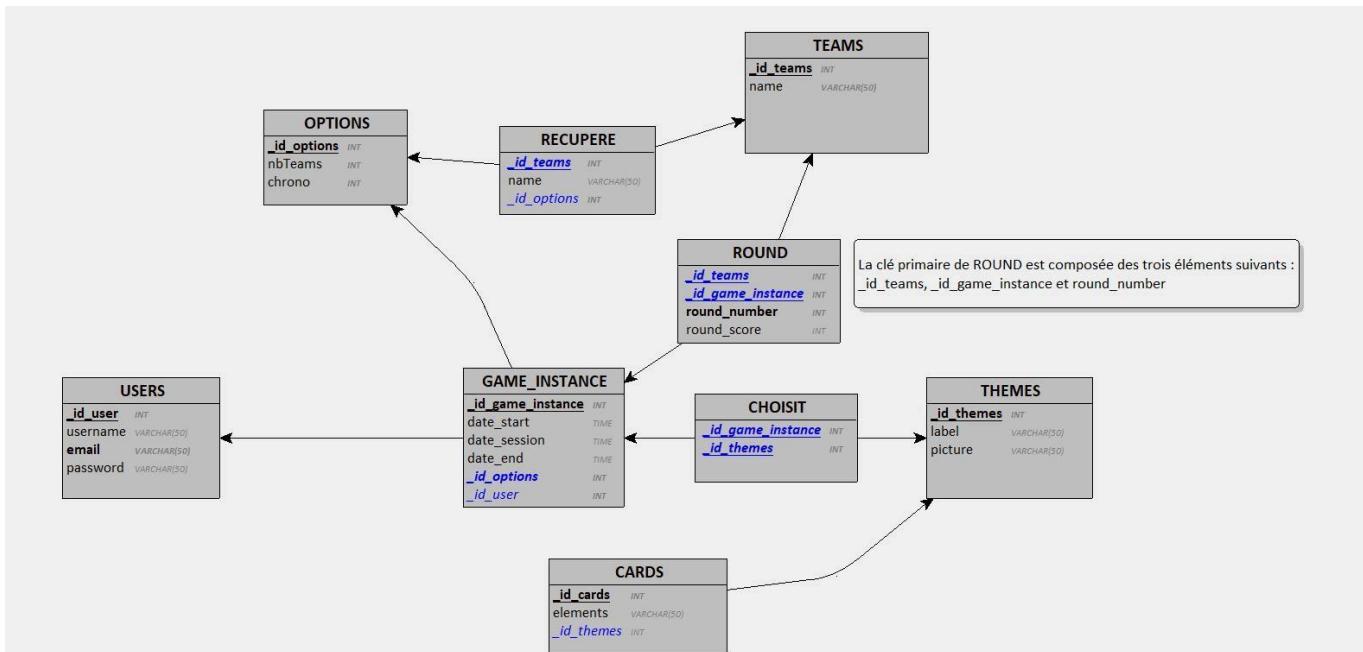
6.6.1. Modèle Conceptuel de Données (MERISE)



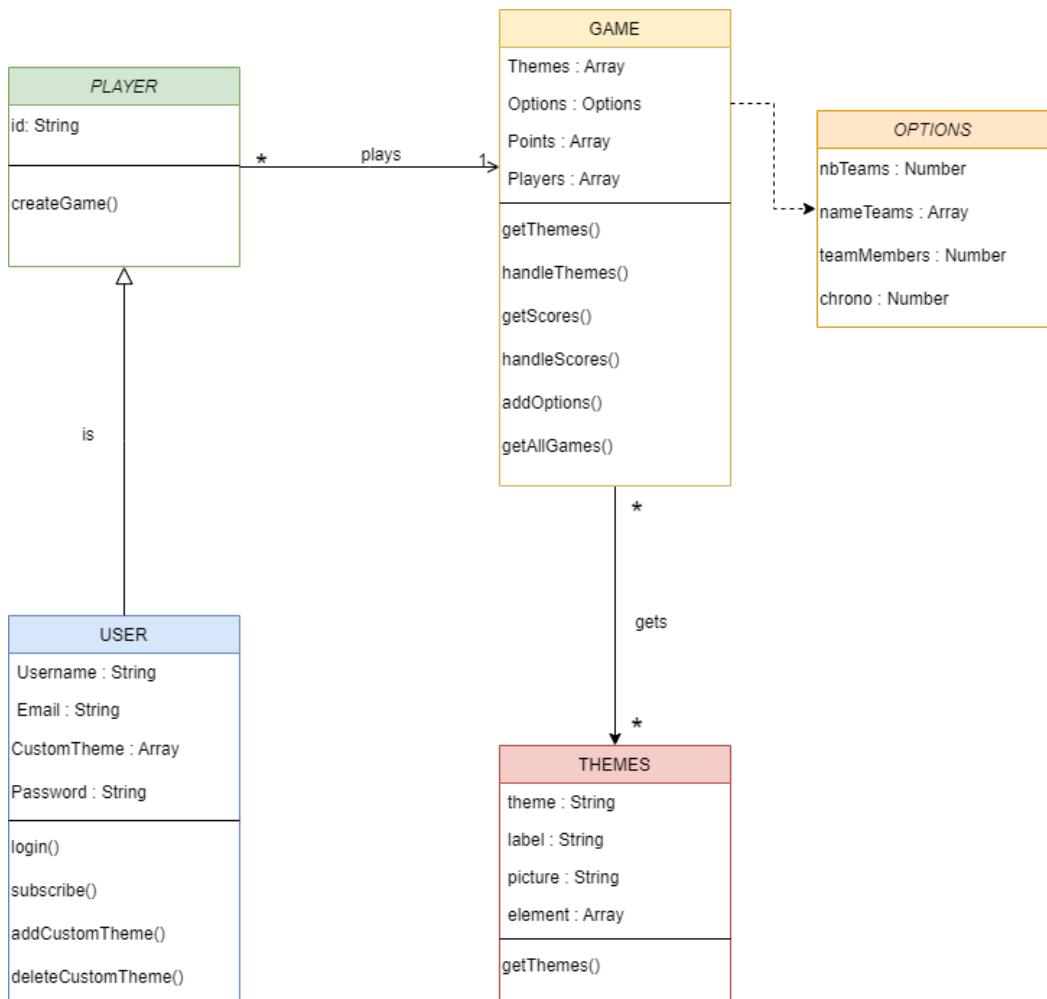
6.6.2. Modèle Logique de Données (MERISE)



6.6.3. Modèle Physique de Données (MERISE)



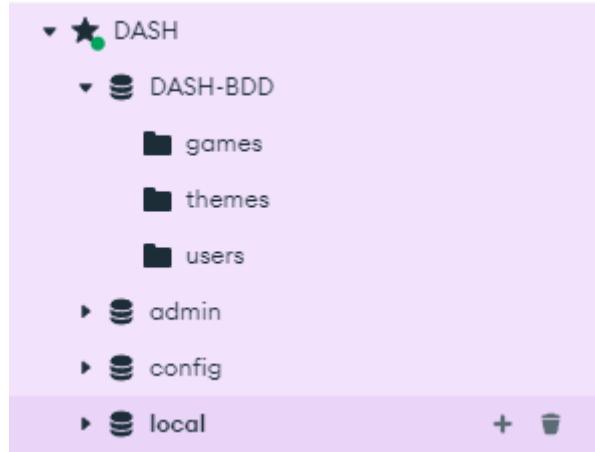
6.6.4. Diagramme de classes (UML)



6.6.5. Structure JSON de notre base de données MongoDB

Notre application utilise MongoDB, une base de données non SQL, notre modèle physique de données sera donc plus souple et moins formel qu'un modèle physique de données basé sur du SQL.

Notre base de données est composée de 3 collections. Voici leur structure.



- **Collection Games** : un document Games est créé à chaque nouvelle partie de jeu.

```
Unset
{
  "_id": {
    "$oid": "66fba7f502c93cedd8d76eac"
  },
  "themes": [],
  "options": {
    "nbTeams": [
      "nameTeams": [],
      "teamMembers": [
        "chrono": []
      ],
      "players": [],
      "points": []
    ],
    "__v": 0
  }
}
```

- **Collection Themes** : contient autant de documents que de thèmes disponibles dans notre application (20 à l'heure actuelle).

```
Unset
{
  "_id": {
```

```
    "$oid": "660aa7579e9dc3bcfd81b313"  
},  
"id": 4,  
"themes": "sports",  
"label": "Sports",  
"picture": "sports.png",  
"element": [  
    //Liste des 80 éléments du thème  
],
```

- **Collection Users** : un document est créé à chaque inscription d'un nouvel utilisateur.

```
Unset  
{  
    "_id": {  
        "$oid": "6616a2fe11d73513f99588ab"  
    },  
    "username": "M       e",  
    "email": "m       e@m       e.fr",  
    "password": "$2a$10$oobJ0xxEZXgCx3YnrwZv100T05JLghb7gp1U.p0hV7yyAosAqZpNu",  
    "__v": 0  
}
```

7. RÉALISATIONS

7.1 Exemple 1 : Validation des entrées du formulaire d'inscription côté front avec affichage des erreurs à l'utilisateur selon les cas échéants.

7.1.1 Extrait de code (voir code complet en annexe, p 105)

```
const handleSignup = async () => {
  if (emailError || passwordError || confirmPasswordError) {
    return;
  }

  try {
    const response = await fetch(`#${apiUrl}/signup`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        username,
        email,
        password,
        confirmPassword
      }),
    });

    const data = await response.json();

    if (response.status === 200) {
      await saveToken(data.token);
      navigation.navigate('UserPage');
    } else if (data.message && data.message.includes('duplicate key')) {
      setEmailError("Cet e-mail est déjà utilisé. Veuillez en choisir un autre.");
    } else {
      console.log(data.message);
    }
  } catch (error) {
    console.error("Erreur lors de l'inscription:", error);
  }
};
```

7.1.2 Argumentation

Inscription ≡

Bob

bob

Veuillez entrer une adresse e-mail valide.

J'accepte les Conditions et la Politique de confidentialité.

S'inscrire

Inscription ≡

Bob

m.thomas1403@gmail.com

Cet e-mail est déjà utilisé. Veuillez en choisir un autre.

J'accepte les Conditions et la Politique de confidentialité.

S'inscrire

Inscription ≡

Bob

m.thomas1403@gmail.com

Le mot de passe doit contenir au moins 8 caractères, incluant des majuscules, minuscules, chiffres et caractères spéciaux.

J'accepte les Conditions et la Politique de confidentialité.

S'inscrire

Inscription ≡

Bob

bob@bob.fr

Les mots de passe ne correspondent pas.

J'accepte les Conditions d'Utilisation et la Politique de confidentialité.

S'inscrire

Ce code pour la page d'inscription dans une application mobile présente plusieurs avantages pour l'expérience utilisateur (UX) en se concentrant sur la fluidité, la clarté et la réactivité du formulaire d'inscription.

1. Validation en temps réel des erreurs :

En affichant des messages d'erreur directement sous les champs (comme pour l'email, le mot de passe, ou la confirmation du mot de passe), l'utilisateur reçoit des retours instantanés sans avoir à attendre de soumettre le formulaire. Cela évite la frustration de remplir tout un formulaire pour ensuite voir apparaître une erreur. C'est une bonne pratique d'accessibilité.

2. Gestion de la force du mot de passe :

Le code implémente une vérification de la force du mot de passe avec des critères comme la longueur, l'utilisation de caractères spéciaux, majuscules, minuscules et chiffres. Cela encourage les utilisateurs à créer des mots de passe plus robustes, renforçant ainsi la sécurité globale de l'application.

L'utilisateur est informé de manière précise des exigences de sécurité pour son mot de passe, renforçant la transparence et évitant des essais infructueux.

3. Personnalisation des erreurs :

Plutôt que d'afficher une alerte générique (comme un pop-up), chaque champ affiche un message spécifique lié à son erreur. Cela simplifie la compréhension et rend l'interface plus intuitive.

Les messages d'erreur comme "Cet e-mail est déjà utilisé" ou "Les mots de passe ne correspondent pas" permettent aux utilisateurs de corriger les problèmes sans confusion.

4. Gestion des conditions d'utilisation :

Le fait d'inclure une case à cocher pour accepter les conditions d'utilisation et la politique de confidentialité renforce la conformité légale tout en étant clair pour l'utilisateur. Cela responsabilise l'utilisateur tout en améliorant la transparence de l'application.

5. Prévention des inscriptions incomplètes :

Bouton désactivé jusqu'à acceptation des termes : Tant que l'utilisateur n'accepte pas les conditions, le bouton d'inscription est grisé, ce qui empêche une tentative d'inscription prémature ou incomplète. Cela guide l'utilisateur étape par étape et évite des frustrations inutiles.

7.2 Exemple 2 : la validation des champs côté back et la sécurisation des données.

7.2.1 Extraits de code

```
validateSignupData(data) {
  const schema = Joi.object({
    username: Joi.string().min(3).max(30).required().messages({
      'string.empty': 'Le nom d\'utilisateur ne peut pas être vide.',
      'string.min': 'Le nom d\'utilisateur doit contenir au moins 3 caractères.',
      'string.max': 'Le nom d\'utilisateur ne peut pas dépasser 30 caractères.'
    }),
    email: Joi.string().email().required().messages({
      'string.email': 'L\'adresse email doit être valide.',
      'string.empty': 'L\'email ne peut pas être vide.'
    }),
    password: Joi.string().pattern(new RegExp('^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,$}')).required().messages({
      'string.pattern.base': 'Le mot de passe doit contenir au moins 8 caractères, incluant des majuscules, minuscules, chiffres et caractères spéciaux.',
      'string.empty': 'Le mot de passe ne peut pas être vide.'
    }),
    confirmPassword: Joi.any().valid(Joi.ref('password')).required().messages({
      'any.only': 'Les mots de passe doivent correspondre.'
    })
  });

  return schema.validate(data, { abortEarly: false });
}


```

```
async signup(req, res) {
  const { username, email, password } = req.body;
  // Valider les données de la requête
  const { error } = this.validateSignupData(req.body);
  if (error) {
    const errorMessages = error.details.map(detail => detail.message);
    return res.status(400).json({ errors: errorMessages });
  }

  try {
    const passwordSalt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, passwordSalt);

    const newUser = new UserModel({
      username,
      email,
      password: hashedPassword
    });
    const user = await newUser.save();
    const token = this.jwtHandler.buildJwt(user);

    return res.json({
      message: `Bienvenue ${user.username}`,
      token: token,
      userId: user._id
    });
  } catch (error) {
    return res.status(500).json({ message: error.toString() });
  }
}
```



7.2.2 Argumentation

1. Validation stricte des données avec Joi

La fonction `validateSignupData` utilise Joi, une librairie de validation de données, pour assurer que les informations fournies par l'utilisateur respectent certaines règles avant d'être traitées par le serveur.

- Nom d'utilisateur : Doit contenir entre 3 et 30 caractères. Cela évite les noms d'utilisateur trop courts (inutiles) ou excessivement longs (qui pourraient poser des problèmes d'interface ou de stockage).
- Email : Vérifie que l'adresse e-mail est valide via un format standard. Cela évite d'accepter des chaînes de caractères qui ne respectent pas le format d'email, minimisant les risques de recevoir des informations incorrectes.
- Mot de passe : La politique de mot de passe impose l'usage d'une combinaison complexe : majuscules, minuscules, chiffres, et caractères spéciaux. Cela garantit que l'utilisateur crée un mot de passe sécurisé, ce qui rend les attaques de force brute ou de devinettes plus difficiles. Cette validation côté serveur est importante même si elle a déjà été faite côté client, car les données provenant du client peuvent être manipulées.
- Confirmation du mot de passe : Assure que les deux mots de passe correspondent, minimisant les erreurs utilisateur lors de l'inscription.

Cette validation protège contre les tentatives d'injection de données non sécurisées ou mal formatées, en amont du traitement. En renvoyant des messages d'erreurs explicites (comme les mots de passe non conformes ou les emails mal formés), elle améliore également l'expérience utilisateur tout en maintenant un haut niveau de sécurité.

2. Hachage sécurisé du mot de passe avec bcrypt

La partie suivante du code, dans la méthode `signup`, concerne le traitement sécurisé du mot de passe :

`bcrypt` est utilisé pour hacher le mot de passe avant de le stocker dans la base de données. Cette pratique est essentielle pour la sécurité, car cela évite de stocker des mots de passe en texte clair, réduisant les conséquences potentielles en cas de violation de la base de données. De plus, `bcrypt` inclut un mécanisme de salage (salt), qui rend chaque hachage unique, même pour des mots de passe identiques, rendant les attaques de "rainbow tables" inefficaces..

3. Gestion des erreurs

En cas d'erreurs dans la validation des données, le code renvoie des messages d'erreurs explicites au client. Cela permet à l'utilisateur de comprendre pourquoi ses données ont été rejetées (par exemple, des mots de passe non conformes ou des emails invalides).

C'est une pratique clé pour garantir à la fois la sécurité (en s'assurant que seules les données valides sont traitées) et la transparence pour l'utilisateur. Cela limite aussi les chances de recevoir des données corrompues ou mal formatées dans la base de données.

4. Génération de token JWT pour l'authentification

Après l'inscription, un token JWT (JSON Web Token) est généré pour l'utilisateur avec `this.jwtHandler.buildJwt(user)`. Ce token permet d'authentifier l'utilisateur pour les prochaines requêtes sans avoir à re-soumettre ses identifiants (comme son mot de passe). Le token est sécurisé et contient les informations de l'utilisateur de manière cryptée.

Utiliser des tokens JWT pour gérer les sessions utilisateur est une pratique courante et sécurisée. Cela permet d'éviter de stocker des sessions côté serveur tout en assurant une authentification robuste.

Ce code met en place une solide validation des données côté serveur avec Joi, sécurise les mots de passe grâce à bcrypt, et utilise JWT pour l'authentification. La combinaison de ces pratiques assure que seules les données conformes sont traitées et stockées, que les mots de passe sont protégés en cas de fuite de données, et que les utilisateurs peuvent se connecter de manière sécurisée via des tokens.

7.3 Exemple 3 : Récupération, traitement et affichage des scores finaux pour le Podium de fin de partie.

7.3.1 Extraits de code

```

const getScores = async () => {
  try {
    const response = await fetch(`/${apiUrl}/games/${gameId}/points`);
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();

    const scoresWithTotal = data
      .map(team => ({
        ...team,
        total: (team.scoreRound1 || 0) + (team.scoreRound2 || 0) + (team.scoreRound3 || 0),
      }));
    .sort((a, b) => b.total - a.total);

    setScores(scoresWithTotal);
  } catch (error) {
    console.error('Error fetching game by id:', error);
    return null;
  }
};

```

Voici l'organisation d'une instance de jeu au sein de notre base de données.

```
{
  "_id": {
    "$oid": "66fba88902c93cedd8d76ed4"
  },
  "themes": [
    "660aa7579e9dc3bcfd81b326"
  ],
  "options": {
    "nbTeams": 3,
    "nameTeams": [
      "Jason",
      "Fayssal",
      "Marianne"
    ],
    "teamMembers": 2,
    "chrono": 30
  },
  "players": [
    "66695afca0087892f2468350"
  ],
  "points": [
    {
      "teamName": "Jason",
      "scoreRound1": 15,
      "scoreRound2": 16,
      "scoreRound3": 12,
      "_id": {
        "$oid": "66fba92302c93cedd8d77276"
      }
    },
    {
      "teamName": "Fayssal",
      "scoreRound1": 14,
      "scoreRound2": 12,
      "scoreRound3": 10,
      "_id": {
        "$oid": "66fba92302c93cedd8d77277"
      }
    },
    {
      "teamName": "Marianne",
      "scoreRound1": 13,
      "scoreRound2": 14,
      "scoreRound3": 19,
      "_id": {
        "$oid": "66fba92302c93cedd8d77278"
      }
    }
  ],
  "__v": 0
}
```

```

const renderScores = () => {
  if (scores && scores.length > 0) {
    let currentRank = 0;
    let previousTotal = null;

    return scores.map((team, index) => {
      let medalSource;
      if (previousTotal === team.total) {
      } else {
        currentRank = index + 1;
      }

      previousTotal = team.total;

      if (currentRank === 1) {
        medalSource = require('../img/gold.png');
      } else if (currentRank === 2) {
        medalSource = require('../img/argent.png');
      } else if (currentRank === 3) {
        medalSource = require('../img/bronze.png');
      }

      return (
        <View key={index} style={styles.scoreCard}>
          {medalSource && <Image source={medalSource} style={styles.medalImage} />}
          <Text style={styles.teamName}>{team.teamName}</Text>
          <Text style={styles.score}>Total: {team.total}</Text>
        </View>
      );
    });
  }
  return <Text style={styles.noScores}>No scores available</Text>;
};


```

La fonction getScores fait une requête à l'API pour récupérer les scores depuis la base de données.

Les scores sont ensuite traités pour calculer un score total par équipe, par l'addition des scores de chaque round.

Les équipes sont triées en fonction de ce score total, par ordre décroissant, puis les résultats sont stockés dans l'état (via setScores).

Si une erreur survient à tout moment, elle est capturée et affichée.

Cette fonction est conçue pour traiter les scores de manière dynamique, assurant que les scores sont correctement calculés et ordonnés avant d'être affichés dans l'application.

Dans l'optique de l'examen du diplôme, examinons comment cette requête pourrait se traduire dans un environnement SQL.

Je veux récupérer pour une instance de jeu donnée et pour chaque équipe, les scores de chaque round, les additionner, puis les classer par ordre décroissant.

En SQL, ça donnerait :

```
Unset
SELECT
    team_name,
    score_round1,
    score_round2,
    score_round3,
    (score_round1 + score_round2 + score_round3) AS total_score
FROM teams
WHERE game_id = 66fba88902c93cedd8d76ed4
ORDER BY total_score DESC;
```

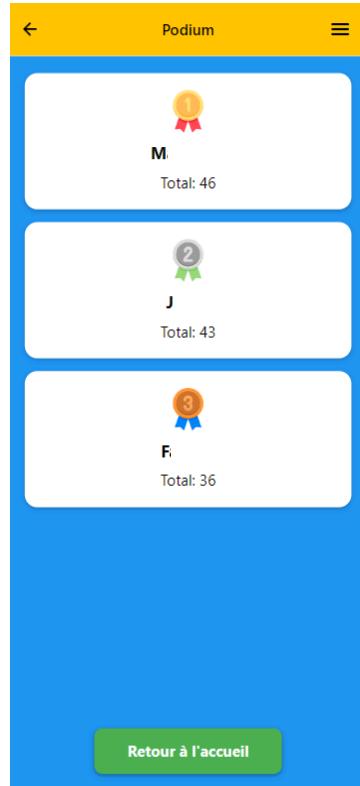
Puis la fonction renderScores génère une liste d'affichage des scores des équipes.

Elle vérifie en premier lieu s'il y a des scores disponibles.

Pour chaque équipe, elle attribue un rang (classement) basé sur le score total.

Elle assigne une médaille (or, argent, bronze) aux trois premières équipes, en fonction de leur rang. Elle gère également les égalités, si deux équipes ont le même score, en fonction de leur rang, une même médaille s'affichera.

Voici le résultat dans l'application :



8. PRÉSENTATION DU JEU D'ESSAI DE LA FONCTIONNALITÉ LA PLUS REPRÉSENTATIVE

8.1 Fonctionnalité testée

Nous testons le parcours de lancement d'une partie depuis la page d'accueil jusqu'au début proprement dit du jeu.

8.2 Description des scénarios

Jeu d'essai	Description	Résultat attendu Back-end Front-end
Connexion	L'utilisateur se connecte à l'application au moyen de son email et de son identifiant.	- Appel API pour vérifier que l'utilisateur existe dans la base de données.

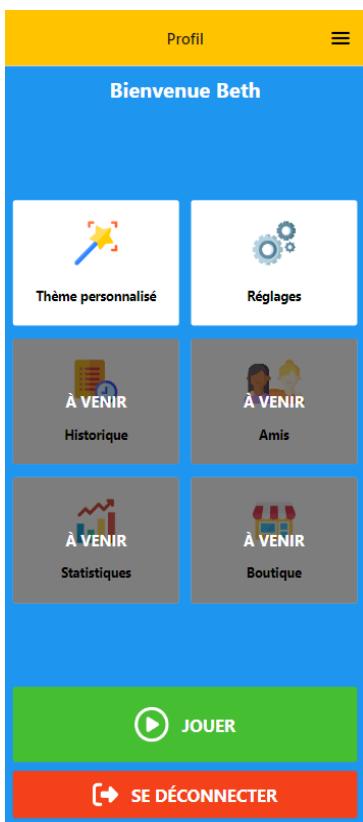
		<ul style="list-style-type: none"> - La connexion réussit, l'utilisateur est dirigé vers sa page Profil.
Jouer	L'utilisateur clique sur Jouer, depuis la page Profil.	<ul style="list-style-type: none"> - Un Objet game avec un identifiant unique se crée dans la base de données. - Le parcours de préparation du jeu commence, navigation vers la page Thèmes.
Thèmes	L'utilisateur choisit un ou plusieurs thèmes, puis les valide.	<ul style="list-style-type: none"> - Les thèmes sont persistés dans l'objet game de la base de données. - navigation vers la page Options.
Options	L'utilisateur sélectionne le nombre d'équipes, le nom des équipes, le nombre de joueurs par équipes, le chronomètre puis valide sa sélection.	<ul style="list-style-type: none"> - Les options sont persistées dans l'objet game de la base de données. - navigation vers la page Start. <p>Fin de la préparation du jeu.</p>

8.3 Résultats des tests

8.3.1 Jeu d'essai avec des données valides

Jeu d'essai	Résultat attendu	Résultat obtenu
Connexion	<p>L'utilisateur se connecte à l'application au moyen de son email et de son identifiant. La connexion réussit, il parvient sur sa page Profil.</p>	<pre>{ "_id": { "\$oid": "66695afca0087892f2468350" }, "username": "Beth", "email": "m.thomas1403@gmail.com", "password": "\$2a\$10\$mBRf.s..a3wahuyp3n9zbeBxg.quSr1mHVJGVxzJlnFfv4Hrhp7U2", "__v": 36, "customThemes": [] }</pre> <p>La requête API a cherché un utilisateur correspondant aux identifiants entrés et l'a trouvé dans la base de données.</p>

Jouer

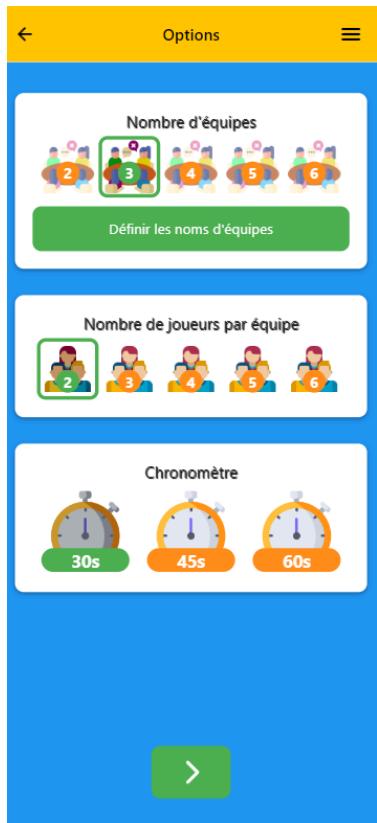


Clic sur Jouer

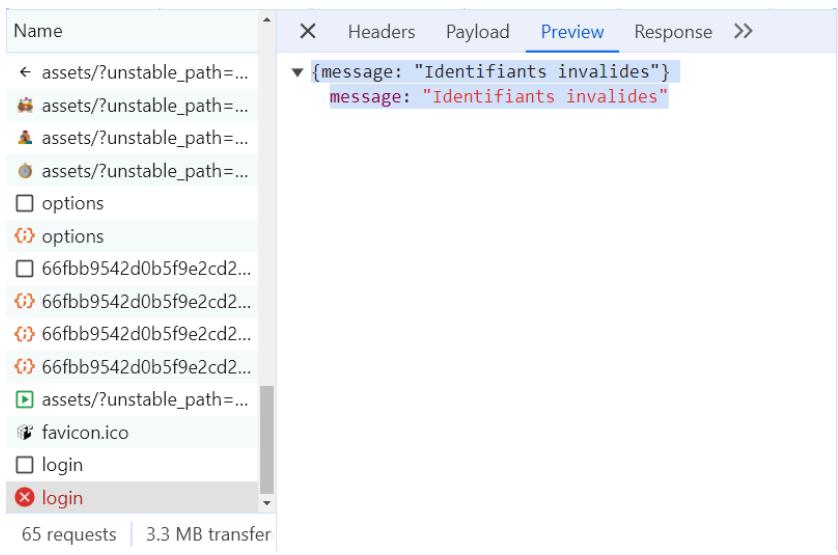
```
{  
  "_id": {  
    "$oid": "66fd51a9d29ade2fcebc9f5f"  
  },  
  "themes": [],  
  "options": {  
    "nameTeams": []  
  },  
  "players": [  
    "66695afca0087892f2468350"  
  ],  
  "points": [],  
  "__v": 0  
}
```

Un Objet Game est bien créé en base de données. Prêts à recevoir les données du jeu. Le joueur connecté est enregistré par son id.

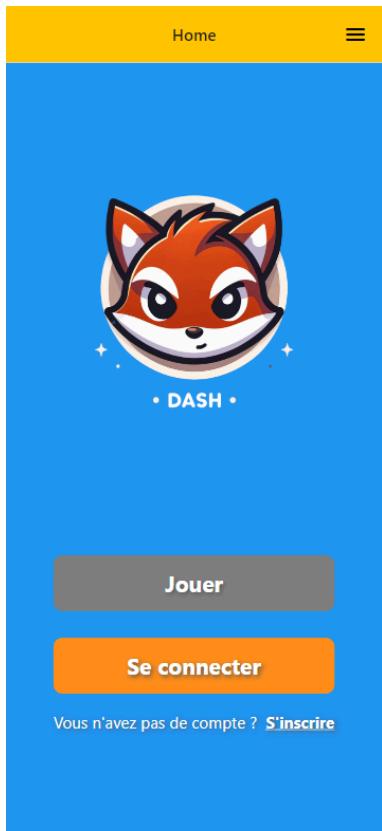
<p>Thèmes</p>	 <p>Sélection des thèmes puis validation. → Navigation vers Options.</p>	<pre>{ "_id": { "\$oid": "66fd51a9d29ade2fcebc9f5f" }, "themes": ["660aa7579e9dc3bcfd81b31a", "660aa7579e9dc3bcfd81b314"], "options": { "nameTeams": [] }, "players": ["66695afca0087892f2468350"], "points": [], "__v": 0 }</pre> <p>Les thèmes sont bien persistés dans l'objet game de la base de données, par leur id.</p>
---------------	---	--

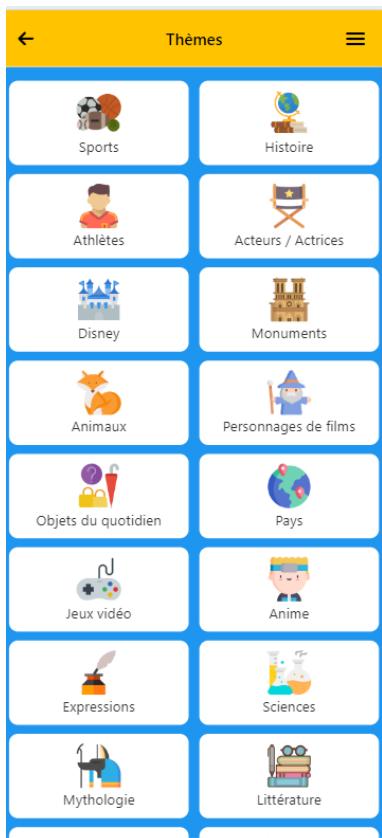
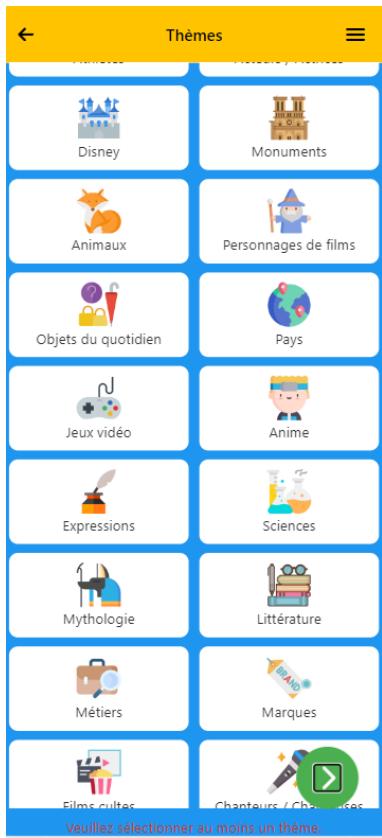
<p>Options</p>  <p>Sélection des options et validation. → Navigation vers la page Start.</p>	<pre>{ "_id": { "\$oid": "66fd51a9d29ade2fcebc9f5f" }, "themes": ["660aa7579e9dc3bcfd81b31a", "660aa7579e9dc3bcfd81b314"], "options": { "nbTeams": 3, "nameTeams": ["Fayssal", "Jason", "Marianne"], "teamMembers": 2, "chrono": 30 }, "players": ["66695afca0087892f2468350"], "points": [], "__v": 0 }</pre> <p>Les options sont bien persistées dans l'objet game sous la forme d'un objet options.</p>
---	--

8.3.2 Jeu d'essai avec des données invalides

Jeu d'essai	Résultat attendu	Résultat obtenu
Connexion	<p>Tentative de connexion avec des identifiants inconnus.</p> <p>La navigation vers la page Profil doit être impossible.</p> <p>L'utilisateur doit être informé du souci.</p>	 <p>Le front renvoie un message "identifiants incorrects". La navigation ne se fait pas.</p> 

		Le back bloque la requête et renvoie le message “identifiants invalides”.
--	--	---

Jouer	<p>Impossible de jouer en mode connecté mais possible de jouer en mode invité en cliquant sur le bouton gris JOUER</p> 	<pre>{ "_id": { "\$oid": "66fd52a7d29ade2fcebc9f78" }, "themes": [], "options": { "nameTeams": [] }, "players": [], "points": [], "__v": 0 }</pre> <p>Un Objet Game est également créé en base de données, mais il ne comporte pas d'id de joueur authentifié. La navigation peut se poursuivre vers la page Themes sans problème.</p>
-------	---	--

Thèmes	<p>Si je ne sélectionne pas de thème, aucun bouton pour passer à la suite ne s'affiche.</p> <p>Si je sélectionne puis désélectionne un thème, un message d'erreur s'affiche, me demandant de sélectionner au moins un thème.</p> <p>La navigation est bloquée et rien n'est envoyé en BDD tant qu'un thème au moins n'est pas sélectionné.</p>	 
--------	--	--

Options	<p>Si je ne sélectionne pas une option, un message d'erreur doit s'afficher sous chaque champ manquant pour informer l'utilisateur au mieux.</p>	 <p>Rien n'est envoyé en base de données, tant que les champs ne sont pas tous remplis.</p>

8.4 Conclusion

Suite aux tests, nous remarquons qu'il manque une fonctionnalité pour l'utilisateur qui aurait oublié ses identifiants : la possibilité de les récupérer par mail, ou même seulement un bouton de retour à l'accueil qui ne s'afficheraient qu'en cas d'échec de connexion.

Cette erreur sera corrigée à la prochaine mise à jour.

9. DESCRIPTION DE LA VEILLE SUR LES VULNÉRABILITÉS DE SÉCURITÉ

Dans un environnement technologique qui évolue vite, la sécurité est un élément fondamental. Je suis consciente des risques croissants liés aux cyberattaques, j'ai donc mis en place des méthodes d'apprentissage pour garantir que mes pratiques de développement restent solides et en phase avec les dernières menaces.

Comment je me renseigne ?

Pour rester informée sur les meilleures pratiques de sécurité, j'utilise plusieurs canaux et ressources fiables, notamment un agrégateur personnalisé (daily.dev en page d'accueil de mon navigateur) qui me permet de suivre les articles, nouvelles vulnérabilités et bonnes pratiques liés aux frameworks et technologies que j'utilise.

Je surveille en continu les mises à jour des frameworks que nous utilisons, comme React, Node.js, ou MongoDB. Ces mises à jour incluent souvent des correctifs de sécurité, et je les intègre dans nos projets dès que possible. Les changelogs et les forums associés sont également des sources importantes pour comprendre les enjeux de ces évolutions.

Je mets à profit les alertes Github également qui sont précieuses pour informer sur certains problèmes de sécurité souvent liés à des dépendances obsolètes.

À quelle fréquence je me renseigne ?

La fréquence de mes recherches et de ma veille est hebdomadaire, mais j'effectue également une surveillance active des mises à jour critiques.

Ce que je mets en œuvre dans ce domaine

Pour garantir une sécurité optimale, j'ai adopté plusieurs stratégies concrètes :

- J'utilise des algorithmes de hachage robustes comme **bcrypt** pour le stockage des mots de passe et du **SSL/TLS** pour sécuriser les communications entre les utilisateurs et le serveur.
- Nous allons mettre en place des procédures d'authentification strictes, telles que l'intégration d'OAuth 2.0 pour les connexions via Google, en plus de notre système d'authentification classique, afin de minimiser les risques liés à l'usurpation d'identité.



- L'utilisation de tokens **JWT** pour l'authentification.
- Nous appliquons des contrôles de validation stricts pour toutes les données entrées par les utilisateurs via des bibliothèques telles que **Joi**, ce qui permet de réduire les attaques par injection SQL, XSS ou autres formes d'exploitations.
- Nous déployons les mises à jour critiques dès qu'elles sont disponibles afin de protéger notre application contre les nouvelles vulnérabilités. Cela inclut les mises à jour des frameworks et des dépendances.

10. DESCRIPTION D'UNE SITUATION DE TRAVAIL AYANT NÉCESSITÉ UNE RECHERCHE SUR SITE ANGLOPHONE ou FRANCOPHONE

10.1 Description du besoin

Mon application envoie des données critiques, comme les scores des équipes par exemple, au serveur via des requêtes HTTP. Il est important que ces données soient valides avant de les enregistrer dans la base de données. Sans validation appropriée, des erreurs comme des champs manquants, des types incorrects (par exemple, des NaN au lieu de nombres) ou des valeurs invalides peuvent se retrouver dans la base de données, entraînant des bugs difficiles à traquer. On doit aussi pouvoir leur donner des contraintes : par exemple pour les mots de passe, qu'ils fassent 8 caractères au minimum, qu'ils respectent une regex en particulier etc.

La validation stricte des données est fastidieuse en Node.js, il doit exister une librairie dédiée à mon besoin.

10.2 Description de la recherche

10.2.1 Moteur de recherche

Duck duck go

10.2.2 Mots clés

library node js validation data

10.2.3 Résultat de la recherche (liste Sites)

library data validation node.js at DuckDuckGo Search - library

https://www.npmjs.com/package/validator
validator.js - npm
This library validates and sanitizes strings only. If you're not sure if your input is a string, coerce it using `input + ""`. Passing anything other than a string will result in an error. Installation and Usage. Server-side usage. Install the library with `npm install validator`. No ES6.

https://byby.dev/js-object-validation
Top 10 JavaScript Data Validation Libraries - byby.dev
2 juin 2024 · Data validation libraries are designed for validating data at runtime, so it throws or returns detailed runtime errors for you or your end users. This is especially useful in situations like accepting arbitrary input in a REST or GraphQL API. It can even be used to validate internal data...

https://express-validator.github.io/docs
express-validator | express-validator - GitHub Pages
This version of express-validator requires that your application is running on Node.js 14+. It's also verified to work with express.js 4.x. Note that, despite the name, express-validator might work with libraries that aren't express.js.

https://dev.to/bnevilleoneill/how-to-handle-data-validation-in-node-using-validatorjs-4if0
How to handle data validation in Node using validatorJS
The validator function which extends the library's validator constructor (as seen in `src/helpers/validate.js` above) accepts four arguments — data to be validated, the validation rule, custom error messages (if any), and a callback method.

https://toxigon.com/data-validation-in-nodejs
Data Validation in Node.js: A Comprehensive Guide
Learn how to implement data validation in your Node.js applications using popular libraries like Express-validator and Joi. Discover best practices for validating data, handling errors, and ensuring data integrity in your Node.js backend. Get insights and examples to build secure and reliable Node.j...

https://github.com/hapijs/joi
hapijs/joi: The most powerful data validation library for JS - GitHub
The most powerful data validation library for JS. Contribute to hapijs/joi development by creating an account on GitHub.

10.2.4 Critères de sélection et choix du(es) site(s)

J'élimine d'emblée les sites du type "Top 10". Ils sont basés sur un instant donné, sont peut-être obsolètes ou sponsorisés.

Il me reste trois choix de librairies fiables : express-validator, validatorJS et Joi.

Mon choix s'est porté sur Joi uniquement pour la qualité de leur [documentation](#) que j'ai trouvé très bien faite et limpide. De plus, j'ai apprécié de trouver la librairie par leur lien Github, de voir qu'elle est bien maintenue et utilisée par plus de 3.2 millions de personnes.

10.2.5 Extrait du site

General Usage

Usage is a two steps process:

First, a schema is constructed using the provided types and constraints:

```
JavaScript
const schema = Joi.object({a: Joi.string()});
```

Note that joi schema objects are immutable which means every additional rule added (e.g. .min(5)) will return a new schema object.

Second, the value is validated against the defined schema:

```
JavaScript
const { error, value } = schema.validate({ a: 'a string' });
```

If the input is valid, then the error will be undefined. If the input is invalid, error is assigned a [ValidationError](#) object providing more information.

The schema can be a plain JavaScript object where every key is assigned a joi type, or it can be a joi type directly:

```
JavaScript
const schema = Joi.string().min(10);
```

If the schema is a joi type, the schema.validate(value) can be called directly on the type. When passing a non-type schema object, the module converts it internally to an object() type equivalent to:

JavaScript

```
const schema = Joi.object().keys({a: Joi.string()});
```

When validating a schema:

Values (or keys in case of objects) are optional by default.

JavaScript

```
Joi.string().validate(undefined); // validates fine
```

To disallow this behavior, you can either set the schema as required(), or set presence to "required" when passing options:

JavaScript

```
Joi.string().required().validate(undefined);
```

or

```
Joi.string().validate(undefined, /* options */ { presence: "required" });
```

- Strings are utf-8 encoded by default.
- Rules are defined in an additive fashion and evaluated in order, first the inclusive rules, then the exclusive rules.

2.1.4. Traduction de l'extrait

Utilisation générale

L'utilisation de Joi se fait en deux étapes :

D'abord, un schéma est construit en utilisant les types et contraintes fournis :

Unset

```
const schema = Joi.object({ a: Joi.string() });
```

- Notez que les objets de schéma Joi sont immuables, ce qui signifie que chaque règle supplémentaire ajoutée (par exemple, .min(5)) retournera un nouvel objet de schéma.

Ensuite, la valeur est validée par rapport au schéma défini :

Unset

```
const { error, value } = schema.validate({ a: 'une chaîne de caractères' });
```

- Si l'entrée est valide, alors l'erreur sera undefined. Si l'entrée est invalide, error sera assignée à un objet ValidationError fournissant plus d'informations.

Le schéma peut être un simple objet JavaScript où chaque clé est assignée à un type Joi, ou il peut s'agir directement d'un type Joi :

Unset

```
const schema = Joi.string().min(10);
```

Si le schéma est un type Joi, on peut appeler directement schema.validate(value) sur ce type. Lorsqu'on passe un schéma qui n'est pas un type (un objet), le module le convertit en interne en un type object() équivalent à :

Unset

```
const schema = Joi.object().keys({ a: Joi.string() });
```

Lors de la validation d'un schéma :

Les valeurs (ou les clés dans le cas des objets) sont optionnelles par défaut.

Exemple :

Unset

```
Joi.string().validate(undefined); // valide correctement
```

Pour désactiver ce comportement, vous pouvez soit définir le schéma comme étant required(), soit définir l'option présence à "required" lors du passage des options :



```

Unset
Joi.string().required().validate(undefined);
// ou
Joi.string().validate(undefined, /* options */ { presence: "required" });

```

Les chaînes de caractères sont encodées en UTF-8 par défaut.

Les règles sont définies de manière additive et évaluées dans l'ordre : d'abord les règles inclusives, puis les règles exclusives.

10.3 Exploitation du résultat

10.3.1 Extrait de code

```

validateSignupData(data) {
  const schema = Joi.object({
    username: Joi.string().min(3).max(30).required().messages({
      'string.empty': 'Le nom d\'utilisateur ne peut pas être vide.',
      'string.min': 'Le nom d\'utilisateur doit contenir au moins 3 caractères.',
      'string.max': 'Le nom d\'utilisateur ne peut pas dépasser 30 caractères.'
    }),
    email: Joi.string().email().required().messages({
      'string.email': 'L\'adresse email doit être valide.',
      'string.empty': 'L\'email ne peut pas être vide.'
    }),
    password: Joi.string().pattern(new RegExp('^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$')).required().messages({
      'string.pattern.base': 'Le mot de passe doit contenir au moins 8 caractères, incluant des majuscules, minuscules, chiffres et caractères spéciaux.',
      'string.empty': 'Le mot de passe ne peut pas être vide.'
    }),
    confirmPassword: Joi.any().valid(Joi.ref('password')).required().messages({
      'any.only': 'Les mots de passe doivent correspondre.'
    })
  });

  return schema.validate(data, { abortEarly: false });
}

```

10.3.2 Argumentation

L'utilisation de **Joi** dans la fonction **validateSignUpData** est pertinente pour :

- Assurer la validité des données utilisateurs avant tout traitement. Cela empêche des entrées incorrectes ou malveillantes de causer des erreurs ou des failles de sécurité dans le système.

- Simplifier la gestion des règles complexes de validation, comme celles pour les mots de passe (regex et comparaison des mots de passe).
- Améliorer l'expérience utilisateur grâce aux messages d'erreur clairs.
- Renforcer la sécurité de l'application avec des critères stricts pour des données sensibles comme le mot de passe.

En résumé, **Joi** permet d'écrire un code de validation efficace, sécurisé et maintenable, ce qui est essentiel dans des applications où les données d'entrée des utilisateurs sont sensibles.

11. Annexes

11.1 Code de la page d'options.

1. **Pressable** disabled tant que le nombre d'équipes n'est pas sélectionné.

```
<Pressable
    onPress={openModal}
    style={[
        styles.modalButton,
        !selectedNbTeam ? styles.disabledButton : null
    ]}
    disabled={!selectedNbTeam}
>
    <Text style={styles.teamsNamesText}>Définir les noms d'équipes</Text>
    {errorNameTeams && <Text style={styles.errorMessage}>Vous devez entrer les noms des équipes</Text>}
</Pressable>
```

2. **Modal** qui contient les champs de saisie pour les noms d'équipes.

```
<Modal visible={modalVisible} animationType="slide" onRequestClose={closeModal}>
    <View style={styles.teamNamesModalContainer}>
        <ScrollView
            style={styles.scrollViewContainer}
            contentContainerStyle={styles.scrollViewContent}>
            <{renderTextInputFields()}>
        </ScrollView>
        <View style={styles.btnContainer}>
            <Pressable
                style={styles.confirmButton}
                onPress={closeModal}>
                <FontAwesomeIcon style={styles chevronRight} icon={'chevron-right'} color='white' size={30} />
            </Pressable>
        </View>
    </View>
</Modal>
```

```
const renderTextInputFields = () => {
  const textInputs = [];
  for (let i = 0; i < selectedNbTeam; i++) {
    textInputs.push(
      <View style={styles.inputContainer} key={i}>
        <Text style={styles.label}>Nom de l'équipe {i + 1}</Text>
        <TextInput
          style={styles.input}
          onChangeText={(value) => handleTeamNameChange(i, value)}
        />
      </View>
    );
  }
  return textInputs;
};
```

3. Fonction **handleConfirmClick** qui vérifie qu'il n'y a pas d'erreurs avant d'appeler la fonction **addOptions** pour inscrire les options en base de données.

```

const handleConfirmClick = async () => {
    setErrorNbTeam(false);
    setErrorNameTeams(false);
    setErrorTeamMembers(false);
    setErrorTimer(false);

    let isValid = true;
    if (!selectedNbTeam) {
        setErrorNbTeam(true);
        isValid = false;
    }

    if (!teamNames || teamNames.length !== selectedNbTeam ||
        teamNames.includes(undefined) || teamNames.includes(''))
    {
        setErrorNameTeams(true);
        isValid = false;
    }

    if (!selectedTeamMembers) {
        setErrorTeamMembers(true);
        isValid = false;
    }

    if (!selectedTimer) {
        setErrorTimer(true);
        isValid = false;
    }

    if (isValid) {
        await addOptions(gameId.gameId, {
            nbTeams: selectedNbTeam,
            nameTeams: teamNames,
            teamMembers: selectedTeamMembers,
            chrono: selectedTimer,
        });
        openRecapModal();
    }
};

```

4. Fonction **addOptions** qui fait la requête POST pour inscrire les options en BDD.

```
const addOptions = async (gameId, options) => {
  try {
    const response = await fetch(`/${apiUrl}/games/${gameId}/options`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ gameId: gameId, options }),
    });

    if (!response.ok) {
      throw new Error('Network response regarding options was not ok');
    }

    const data = await response.json();
    return data;
  } catch (error) {
    console.error('Erreur:', error);
    throw error;
  }
};
```

5. Méthode **addOptions** du contrôleur Games qui valide les données reçues et les inscrit en BDD

```

addOptions: async (req, res) => {
  const { gameId, options } = req.body;

  console.log('Received gameId:', gameId);
  console.log('Received options:', options);

  const { error } = optionsSchema.validate(options);
  if (!mongoose.Types.ObjectId.isValid(gameId) || error) {
    console.error('Invalid input:', { gameId, options, error });
    return res.status(400).json({ error: 'Invalid input', details: error.details ? error.details : null });
  }

  try {
    const game = await GamesModel.findByIdAndUpdate(
      new mongoose.Types.ObjectId(gameId),
      { options: options },
      { new: true }
    );

    if (!game) {
      console.error('Game not found for gameId:', gameId);
      return res.status(404).send('Game not found');
    } else {
      console.log('Game options updated successfully:', game);
      return res.json(game);
    }
  } catch (error) {
    console.error('Error updating game options:', error);
    return res.status(500).send('Internal server error');
  }
}
},
6.

```

6. Schéma de l'objet options défini dans le modèle

```

const GamesSchema = new Schema({
  themes: Array,
  options: {
    nbTeams: Number,
    nameTeams: Array,
    teamMembers: Number,
    chrono: Number
  },
  points: [
    {
      teamName: String,
      scoreRound1: Number,
      scoreRound2: Number,
      scoreRound3: Number
    }
  ],
  players: Array
});

```

7. Inscription en BDD

```
{  
  "_id": {  
    "$oid": "672c99d6579472388a05c331"  
  },  
  "themes": [  
    "660aa7579e9dc3bcfd81b326"  
  ],  
  "options": {  
    "nbTeams": 2,  
    "nameTeams": [  
      "Bob",  
      "Bobette"  
    ],  
    "teamMembers": 2,  
    "chrono": 30  
  },  
  "players": [],  
  "points": [],  
  "__v": 0  
}
```

8. Récapitulatif des options

```

<Modal visible={recapModalVisible} animationType="slide" onRequestClose={closeRecapModal}>
  <View style={styles.modalContainer}>
    <Text style={styles.modalTitle}>Récapitulatif des options :</Text>

    <View style={styles.recapOptionContainer}>
      <Text style={styles.optionTitle}>Nombre d'équipes</Text>
      <Text style={styles.optionValue}>{selectedNbTeam}</Text>
    </View>

    <View style={styles.recapOptionContainer}>
      <Text style={styles.optionTitle}>Noms des équipes</Text>
      <Text style={styles.optionValue}>{teamNames.join(', ')})</Text>
    </View>

    <View style={styles.recapOptionContainer}>
      <Text style={styles.optionTitle}>Nombre de joueurs par équipe</Text>
      <Text style={styles.optionValue}>{selectedTeamMembers}</Text>
    </View>

    <View style={styles.recapOptionContainer}>
      <Text style={styles.optionTitle}>Chronomètre</Text>
      <Text style={styles.optionValue}>{selectedTimer}</Text>
    </View>

    <Pressable onPress={toStart} style={styles.toStartButton}>
      <Animated.View style={{ transform: [{ scale: playButtonAnimation }] }}>
        <FontAwesomeIcon icon='play' size={40} color="white" />
      </Animated.View>
    </Pressable>

    <Pressable onPress={closeRecapModal} style={styles.modifyButton}>
      <Text style={styles.modalText}>Modifier les options</Text>
    </Pressable>
  </View>
</Modal>

```

Requête SQL pour obtenir toutes les cartes d'un thème

```
SELECT
    themes.ID_themes
    themes.label
    themes.picture
    cards.ID_cards
    cards.element
FROM
    themes
JOIN
    cards on themes.ID_themes
        = cards.ID_themes
WHERE
    themes.label = 'Histoire'
```