

Dossier de projet professionnel



Coté pote

Titre professionnel : Concepteur développeur d'application

Nom : Arfi

Prénom : Benjamin

En Formation à La Plateforme_, 8 rue d'Hozier 13002 Marseille,

En alternance : Alstom Aix-en-Provence

1. Introduction	5
1.1. Présentation personnelle	5
1.2. Presentation of my activity in English.	5
1.3. Présentation de l'entreprise	6
1.4. Présentation de l'équipe Software Development	8
1.5. Mon rôle au sein de l'entreprise	9
2. Résumé du projet « Côté Pote »	10
2.1. Summary of the project "Côté Pote" in English.	11
2.2. Compétences couvertes par le projet	12
3. Cahier des charges	13
3.1. Définition des besoins	13
3.1.1. Les objectifs de l'application	13
3.1.2. Périmètre du projet	14
3.1.3. Fonctionnalités	14
3.1.4. Contraintes techniques	15
3.2 Conception	16
3.2.1 User Story	16
3.2.2 UX : User Experience	17
3.2.2.1 Définition du parcours utilisateur	19
3.2.3 UI : Côté Pote & charte graphique	20
3.2.4 Maquette & Prototype	20
3.3 Modélisation de la base de données	23
3.3.1 Méthode MERISE	23
3.3.2 Modèle conceptuel de données (MCD)	23
3.3.3 Modèle Logique de Données (MLD)	25
3.3.4 Modèle Physique de Données (MPD)	27
4. Collaboration à la gestion de projet	28
4.1. Objectifs	28
4.2. Organisation	28
4.3. Versionning & Workflow	29
Structure des branches avec GitFlow	31
4.4. Gestion des dépendances	31
5. Environnement Technique	33
5.1. Technologies & Outils	33
5.1.1. Spécificités techniques : Backend	33
5.1.1.1. API REST	33

5.1.1.2. Symfony / API Platform	33
5.1.1.3. Symfony / API Platform : Installation & Configuration	
34	
5.1.1.4. Crédation d'une Collection avec Symfony / API Platform	
39	
6. Crédation de l'entité avec Doctrine	39
6.1. Symfony / API Platform : Ajout des relations	46
6.1.1. Spécificités techniques : Frontend	50
6.1.1.1. React Native	50
6.1.1.2. Expo	51
6.1.1.3. React Navigation	54
6.1.1.4. Axios : Interaction avec la base de données	56
7. Présentation du jeu d'essai	60
7.1. Parcours utilisateur pour la connexion d'un user	60
8. Extrait d'une recherche à partir de site anglophone	76
Conclusion	77

1. Introduction

1.1. Présentation personnelle

Je m'appelle Benjamin Arfi et je suis passionné par l'informatique depuis toujours. Curieux et rigoureux, j'aime explorer les nouvelles technologies, résoudre des problèmes complexes et concevoir des solutions logicielles innovantes.

Actuellement, je travaille en tant qu'apprenti en validation logiciel pour les systèmes embarqués à Alstom . Mon rôle consiste à tester et valider le bon fonctionnement des logiciels intégrés dans des environnements critiques, en veillant à leur fiabilité, performance et conformité aux exigences. J'apprécie particulièrement l'automatisation des tests, l'analyse des résultats et l'amélioration continue des processus de validation.

Toujours en quête d'apprentissage, je me tiens informé des évolutions technologiques et des bonnes pratiques en développement et test logiciel. Mon objectif est d'acquérir une expertise approfondie dans le domaine informatique et de contribuer à des projets innovants et exigeants.

1.2. Presentation of my activity in English.

For my apprenticeship, I had the opportunity to work at **Alstom**, a global leader in the railway transportation sector. The company specializes in designing, manufacturing, and maintaining trains, trams, and metros, as well as developing innovative solutions to enhance the safety and efficiency of public transport. With a strong international presence, Alstom plays a key role in the transition towards more sustainable mobility.

My role within the company was to validate software and provide fixes for applications used to test electronic boards. We worked with **safety-critical motherboards** used in trains and trams to ensure key functions, such as stopping the train or managing security gates.

Previously, when a **UCS (Unité Centrale de Sécurité)** motherboard malfunctioned, it took weeks of investigation to identify the faulty component. To streamline this

process, we developed an application running on the **Zephyr OS**, allowing us to test each component on the board and quickly detect failures. This solution made it possible to replace only the defective component instead of discarding the entire board.

My main task was to ensure that the software accurately identified faulty components. To achieve this, I developed **scripts** and created a **Human-Machine Interface (HMI)** to facilitate software validation.

I also worked on a **drawer testing software**, which involved testing multiple boards, including **UCS** units and **IFEL** power supply boards. The IFEL board is responsible for **starting the tram and regulating its speed**. The goal was to precisely identify which board was causing a malfunction, making repairs more efficient.

1.3. Présentation de l'entreprise



Chiffres clés :

- **Chiffre d'affaires** : 17,5 milliards d'euros pour l'exercice fiscal 2023/2024.
- **Effectifs** : Plus de 80 000 employés répartis dans 64 pays, représentant 184 nationalités.
- **Présence mondiale** : Plus de 150 000 véhicules en service commercial à travers le monde.

Engagement en faveur de la mobilité durable :

Alstom se positionne en leader de la mobilité verte et intelligente, avec une stratégie axée sur l'innovation verte et digitale, l'efficacité opérationnelle et la responsabilité sociale.

Produits phares :

- **Avelia Horizon** : Le seul train à grande vitesse à deux niveaux au monde, combinant capacité élevée et efficacité énergétique.
- **TGV M** : Dernière génération de TGV développée en collaboration avec la SNCF, offrant modularité, performance et respect de l'environnement.

Projets récents :

- **Rames Régiolis** : Livraison des premières rames d'une nouvelle commande de cinq trains Régiolis à la région Sud, renforçant l'offre de transport régional.

Innovation et digitalisation :

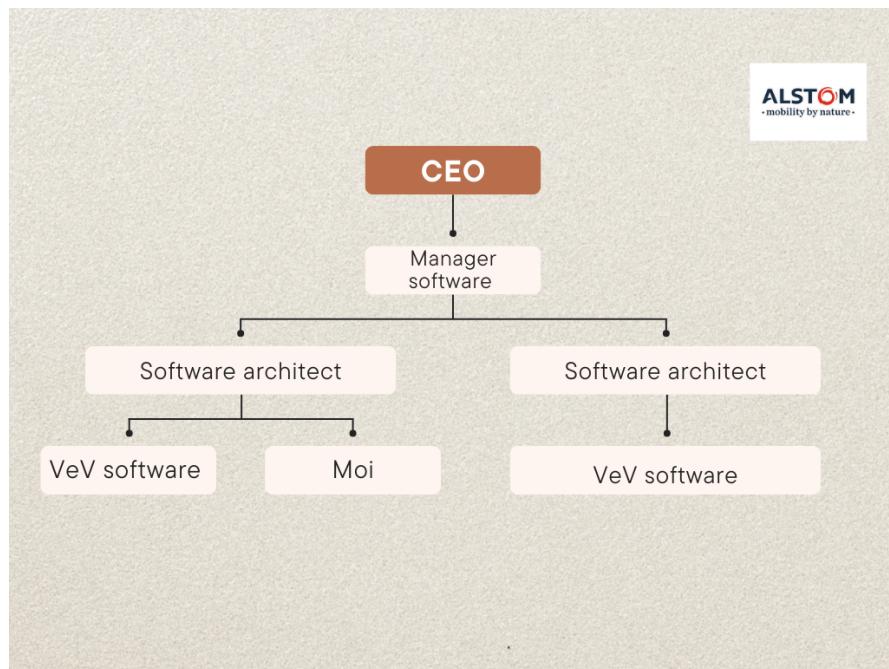
Alstom investit dans des univers virtuels pour donner vie à l'innovation ferroviaire, intégrant des technologies numériques avancées dans ses processus de conception et de production.

Grâce à son engagement envers l'excellence, l'innovation et la durabilité, Alstom continue de jouer un rôle clé dans la transformation du paysage mondial du transport ferroviaire.

1.4. Présentation de l'équipe Software Development

Au sein d'Alstom, je fais partie du pôle Software, où je collabore étroitement avec les Software Architects ainsi qu'avec l'équipe V&V Software. Mon travail consiste principalement à contribuer au développement et à la validation des logiciels, en garantissant leur conformité avec les exigences techniques et les standards de l'entreprise. L'équipe est encadrée par un responsable qui supervise l'ensemble des activités du pôle Software.

Au sommet de l'organigramme se trouve le Directeur Général d'Alstom France, qui dirige l'ensemble des activités de l'entreprise sur le territoire.



1.5. Mon rôle au sein de l'entreprise

Pour mon alternance, j'ai eu la chance de travailler pour **Alstom**, un leader mondial dans le secteur du transport ferroviaire. L'entreprise est spécialisée dans la conception, la fabrication et la maintenance de trains, tramways et métros, ainsi que dans le développement de solutions innovantes pour améliorer la sécurité et l'efficacité des transports publics. Présente à l'international, Alstom joue un rôle clé dans la transition vers une mobilité plus durable.

Mon rôle au sein de l'entreprise consistait à valider les logiciels et à apporter des correctifs aux applications de test des cartes électroniques. Nous travaillons avec des cartes mères sécuritaires utilisées dans les trains et tramways pour assurer des fonctions critiques, comme l'arrêt du train ou la gestion des portiques de sécurité.

Auparavant, lorsqu'une carte mère nommée **UCS (Unité Centrale de Sécurité)** présentait un dysfonctionnement, il fallait plusieurs semaines d'investigation pour identifier le composant défectueux. Pour optimiser ce processus, nous avons développé une application sous le **RTOS Zephyr** permettant de tester chaque composant de la carte et ainsi détecter rapidement les pannes. Grâce à cette solution, il est désormais possible de remplacer uniquement le composant défectueux plutôt que de jeter toute la carte.

Ma mission principale a été de vérifier que le logiciel identifie correctement les composants en panne. Pour cela, j'ai développé des scripts et une interface homme-machine (**IHM**) facilitant la validation du logiciel.

J'ai également travaillé sur un **logiciel de test tiroirs**, qui sont des ensembles de plusieurs cartes, notamment des **UCS** et des cartes d'alimentation, ainsi que des **IFEL**. Ces dernières gèrent le démarrage du tramway ainsi que la régulation de la vitesse. L'objectif était d'identifier précisément la carte responsable d'un dysfonctionnement afin d'optimiser les réparations.

2. Résumé du projet « Côté Pote »

Pendant nos périodes de formation, nous avons consacré du temps au développement d'une application mobile, J'ai choisi de créer une application de paris entre amis baptisée "Côté Pote".

Côté Pote est une application qui offre à chaque utilisateur qu'il soit seul ou intégré à un groupe la possibilité de créer et de partager des paris sur tous types de sujets:sports, musique, jeux-vidéo ou tout autre événement.

L'objectif principal de cette application est de permettre à des groupes d'amis (ou à des utilisateurs individuels) de lancer facilement des paris entre eux. Chaque pari publié apparaît dans un fil d'actualité, ainsi tous les utilisateurs peuvent consulter l'ensemble des paris en cours.

L'application s'inspire à la fois des plateformes de paris traditionnelles et des applications sociales dédiées à la convivialité entre amis.

J'ai souhaité proposer une application à l'ambiance ludique, reprenant les codes des applications de soirées entre amis et des applications de paris sportifs, tout en y apportant une touche innovante et une véritable plus-value pour ses utilisateurs.

2.1. Summary of the project "Côté Pote" in English.

Côté Pote is an application that allows a user or a group of users to publish bets of any kind, be it sports, music, video games...

The goal is to create an application where groups of friends or solo users without a group of friends can create bets.

The user will be able to find each bet posted on a feed where all bets can be viewed.

Users can place bets on any type of event (world cup, gaming competitions, etc.).

The application is inspired by both traditional betting applications and party applications.

I wanted a fun application, with the same codes as the party app or sports betting app, but with an added value.

The application is developed in React Native for the front-end and Symfony / API Platform for the back-end.

2.2. Compétences couvertes par le projet

Voici les compétences du référentiel couvertes par le développement réalisé sur ce projet.

- **Activité type 1 :** Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité
 - Maquetter une application
 - Développer des composants d'accès aux données
 - Concevoir et développer des interfaces utilisateur sécurisées
 - Prendre en compte l'accessibilité numérique (RGAA) lors de la conception des interfaces
 - Respecter les procédures qualité et les normes en vigueur
- **Activité type 2 :** Concevoir et développer la persistance des données en intégrant les recommandations de sécurité
 - Concevoir une base de données
 - Mettre en place une base de données
 - Développer des composants dans le langage d'une base de données
 - Assurer la sécurité, la confidentialité et l'intégrité des données
 - Appliquer les exigences du RGPD (mentions légales, gestion des consentements, droit à l'oubli)
- **Activité type 3 :** Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité
 - Concevoir une application
 - Développer des composants métier
 - Construire une application organisée en couches
 - Développer une application mobile
 - Préparer et exécuter les plans de tests d'une application
 - Préparer et exécuter le déploiement d'une application
 - Installer et configurer l'environnement de travail adapté au projet
 - Mettre en place une veille technologique et appliquer les évolutions pertinentes

3. Cahier des charges

Avant de commencer la réalisation de la maquette ou toute étape de développement, notre groupe composé de Robin Papazi, Robin Arbona et moi-même on a élaboré un cahier des charges afin de cadrer le projet, définir les fonctionnalités et de fixer des objectifs clairs.

3.1. Définition des besoins

3.1.1. Les objectifs de l'application

• Objectifs qualitatifs

Créer une application mobile ludique permettant à des groupes d'amis ou à des utilisateurs seuls de lancer et participer à des paris (et éventuellement des défis à l'avenir).

Disponibilité : L'application sera disponible sur Android et iOS, avec une expérience utilisateur fluide et intuitive.

• Objectifs quantitatifs

- **Sécurité :** Stocker de manière sécurisée les informations personnelles et les données liées aux paris.
- **Identité visuelle :** Proposer une identité visuelle moderne et attractive, basée sur un design épuré (Flat Design).
- **Navigation :** Garantir une navigation naturelle et intuitive, avec une forte attention portée à l'UI/UX pour toucher un large public.

• Cibles utilisateurs

Toute personne souhaitant s'amuser en lançant ou participant à des paris entre amis, quel que soit son profil social ou ses loisirs.

Utilisateurs intéressés par le suivi de leurs émotions ou phases de bien-être via des paris thématiques.

Utilisateurs inscrits uniquement, qui pourront :

- Participer à des paris et consulter leur historique
- Modifier leur profil (photo, nom, prénom, date de naissance, téléphone, email)
- Accéder aux défis hebdomadaires
- Créer et gérer des groupes de paris
- Inviter des amis dans les groupes

3.1.2. Périmètre du projet

Langue et accessibilité

L'application sera disponible uniquement en français.

Elle sera adaptée à tous les systèmes d'exploitation mobiles (Android et IOS) afin de garantir à chaque utilisateur une expérience optimale, quel que soit son appareil.

Fonctionnalités natives

L'application pourra exploiter les fonctionnalités natives des terminaux mobiles, comme l'appareil photo ou la géolocalisation, pour enrichir l'expérience utilisateur.

Sécurité des données

Les données personnelles des utilisateurs seront stockées dans une base de données sécurisée.

3.1.3. Fonctionnalités

Page d'accueil

L'application propose une page d'accueil depuis laquelle il est possible de se connecter ou de s'inscrire.

Fonctionnement de l'application

Après inscription, l'utilisateur peut consulter un guide ou une présentation du fonctionnement de l'application.

Espace utilisateur sécurisé

Chaque utilisateur bénéficie d'un espace personnel sécurisé, protégé par un mot de passe. Les mots de passe sont stockés de manière sécurisée (hachés) dans la base de données, conformément à la réglementation RGPD.

Gestion du profil

Une page de profil est mise à disposition pour permettre aux utilisateurs de modifier, ajouter ou supprimer leurs informations personnelles.

Gestion des paris

- Création, modification, suppression :**

Les utilisateurs peuvent créer, modifier ou supprimer un pari.

- Participation :**

Il est possible de participer à un pari.

- Recherche :**

Une fonctionnalité de recherche permet de retrouver un pari spécifique au sein de l'application.

3.1.4. Constraintes techniques

Afin d'assurer la compatibilité native avec les systèmes d'exploitation iOS et Android, des choix techniques spécifiques ont été effectués.

Du côté du développement frontend, nous avons opté pour l'utilisation de la bibliothèque **Expo React Native**. Cette solution permet de concevoir des applications mobiles compatibles avec les deux plateformes à partir d'une seule base de code, sans nécessiter la maîtrise des langages natifs propres à chaque système. Elle offre ainsi un gain significatif de temps et de productivité, tout en garantissant une expérience utilisateur fluide et ergonomique.

Par ailleurs, la sécurité constitue une priorité : la base de données ainsi que l'API permettant les requêtes HTTP seront protégées. Les mots de passe des utilisateurs seront stockés de façon sécurisée, grâce à un hachage appliqué avant leur enregistrement en base de données. Enfin, chaque utilisateur aura la possibilité de consulter, exporter ou demander la suppression de ses données personnelles à tout moment, conformément à la réglementation en vigueur.

3.2 Conception

Afin de mieux appréhender nos étapes de conception et d'intégration, nous avons fait le choix d'utiliser une approche AGILE, celle-ci nous a paru idéale pour optimiser au mieux notre travail.

3.2.1 User Story

Avant d'entamer les étapes de conception UX et UI, nous avons mené une veille approfondie et testé plusieurs applications mobiles similaires, telles que Winamax, Piccolo ou Betclic. Cette démarche nous a permis de nous placer dans la peau des utilisateurs et de préciser les fonctionnalités essentielles à intégrer dans notre application.

Les « User Stories » ont été utilisées pour identifier et formaliser les besoins des utilisateurs. Elles permettent de définir les caractéristiques, les fonctions et les exigences de l'application à développer. Grâce à elles, il devient plus facile de comprendre l'utilité d'une fonctionnalité, d'évaluer sa nécessité et de déterminer sa priorité dans le projet.

L'utilisation des user stories a ainsi permis à notre équipe de construire une arborescence qui reflète fidèlement le parcours utilisateur au sein de l'application.

3.2.2 UX : User Experience

Les premières minutes d'utilisation d'une application mobile sont déterminantes pour garantir son adoption et son utilisation sur le long terme. Il est donc essentiel d'offrir une expérience agréable, intuitive et visuellement attractive, avec une identité jeune et dynamique.

L'UX (User Experience) appliquée au mobile consiste à concevoir des applications qui procurent une expérience ludique, conviviale et humaine. Dans le cas de notre projet, cette démarche a été mobilisée dès le début, lors de l'élaboration de la stratégie, afin de :

- **Faciliter l'utilisation de l'application** et son fonctionnement
- **Contribuer à l'amusement de l'utilisateur** en sollicitant sa créativité lors de la création de paris
- **Permettre un usage sur différents appareils** et dans des contextes variés (soirées, compétitions, etc.)

Pour optimiser l'application et garantir une expérience plaisante, nous avons porté une attention particulière aux points suivants :

1. Limiter et Simplifier

- **Limiter les fonctionnalités** à l'essentiel afin de privilégier l'amusement sans complexifier l'usage
- **Simplifier le contenu** pour proposer une application fluide et rapide, avec un objectif de fluidité prioritaire

2. Guider l'Utilisateur par le Design

- **Hiérarchiser l'information** grâce à une utilisation réfléchie des couleurs et des formes
- **Inciter à l'action** par une charte graphique cohérente et déclinée sur toute l'application, pour rassurer l'utilisateur et garantir un rendu harmonieux

3. Limiter l'Effort de l'Utilisateur

- **Réduire le nombre d'actions** nécessaires : limiter les étapes, supprimer les champs superflus dans les formulaires, utiliser les données déjà fournies
- **Afficher les bons claviers au bon moment** et exploiter les fonctionnalités natives du smartphone (appareil photo, géolocalisation, etc.)

4. Travailler l'Ergonomie

- **Adapter la taille des éléments** pour une utilisation à une main, dans toutes

- les conditions de navigation
- **Ne pas imposer d'actions obligatoires** et réfléchir à la fréquence des notifications push

5. Inclure la Diversité

- **Concevoir une application accessible** à toutes et tous, sans nécessiter d'adaptation particulière de la part des utilisateurs

6. Respecter les Conventions Mobiles

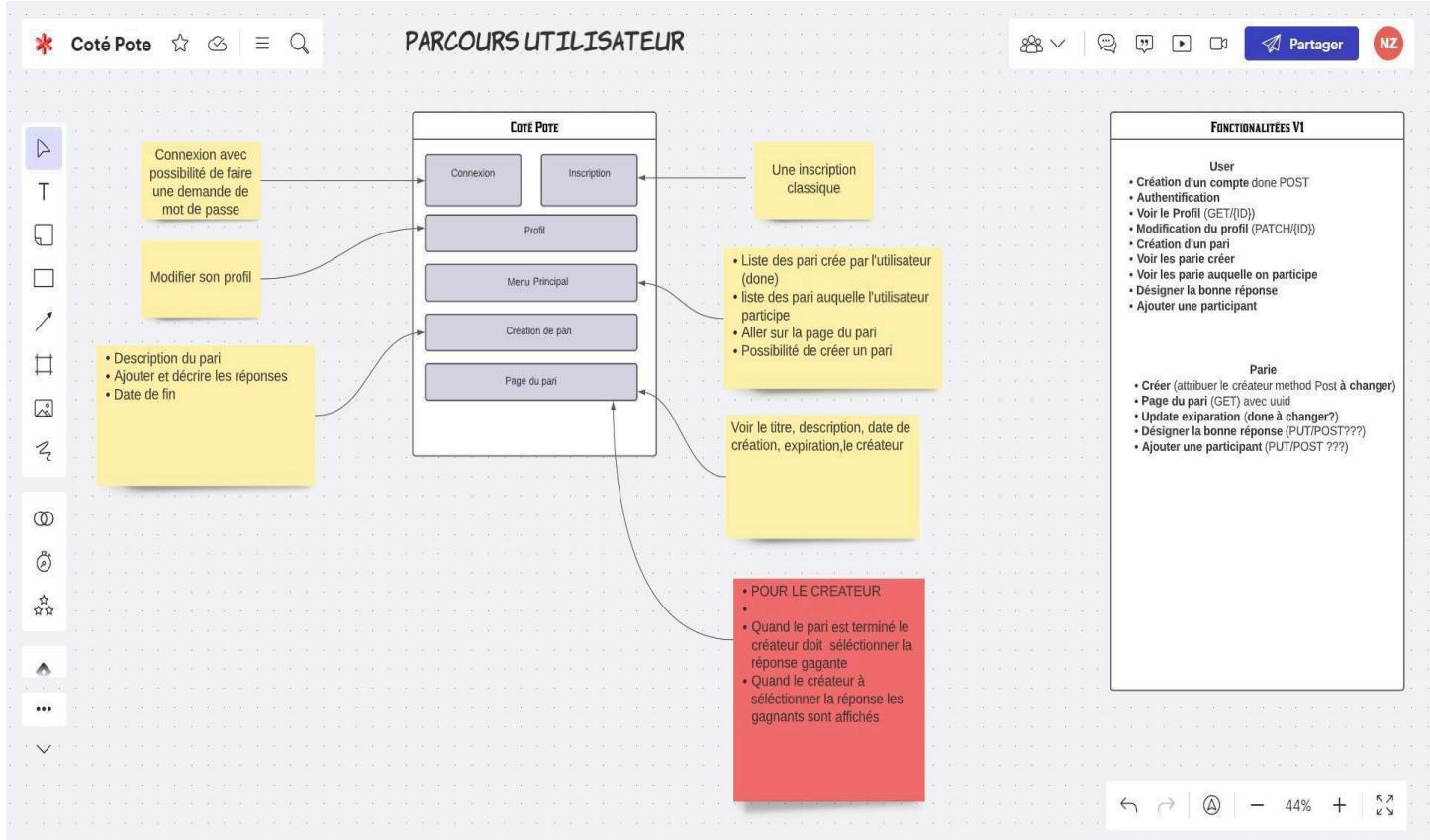
- **Intégrer les conventions mobiles** attendues par les utilisateurs : navigation par Tab Bar, menu burger, etc.

7. Pérenniser et Optimiser l'Application

- **Proposer le bon contenu au bon moment** et estimer le degré de personnalisation en fonction des fonctionnalités et des ressources disponibles
- **Travailler l'ASO (App Store Optimization)** : rédiger des descriptions accrocheuses, choisir la bonne catégorie, ajouter des captures d'écran attrayantes, et encourager les avis positifs pour maximiser la visibilité sur les stores

3.2.2.1 Définition du parcours utilisateur

Afin de définir le parcours utilisateur, nous avons identifié les étapes clés de la navigation au sein de l'application. À partir de ces étapes, nous avons traduit chaque possibilité et chaque action permettant d'accéder aux différentes pages. Cette démarche nous a permis de modéliser un schéma complet, illustrant l'ensemble du parcours utilisateur ainsi que toutes les fonctionnalités disponibles dans l'application.



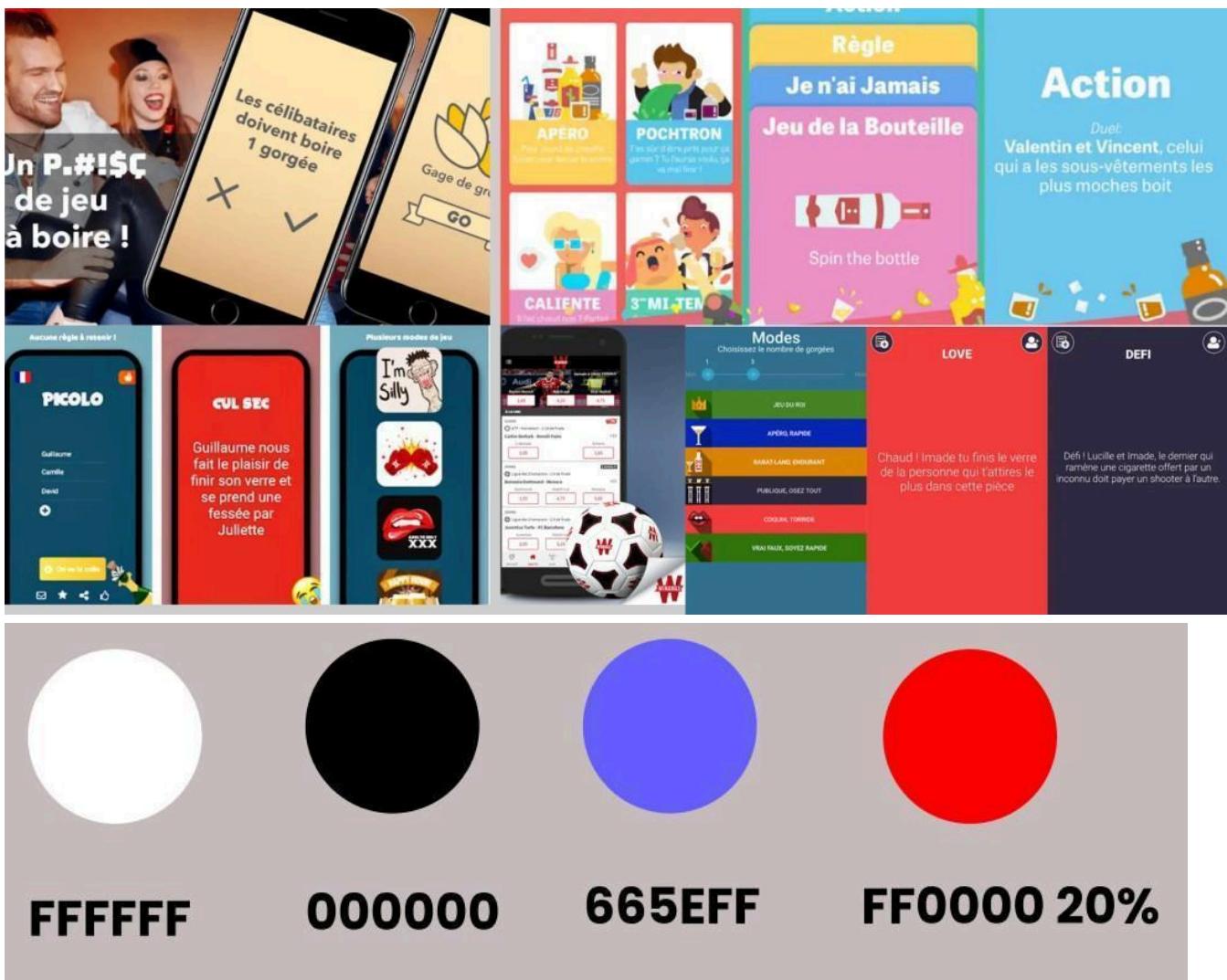
3.2.3 UI : Côté Pote & charte graphique

Pour concevoir la maquette, nous avons d'abord effectué une sélection de designs inspirés d'applications mobiles existantes, à travers un travail de benchmarking. Cette démarche nous a permis de rassembler diverses idées et de définir une direction pour la charte graphique et le design de l'application.

L'élaboration de la charte graphique inclut :

- Un logo bien visible sur l'écran d'accueil
- Une liste de polices à importer et à utiliser sur l'ensemble de l'application
- Les couleurs principales de l'application
- Des icônes, formes et couleurs de boutons unifiées et cohérentes

Cette charte graphique assure ainsi une identité visuelle claire et harmonieuse tout au long de l'expérience utilisateur.



3.2.4 Maquette & Prototype

Pour la création de la maquette, nous avons choisi d'adopter une approche inspirée des principes de l'**Atomic Design**. Cette méthodologie de conception d'interfaces graphiques consiste à décomposer l'interface en composants élémentaires, appelés « atomes », qui peuvent ensuite être combinés pour former des modules plus complexes (« molécules », « organismes »), puis des pages complètes.

Chaque type de composant a une fonction précise. Leur assemblage permet de construire un ensemble de pages cohérent, tout en facilitant la maintenance grâce à la modularité inhérente à ce type de conception.

Cette approche s'apparente à la programmation orientée objet, où chaque élément possède ses propres propriétés et responsabilités. Elle est particulièrement adaptée à notre projet, car nous utilisons la librairie **React Native** qui repose elle-même sur un design pattern de création d'application par composants. Ainsi, l'Atomic Design et React Native s'articulent parfaitement pour garantir une architecture solide, évolutive et facile à maintenir.



Pour reprendre les principes de l'Atomic Design nous avons créé un Kit UI. C'est un ensemble d'éléments graphiques, basés sur la hiérarchie citée précédemment (atomes, molécules, organismes) et qui seront réutilisés pour créer chaque page.

The image displays a user interface (UI) kit and several mobile application screens. The top section shows a dark-themed 'Créer un pari' screen with buttons, icons, and a color palette. To its right is a 'Logo' section featuring two stylized smiley faces (white and black) and their corresponding color codes (FFFFFF and 000000). Below these are color swatches and hex codes: white (FFFFFF), black (000000), blue (665EFF), and red (FF0000 20%). The bottom section shows a grid of mobile screens: a sign-up form, a profile view, a search screen, a 'Mes paris en cours' screen, a 'Mes paris fini' screen, and a 'Bookmaker ou Sûrveur?' screen. A large central image shows a dark-themed mobile application with a navigation bar, a central content area with a profile picture, and a sidebar with various sections like 'Inscription' and 'Modifier mes infos'.

3.3 Modélisation de la base de données

3.3.1 Méthode MERISE

La méthode MERISE (Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise), développée en France à la fin des années 1970, vise à fournir une démarche structurée pour la modélisation et la conception des systèmes d'information, en particulier pour la gestion des fichiers de données, des bases de données et des systèmes de gestion de bases de données (SGBD).

MERISE se distingue par sa séparation claire entre les données et les traitements, et s'appuie sur plusieurs niveaux de modélisation pour garantir la robustesse et la pérennité du système d'information. Pour la conception de bases de données, la méthode s'articule autour de trois étapes principales :

- **Le Modèle Conceptuel des Données (MCD)** : identification des entités, de leurs attributs et des relations entre elles, afin de représenter de façon abstraite les besoins métier.
- **Le Modèle Logique des Données (MLD)** : traduction du modèle conceptuel en un schéma relationnel adapté à la structure d'une base de données.
- **Le Modèle Physique des Données (MPD)** : implémentation technique optimisée pour la performance et le stockage, en fonction du SGBD choisi.

Dans le cadre du projet Côté Pote, la méthode MERISE a été particulièrement utile pour la conception de la base de données. En s'appuyant sur ses principes de modélisation, nous avons pu structurer les informations, définir précisément les relations entre les différentes entités (utilisateurs, paris, groupes, etc.) et garantir la cohérence ainsi que l'évolutivité du système d'information.

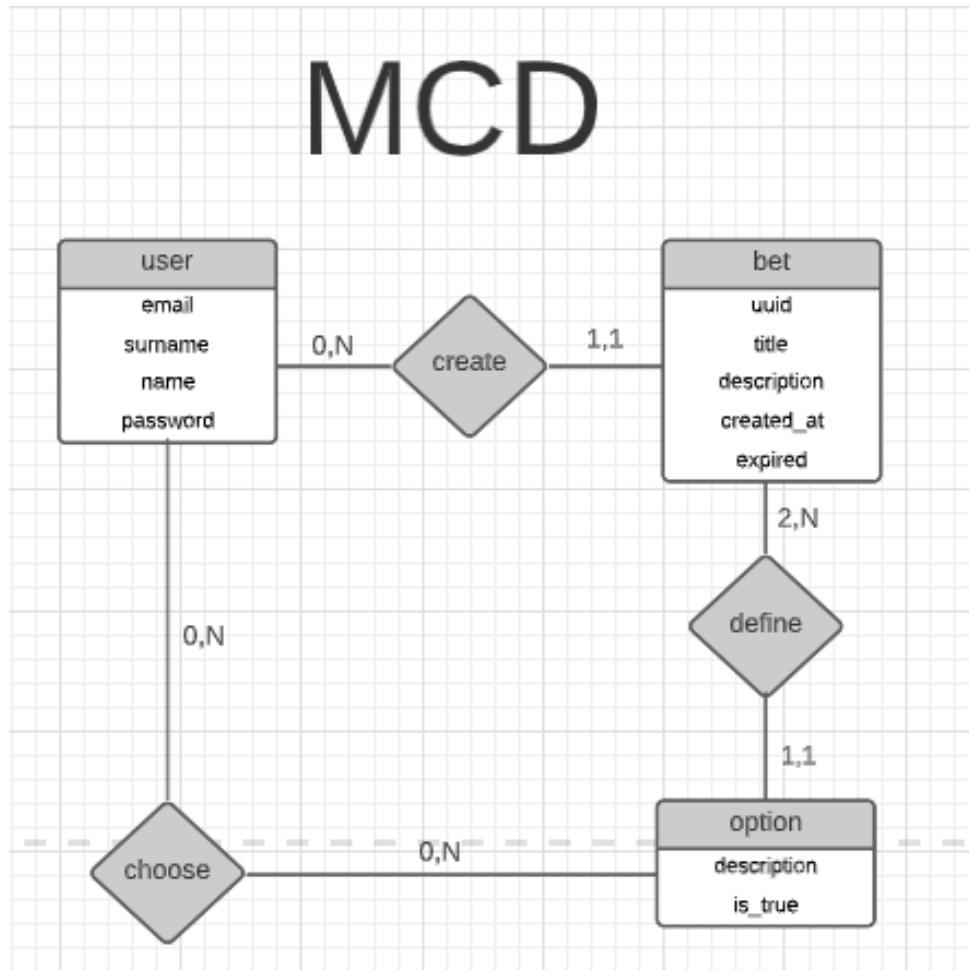
3.3.2 Modèle conceptuel de données (MCD)

Il s'agit ici d'un modèle simplifié de la base de données, où les tables sont représentées uniquement au stade d'entités, ce qui correspond au schéma Entité/Association du Modèle Conceptuel de Données (MCD).

La première étape a consisté à recenser l'ensemble des besoins des futurs utilisateurs de Côté Pote. Sur cette base, on a pu établir les règles de gestion concernant les données à conserver.

Ensuite, il a fallu définir le dictionnaire des données, c'est-à-dire l'ensemble des données élémentaires qui seront stockées dans la base, ainsi que leurs principales caractéristiques. Parmi ces caractéristiques figurent notamment la référence de la donnée (identifiant unique), sa désignation, son type, etc.

Enfin, à partir des informations collectées, chaque entité a été créée et décrite par un ensemble de propriétés. Les associations entre ces entités ont également été définies, ce qui permet de préciser les liens et les cardinalités entre elles dans le MCD.



3.3.3 Modèle Logique de Données (MLD)

Le **modèle logique de données (MLD)** constitue une étape intermédiaire entre le modèle conceptuel de données (MCD) et le modèle physique de données (MPD). Il traduit les entités et relations du MCD en structures adaptées à un système de gestion de base de données relationnel.

Principes de conversion du MCD vers le MLD

- **Transformation des entités en tables**

Chaque entité identifiée dans le MCD devient une table dans le MLD. Dans un SGBD relationnel, une table est une structure où chaque ligne représente un enregistrement (objet) et chaque colonne correspond à une propriété (attribut) de cet objet. Ces colonnes reprennent les caractéristiques définies dans le dictionnaire de données du MCD.

- **Gestion des identifiants**

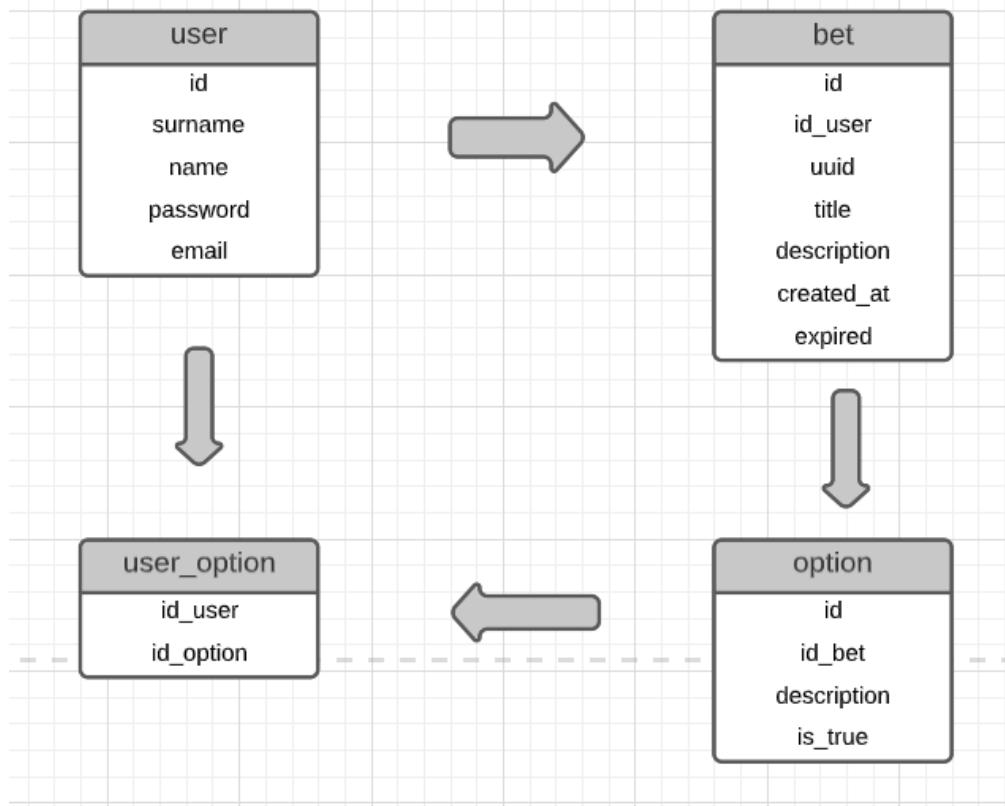
Les identifiants uniques de chaque entité deviennent des clés primaires dans les tables correspondantes. Ces clés primaires permettent d'identifier de façon unique chaque enregistrement et ne peuvent pas être nulles. Les autres propriétés de l'entité deviennent des attributs de la table.

- **Représentation des relations et cardinalités**

Les relations de type « 0:n » ou « 1:n » sont traduites en ajoutant la clé primaire de la table « parent » comme clé étrangère dans la table « enfant ».

Pour les relations de type « n:n », une table de jonction est créée. Cette table contient les clés primaires de chacune des tables liées, qui deviennent alors des clés étrangères dans la table de jonction.

MLD



Ainsi, dans le MLD, on observe que les clés primaires des tables associées sont utilisées comme clés étrangères dans les tables qui reçoivent les données, assurant ainsi l'intégrité référentielle et la cohérence des liens entre les différentes entités.

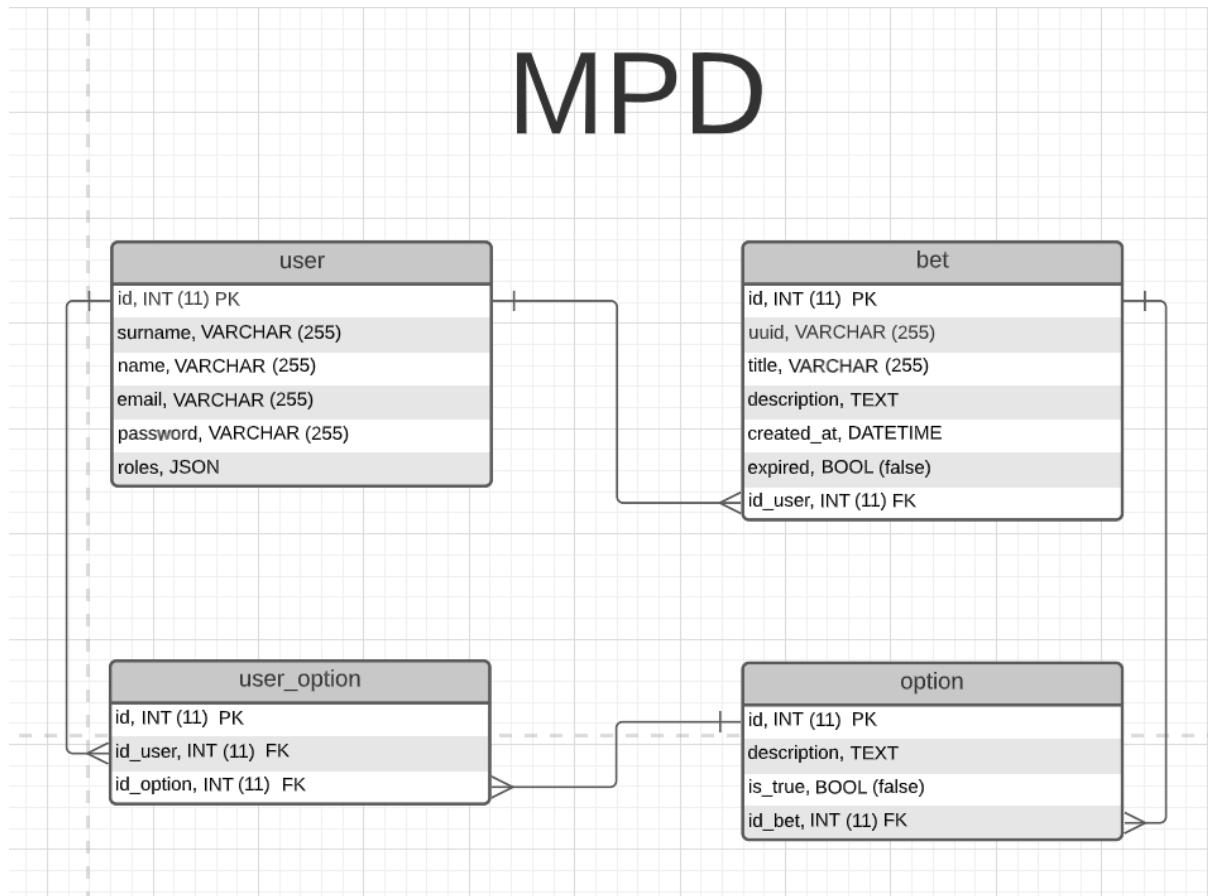
3.3.4 Modèle Physique de Données (MPD)

Le **Modèle Physique de Données (MPD)** représente la dernière étape de la modélisation selon la méthode Merise. Il consiste à traduire le modèle logique de données (MLD) en une structure concrète, adaptée au système de gestion de base de données (SGBD) choisi.

À ce stade, il s'agit d'implémenter la structure finale de la base de données, en précisant pour chaque table les types de données des colonnes (par exemple : **VARCHAR**, **INT**, **DATE**), leur taille, ainsi que les contraintes d'intégrité (clés primaires, clés étrangères, contraintes **NOT NULL**, **UNIQUE**, etc.). Les entités deviennent des tables, les propriétés des champs, et les relations sont matérialisées par des clés étrangères ou des tables de jonction selon les cardinalités.

Dans le cadre du projet Côté Pote, cette étape s'est traduite par l'implémentation du modèle dans le SGBD **MySQL**, en utilisant le moteur de stockage **InnoDB**. Ce choix technique permet de gérer efficacement les contraintes d'intégrité référentielle, notamment les clés étrangères, garantissant la cohérence des données au sein de la base.

En résumé, le MPD concrétise la modélisation en une base de données opérationnelle, prête à être déployée et exploitée par l'application, tout en tenant compte des spécificités techniques du SGBD retenu.



4. Collaboration à la gestion de projet

4.1. Objectifs

Avant de débuter le développement, il était indispensable de définir des objectifs précis concernant les livrables attendus.

Nous avons ainsi choisi de réaliser une **version bêta** de l'application, intégrant les fonctionnalités principales. Cette version a pour vocation d'être testée par un panel d'utilisateurs afin de recueillir leurs retours et d'orienter les évolutions futures du projet en fonction de leurs besoins et suggestions.

La version bêta permet notamment à un utilisateur de :

- S'inscrire et se connecter,
- Ajouter un pari,
- Participer à un pari,
- Accéder à une page profil pour consulter et modifier ses informations personnelles.

La navigation entre les différentes interfaces est facilitée par une **Tab Bar** située en bas de l'application, garantissant une expérience utilisateur fluide et intuitive.

Pour respecter les délais imposés par le rythme de l'alternance, nous avons fait des choix stratégiques en matière d'organisation, de technologies et d'outils, afin d'optimiser le développement et la gestion du projet.

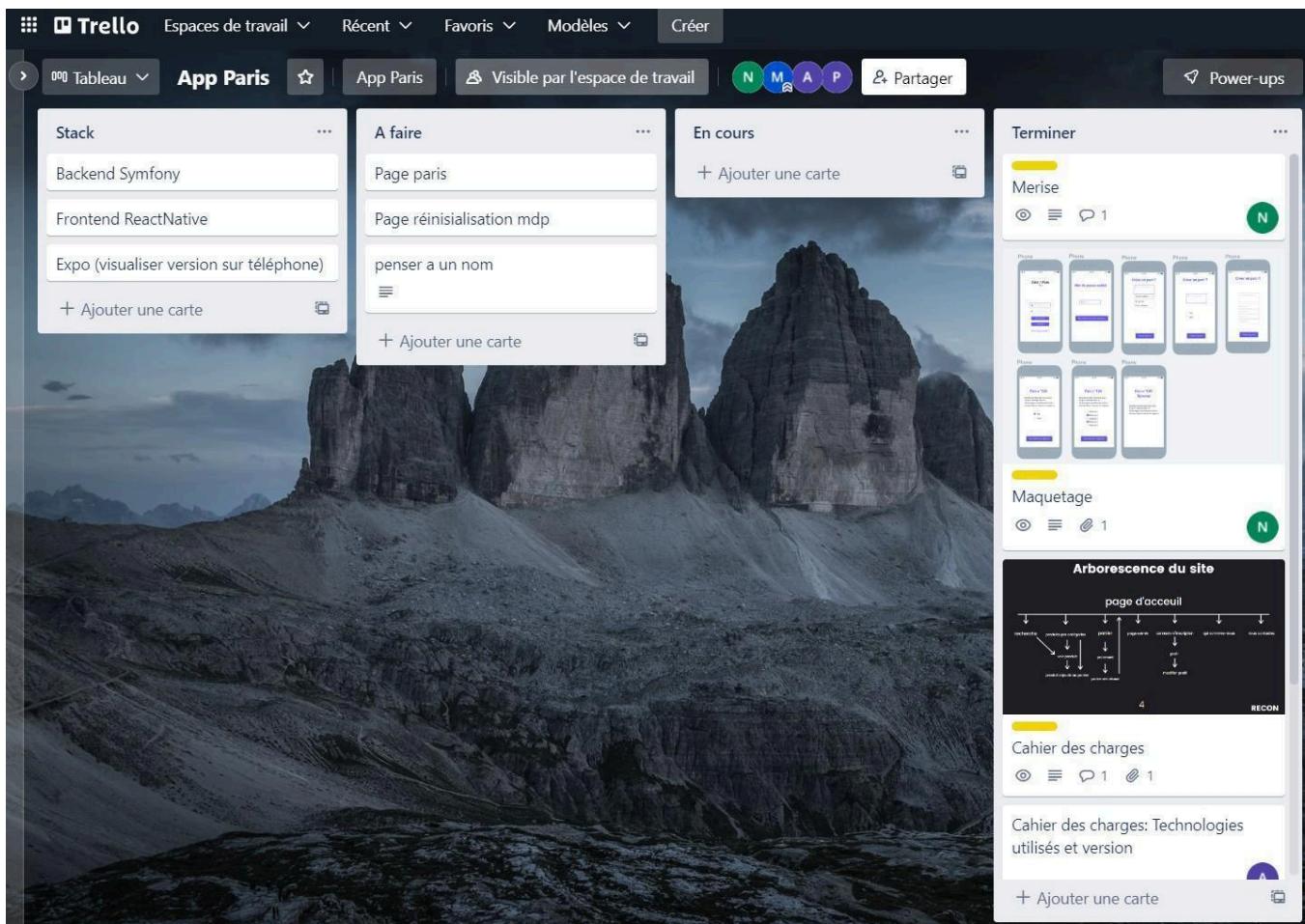
4.2. Organisation

Afin d'optimiser la gestion de projet et de faciliter le travail en équipe, nous avons eu recours à divers outils collaboratifs pour organiser et répartir les tâches de développement.

Pour la gestion des tâches, nous avons choisi d'utiliser **Trello**, un outil de gestion de projet visuel. Trello permet de lister toutes les tâches à réaliser et d'en suivre l'avancement grâce à un système de cartes assignable et déplaçables entre différentes colonnes, reflétant ainsi l'état d'avancement de chaque tâche par les membres de l'équipe.

Dans notre contexte, nous devions composer avec plusieurs contraintes, notamment des délais de réalisation limités par le rythme de l'alternance, ce qui nous laissait peu de temps à consacrer au projet. Pour cette raison, nous avons décidé de ne pas fixer de délais stricts pour la réalisation des tâches.

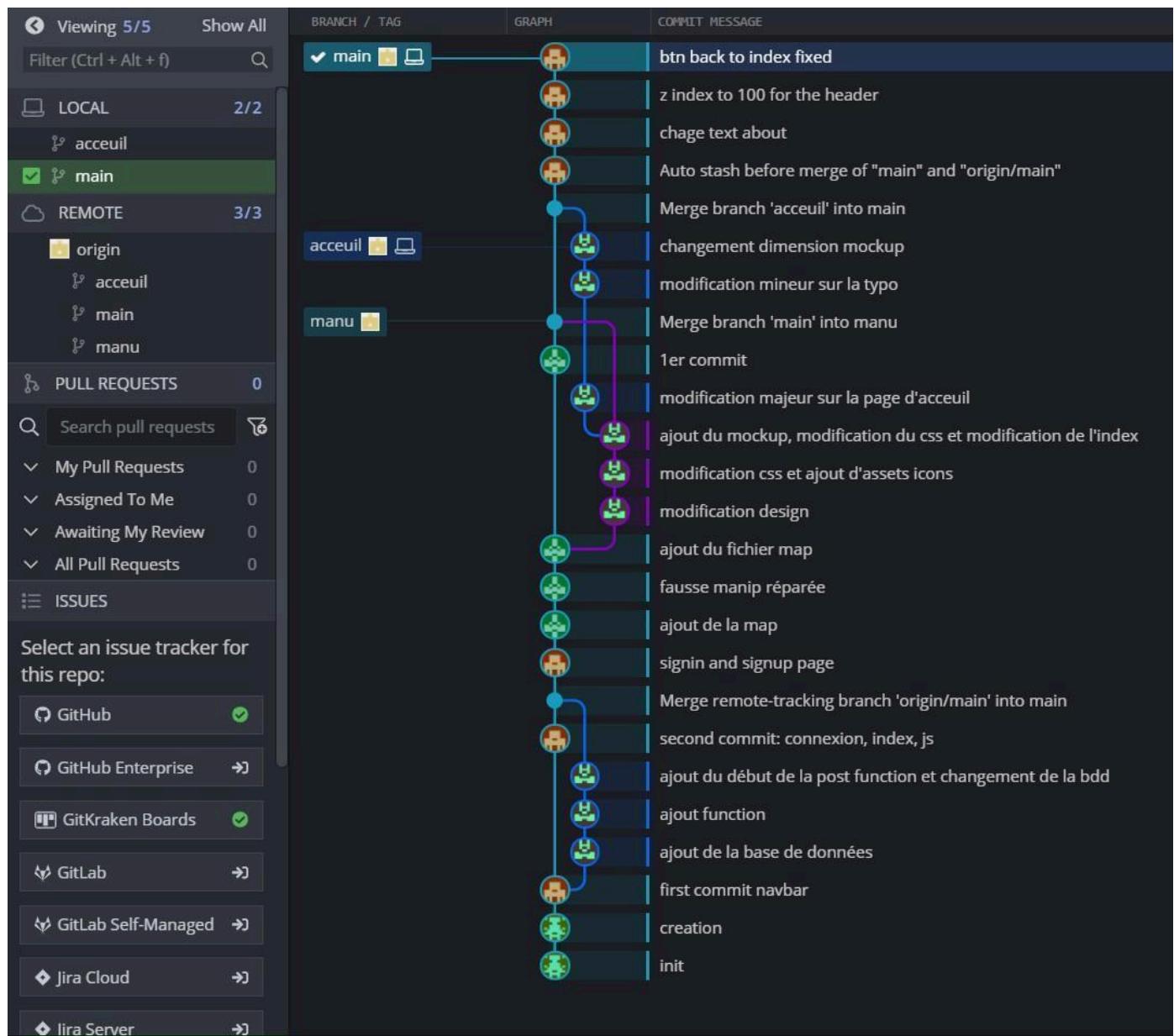
Afin de nous adapter à notre rythme, nous avons adopté une approche inspirée de la méthode **AGILE**, favorisant la collaboration, la flexibilité et l'atteinte progressive de nos objectifs de développement.



4.3. Versionning & Workflow

Pour gérer les différentes modifications du code et assurer la sauvegarde de l'ensemble du travail, nous avons utilisé le logiciel de gestion de versions **GitKraken**, en association avec **GitHub**.

GitKraken propose une interface graphique intuitive qui facilite la gestion et la visualisation des actions réalisées sur Git. Il offre une vue d'ensemble claire des différentes branches, des commits et de l'historique du projet, ce qui simplifie grandement le suivi et la coordination du développement en équipe.



Pour organiser efficacement les différentes parties de l'application, on a mis en place deux repositories distincts :

- **Un repository Backend** : il contient l'API, développée avec Symfony et API Platform.
- **Un repository Frontend** : il regroupe l'interface utilisateur, développée avec Expo React Native.

Sur chacun de ces repositories, nous avons adopté la méthode de travail **GitFlow**.

Cette approche permet de structurer le développement en standardisant la création de branches, et d'organiser le workflow en fonction du type de tâche à réaliser.

Structure des branches avec GitFlow

- **Branches principales :**
 - **main** : représente l'état du projet en production.
- **Branches secondaires :**
 - **feature** : chaque nouvelle fonctionnalité est développée sur une branche dédiée, créée à partir de la version la plus récente de la branche main. Ces branches sont systématiquement préfixées par « feature/ » (exemple : **feature/ajout-pari**).

Cette organisation facilite la gestion du code, le suivi des évolutions et la collaboration entre les membres de l'équipe, tout en assurant la stabilité du projet lors des mises en production.

4.4. Gestion des dépendances

Parmi les différentes fonctionnalités développées, certaines reposent sur l'utilisation de librairies externes. Ces librairies offrent des fonctionnalités déjà implémentées par d'autres développeurs, ce qui nous permet de gagner un temps précieux lors du développement. Voici quelques exemples de librairies utilisées dans le projet :

- **Axios** : un client HTTP qui simplifie la gestion des requêtes vers la base de données.
- **React Navigation** : une librairie permettant le routage et le partage de données entre plusieurs écrans de l'application.

Pour gérer ces dépendances, nous utilisons l'outil **Yarn**. Ce gestionnaire de paquets pour JavaScript permet, à partir d'un fichier "package.json", de lister l'ensemble des modules nécessaires au projet ainsi que leurs versions. Grâce à Yarn, l'installation et la mise à jour des librairies sont facilitées, assurant ainsi la stabilité et la maintenabilité du projet.

```
⚙️ package.json ×
⚙️ package.json > 🚫 private
1  {
2    "name": "app-cotepote",
3    "version": "1.0.0",
4    "main": "node_modules/expo/AppEntry.js",
5    // (débogage)
6    "scripts": {
7      "start": "expo start",
8      "android": "expo start --android",
9      "ios": "expo start --ios",
10     "web": "expo start --web",
11     "eject": "expo eject"
12   },
13   "dependencies": {
14     "@expo-google-fonts/dev": "^0.2.2",
15     "@expo-google-fonts/poppins": "^0.2.2",
16     "@react-native-async-storage/async-storage": "^1.15.17",
17     "@react-native-community/masked-view": "^0.1.11",
18     "@react-navigation/bottom-tabs": "^6.0.9",
19     "@react-navigation/drawer": "^6.3.1",
20     "@react-navigation/native": "^6.0.6",
21     "@react-navigation/stack": "^6.0.11",
22     "axios": "^0.25.0",
23     "buffer": "^6.0.3",
24     "expo": "~44.0.0",
25     "expo-app-loading": "~1.3.0",
26     "expo-constants": "~13.0.1",
27     "expo-font": "~10.0.4",
28     "expo-google-fonts": "^0.0.0",
29     "expo-image-picker": "~12.0.1",
30     "expo-secure-store": "~11.1.0",
31     "expo-status-bar": "~1.2.0",
32     "react": "17.0.1",
33     "react-dom": "17.0.1",
34     "react-native": "0.64.3",
35     "react-native-gesture-handler": "~2.1.0",
36     "react-native-keyboard-aware-scroll-view": "^0.9.5",
37     "react-native-reanimated": "2.3.1",
38     "react-native-safe-area-context": "3.3.2",
39     "react-native-screens": "3.10.1",
40     "react-native-web": "0.17.1",
41     "save": "^2.4.0"
42   },
43   "devDependencies": {
44     ...
45   }
46 }
```

5. Environnement Technique

5.1. Technologies & Outils

5.1.1. Spécificités techniques : Backend

5.1.1.1. API REST

Pour la gestion des données de l'application mobile, nous avons choisi de mettre en place une **API** (Application Programming Interface).

En termes simples, une API est une interface qui permet d'échanger des données de manière réutilisable et standardisée entre différents composants d'une application, ou encore entre une application et des services externes, indépendamment du langage utilisé.

Les données sont généralement organisées en « collections » et peuvent être consultées par le client sous différents formats, comme le JSON.

Il existe plusieurs architectures d'API, parmi lesquelles l'architecture **REST** (Representational State Transfer) est la plus répandue. C'est ce modèle que nous avons retenu pour notre projet.

L'architecture REST exploite les verbes HTTP (GET, POST, PUT, DELETE, etc.), ce qui rend les actions réalisables sur l'API facilement compréhensibles et standardisées. Ce choix facilite ainsi l'intégration, la maintenance et l'évolution de notre application.

5.1.1.2. Symfony / API Platform

Avant de démarrer le développement du back-end, nous avons mené une veille technique pour identifier la solution la plus adaptée à nos contraintes. Notre choix s'est porté sur l'utilisation de **Symfony** couplé à **API Platform**.

API Platform est un framework web open-source, basé sur Symfony, spécifiquement conçu pour la création rapide et efficace d'API RESTful et GraphQL. Il s'appuie sur le modèle MVC (Modèle, Vue, Contrôleur) et permet de générer automatiquement des endpoints d'API à partir des entités métier, tout en respectant les standards modernes comme JSON-LD, OpenAPI ou Hydra. Cette automatisation réduit considérablement le temps de développement et facilite la maintenance de l'API.

La partie serveur d'API Platform est écrite en PHP et repose sur Symfony, tandis que la partie cliente peut être développée en JavaScript ou TypeScript. Un des atouts majeurs d'API Platform est l'intégration native de **Doctrine ORM**, qui simplifie la persistance et l'interrogation des données. Grâce à ce pont, il est possible de bénéficier d'optimisations automatiques des requêtes SQL (par exemple, l'ajout de clauses JOIN

appropriées) et de nombreux filtres puissants pour manipuler les ressources.

Doctrine ORM, via API Platform, prend en charge les principaux SGBD du marché (PostgreSQL, MySQL, MariaDB, SQL Server, Oracle, SQLite, MongoDB ODM).

L'architecture modulaire d'API Platform permet également d'étendre ou de personnaliser facilement les fonctionnalités selon les besoins du projet.

Parmi les autres avantages notables :

- **Documentation interactive générée automatiquement** (Swagger/OpenAPI), facilitant la prise en main par les développeurs et partenaires techniques.
- **Gestion avancée de la sécurité** et des droits d'accès grâce à l'intégration avec les mécanismes de Symfony.
- **Conformité aux standards de l'industrie**, assurant une interopérabilité optimale et une intégration facilitée avec d'autres systèmes.
- **Support de la validation, pagination, tri, filtrage et cache HTTP** directement intégrés.

L'installation et la configuration d'API Platform sont simplifiées grâce à Symfony Flex, permettant une mise en place rapide dans n'importe quel projet Symfony.

En résumé, le choix de Symfony / API Platform s'est imposé pour sa rapidité de développement, sa robustesse, son respect des standards, sa documentation intégrée et sa grande extensibilité, répondant ainsi parfaitement aux besoins de notre projet d'API.

5.1.1.3. Symfony / API Platform : Installation & Configuration

Avant de commencer, il est essentiel de disposer d'un environnement de développement adapté, incluant notamment **WAMP Server** (ou tout autre serveur local) et **Composer**, le gestionnaire de dépendances PHP.

1. Crédit du projet Symfony

Pour démarrer un nouveau projet Symfony, on a exécuté les commandes suivantes dans notre terminal :

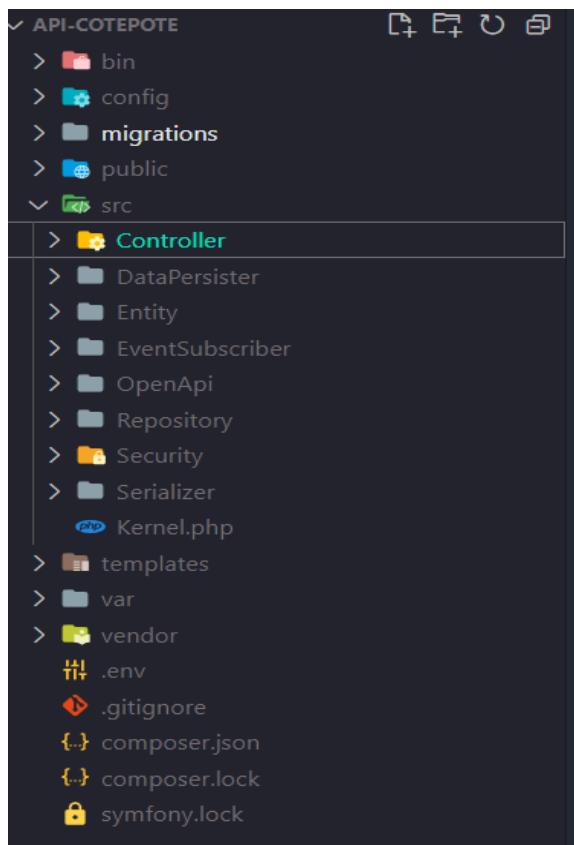
```
symfony new nom-du-projet  
cd nom-du-projet
```

2. Installation d'API Platform

On a ajouté le composant serveur d'API Platform à notre projet Symfony :

```
symfony composer require api
```

une fois ces deux commandes lancés, nous nous retrouvons avec l'architecture suivante :



3. Configuration de la base de données

On a ouvert le fichier `.env` de notre projet et modifiez la ligne `DATABASE_URL` pour indiquer les informations de connexion à votre SGBD (par exemple, MySQL ou PostgreSQL). Exemple pour MySQL :

```
DATABASE_URL="mysql://user:password@127.0.1:3306/nom_de_la_base?serverVersion=8.0.32&charset=utf8mb4"
```

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/apicotepeote?serverVersion=5.7"
```

4. Création de la base de données et du schéma

On a Généré la base de données et son schéma à l'aide des commandes suivantes :

```
symfony console  
doctrine:database:create  
symfony console  
doctrine:schema:create
```

à la suite de ces lignes de commandes, on se retrouve avec notre base de données créée dans notre SGBD :

The screenshot shows the phpMyAdmin interface for the database 'nadir-ziane_cotepote'. The left sidebar lists tables: bet, doctrine_migration_versions, option, refresh_tokens, user, and user_option. The main area displays the table structure for each of these six tables. A summary at the bottom indicates there are 300 rows and a total size of 320,0 kio.

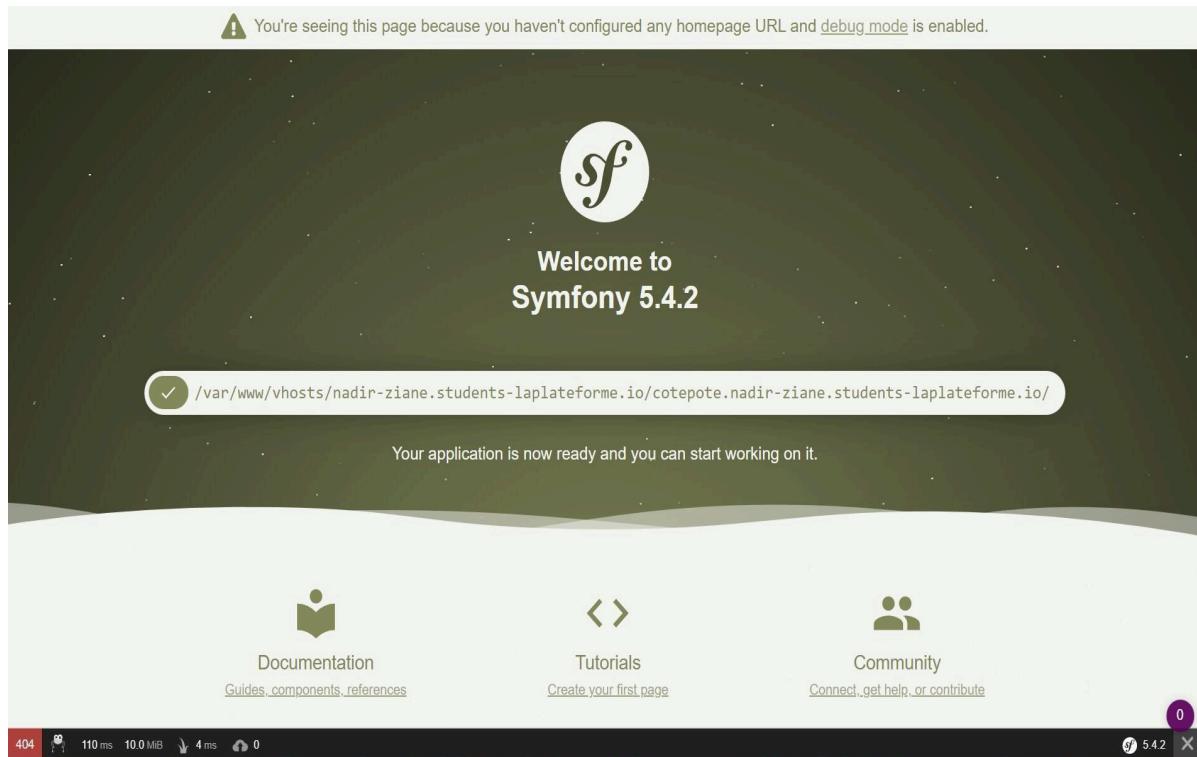
Table	Action	Lignes	Type	Interclassement	Taille	Perte
bet	Parcourir Structure Rechercher Insérer Vider Supprimer	4	InnoDB	utf8mb4_unicode_ci	32,0 kio	-
doctrine_migration_versions	Parcourir Structure Rechercher Insérer Vider Supprimer	13	InnoDB	utf8_unicode_ci	16,0 kio	-
option	Parcourir Structure Rechercher Insérer Vider Supprimer	12	InnoDB	utf8mb4_unicode_ci	32,0 kio	-
refresh_tokens	Parcourir Structure Rechercher Insérer Vider Supprimer	264	InnoDB	utf8mb4_unicode_ci	160,0 kio	-
user	Parcourir Structure Rechercher Insérer Vider Supprimer	7	InnoDB	utf8mb4_unicode_ci	32,0 kio	-
user_option	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_unicode_ci	48,0 kio	0
6 tables	Somme	300	InnoDB	utf8_general_ci	320,0 kio	0

5. Lancement du serveur de développement

On a démarré le serveur PHP intégré pour tester notre application localement :

```
symfony serve
```

On a ouvert ensuite notre navigateur à l'adresse <https://localhost> ou <http://localhost:8000> pour accéder à notre projet Symfony.



6. Accès à la documentation de l'API

API Platform expose automatiquement notre API à l'URL </api/>. Pour consulter la documentation interactive générée (au format OpenAPI/Swagger), rendez-vous sur :

<http://localhost:8000/api/>

On y trouve une interface Swagger UI qui permet de visualiser les endpoints, de consulter les détails des opérations disponibles et même de tester les requêtes directement depuis le navigateur.

The screenshot shows the API Platform interface with the 'Bet' resource selected. At the top, there's a header with the API Platform logo and a '0.0.0 OAS3' button. Below the header, there's a 'Servers' dropdown set to '/' and an 'Authorize' button with a lock icon. The main area displays the Bet resource with its various HTTP methods and their descriptions:

- GET** /api/bets Retrieves the collection of Bet resources.
- POST** /api/bets Creates a Bet resource.
- GET** /api/bets/{id} Retrieves a Bet resource.
- DELETE** /api/bets/{id} Removes the Bet resource.
- PATCH** /api/bets/{id} Updates the Bet resource.
- POST** /api/bets/{id}/image Creates a Bet resource.

7. Gestion des dépendances

Grâce à Composer, toutes les librairies et dépendances utilisées dans notre projet sont listées dans le fichier `composer.json`. On peut facilement ajouter de nouvelles dépendances ou les mettre à jour selon nos besoins.

On peut très facilement les télécharger avec **composer** qui est un logiciel gestionnaire de dépendances libre écrit en **PHP**. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin.

```
composer.json > ...
1  {
2      "type": "project",
3      "license": "proprietary",
4      "minimum-stability": "stable",
5      "prefer-stable": true,
6      "require": {
7          "php": ">=8.0",
8          "ext-ctype": "*",
9          "ext-iconv": "*",
10         "api-platform/core": "^2.6",
11         "doctrine/annotations": "^1.0",
12         "doctrine/doctrine-bundle": "^2.5",
13         "doctrine/doctrine-migrations-bundle": "^3.2",
14         "doctrine/orm": "^2.11",
15         "gesdinet/jwt-refresh-token-bundle": "*",
16         "lexik/jwt-authentication-bundle": "^2.14",
17         "nelmio/cors-bundle": "v2.2",
18         "phpdocumentor/reflection-docblock": "^5.3",
19         "phpstan/phpdoc-parser": "v1.2",
20         "ramsey/uuid": "4.2",
21         "symfony/asset": "5.4.*",
22         "symfony/console": "5.4.*",
23         "symfony/dotenv": "5.4.*",
24         "symfony/expression-language": "5.4.*",
25         "symfony/flex": "v1.17|^2",
26         "symfony/framework-bundle": "5.4.*",
27         "symfony/property-access": "5.4.*",
28         "symfony/property-info": "5.4.*",
29         "symfony/proxy-manager-bridge": "5.4.*",
30         "symfony/runtime": "5.4.*",
31         "symfony/security-bundle": "5.4.*",
32         "symfony/serializer": "5.4.*",
33         "symfony/twig-bundle": "5.4.*",
34         "symfony/validator": "5.4.*",
35         "symfony/yaml": "5.4.*",
36         "vich/uploader-bundle": "v1.19"
37     },
38     "config": {
39         "allow-plugins": [
40             "composer/package-versions-deprecated": true,
41             "symfony/flex": true,
42             "symfony/runtime": true
43         ]
44     }
45 }
```

5.1.1.4. Crédation d'une Collection avec Symfony / API Platform

Une fois l'étape de la mise en place de notre projet en Symfony - Api Platform réalisé, la première étape que l'on a effectué est la création des entités, c'est là où Doctrine ORM rentre en jeu :

Doctrine facilite grandement la création des entités ainsi que des champs, il prend tout en compte pour la création / modélisation de notre base de données.

A l'aide de quelques lignes de commande on crée notre modèle de données et on gère la persistance (la sauvegarde) dans une table.

6. Crédation de l'entité avec Doctrine

La première étape consiste à créer une entité, qui correspondra à une table dans la base de données et à une ressource exposée par l'API. Doctrine ORM facilite cette opération grâce à la commande suivante:

```
php bin/console make:entity
```

Cette ligne de commande nous sert à créer une entité

Le terminal nous invite alors à :

- Choisir le nom de l'entité (par exemple, Pari)
- Définir les différents champs (type, longueur, nullable, etc.)

```
Class name of the entity to create or update (e.g.  
AgreeableJellybean):  
> Product  
created: src/Entity/Product.php  
created: src/Repository/ProductRepository.php  
Entity generated! Now let's add some fields!  
You can always add more fields later manually or by re-running this  
command.
```

La création des champs, se fait en quelques étapes :

```
New property name (press <return> to stop adding fields):  
> name  
Field type (enter ? to see all types) [string]:  
> string  
Field length [255]:  
>  
Can this field be null in the database (nullable) (yes/no) [no]:  
>  
updated: src/Entity/Product.php
```

tout d'abord le type (si c'est une string,, un booléen, un datetime)
dans notre cas c'est une string et par là suite on doit choisir la longueur de
la chaîne de caractère et enfin si elle est nul.

On peut répéter l'opération pour chaque attribut nécessaire à notre entité.

1. Migration de la base de données

Une fois l'entité créée, il faut générer une migration puis l'exécuter pour mettre à jour
la base de données :

```
php bin/console make:migration
```

Cette migration est possible que si dans le fichier .env, il y a la chaîne de
connexion bien remplie avec les identifiant et mot de passe, nom de la base
de donnée et du host.

Il suffit de lancer la ligne de commande suivante :

```
php bin/console  
doctrine:migrations:migrate
```

```

php bin/console doctrine:migrations:migrate
WARNING! You are about to execute a database migration that could
result in schema changes and data loss. Are you sure you wish to
continue? (y/n)y
Migrating up to 20190612091941 from 0

++ migrating 20190612091941

    -> CREATE TABLE product (id INT AUTO_INCREMENT NOT NULL, name
VARCHAR(255) NOT NULL, price INT NOT NULL, PRIMARY KEY(id)) DEFAULT
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci ENGINE = InnoDB

++ migrated (took 128.9ms, used 12M memory)

-----
-- 
++ finished in 133.7ms
++ used 12M memory
++ 1 migrations executed
++ 1 sql queries

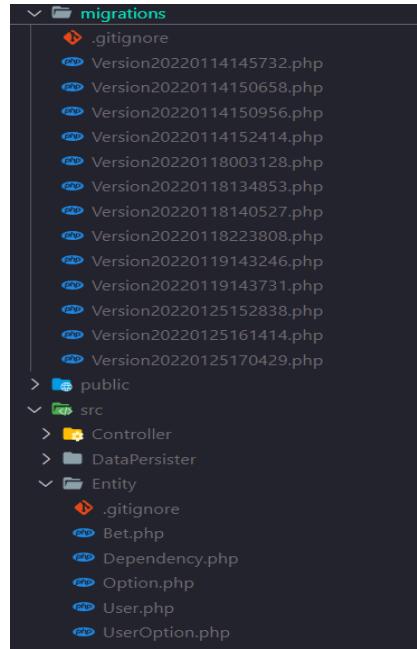
```

Cela crée la table correspondante dans notre SGBD et enregistre la version de la migration.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus
2	email	varchar(180)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
3	roles	longtext	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
4	password	longtext	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
5	name	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
6	surname	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
7	file_path	longtext	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
8	updated_at	datetime			Oui	NULL			Modifier Supprimer Plus

version				
<input type="checkbox"/>	Éditer	Copier	Supprimer	DoctrineMigrations\Version20220114145732
<input type="checkbox"/>	Éditer	Copier	Supprimer	DoctrineMigrations\Version20220114150658
<input type="checkbox"/>	Éditer	Copier	Supprimer	DoctrineMigrations\Version20220114150956
<input type="checkbox"/>	Éditer	Copier	Supprimer	DoctrineMigrations\Version20220114152414

Une fois toutes ces étapes réalisées, on voit que dans l'architecture des fichiers sont mis également à jour, un fichier s'ajoute dans le dossier migrations et également un fichier sera ajouter dans le dossier pour les entités :



2. Exposition de l'entité en tant que ressource API

API Platform détecte automatiquement les entités annotées avec `@ApiResource` ou `##[ApiResource()]` et les expose comme collections REST. Par défaut, l'API propose les opérations CRUD de base (GET, POST, PUT, PATCH, DELETE) sur la collection et sur chaque élément.

```

<?php

namespace App\Entity;

use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Serializer\Annotation\Groups;

/**
 * @ApiResource(
 *   normalizationContext={"groups":{"pari:read"}},
 *   denormalizationContext={"groups":{"pari:write"}}
 *)
 * @ORM\Entity()
 */
class Pari
{
    // ...
    /**
     * @ORM\Column(type="string")
     * @Groups({"pari:read", "pari:write"})
     */
    private $titre;
    // autres champs...
}

```

3. Groupes de sérialisation

Les groupes de sérialisation (@Groups) permettent de contrôler quelles propriétés sont exposées lors de la lecture (normalization) ou de l'écriture (denormalization) via l'API.

On définit les groupes dans l'annotation @ApiResource et on les associe aux propriétés concernées.

Globalement, ce qui va nous intéresser c'est le fichier qui sera ajouter dans le dossier entité :

C'est tout simplement la classe qui va gérer toute notre table et depuis laquelle on va pouvoir ajouter des annotations pour configurer les routes API

Cela va comprendre majoritairement le type de sérialisation et désérialisation que l'on va choisir :

La sérialisation est ce que notre api va retourner, et la désérialisation est ce que l'on envoie à notre API.:

En ce qui concerne le contexte de dénormalisation, cela va définir quelle données on souhaite intercepter de notre API, cela va se faire à travers les annotations dans la classe :

via ce type d'écriture commenter, API Platform va tout simplement lire les annotations au sein de “@ApiResource(...)" pour voir les règles concernant les routes API, ce qui va principalement définir les règles de normalisation ainsi que de dénormalisation des routes sont les deux suivantes :

```
/**
 * @ORM\Entity(repositoryClass=BetRepository::class)
 * @ApiResource(
 *     normalizationContext = {"groups" = {"read:bet"}},
 *     denormalizationContext = {"groups" = {"create:bet"}}
 * )
```

À l'intérieur, on va tout simplement dire quel groupe de lecture on affecte à l'un et à l'autre pour savoir quels éléments sont concernés dans la base de données.

Ces noms de groupes, on n'a plus qu'à les rappeler au dessus des attributs dans la classe pour que l'on décide lesquels seront concernés, la manière est la suivante :

```
class Bet
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     * @Groups({"read:bet"})
     */
    private $id;

    /**
     * @ORM\Column(type="string", Length=255)
     * @Groups({"read:bet"})
     */
    private $uuid;

    /**
     * @ORM\Column(type="string", length=255)
     * @Groups({"read:bet", "create:bet"})
     * @Assert\Length(
     *     min = 5,
     *     max = 255
     * )
     */
    private $title;
```

on met dans l'annotation “@Groups()” l'attribut concerné pour que l'on puisse le retrouver dans le contexte de normalisation, ou celui de dénormalisation.

4. Visualisation et test dans Swagger UI

Une fois la migration effectuée et l'entité exposée, API Platform génère automatiquement la documentation interactive (Swagger UI) accessible à l'adresse :

<http://localhost:8000/api/>

Vous y retrouvez toutes les routes CRUD générées pour la nouvelle collection, et vous pouvez tester les opérations (GET, POST, etc.) directement depuis l'interface.

The screenshot shows the API Platform interface with the following details:

- Servers:** /
- Authorize:** button with a lock icon.
- Bet** resource:
 - GET /api/bets**: Retrieves the collection of Bet resources.
 - POST /api/bets**: Creates a Bet resource.
 - GET /api/bets/{id}**: Retrieves a Bet resource.
 - DELETE /api/bets/{id}**: Removes the Bet resource.
 - PATCH /api/bets/{id}**: Updates the Bet resource.
 - POST /api/bets/{id}/image**: Creates a Bet resource.
 - POST /api/bets/{id}/setExpired**: Permet de set un pari en expiré.

6.1. Symfony / API Platform : Ajout des relations

Pour ajouter des relations (cardinalités) entre nos collections, Doctrine offre la possibilité de les paramétrier lors de la création des tables. Cela permet d'établir les liens entre les modèles en choisissant le type de cardinalité correspondant au MCD.

Dans le cas d'une relation ManyToOne, c'est toujours l'entité du côté « Many » qui doit définir la relation.

La création de cette relation se fait via des lignes de commande. On commence par indiquer l'entité à modifier, puis le nom du champ à ajouter. Ensuite, on précise qu'il s'agit d'une relation de type ManyToOne, on définit le nom de la propriété, on indique si le champ peut être nul, et on choisit éventuellement d'ajouter une propriété inverse dans l'autre entité. Enfin, il est possible de permettre la suppression en cascade, c'est-à-dire que si un utilisateur est supprimé, tous les paris qui lui sont associés le seront également.

```

Field type (enter ? to see all types) [string]:
> ManyToOne

What class should this entity be related to?
> Article

Is the Comment.article property allowed to be null (nullable)? (yes/no) [yes]:
> no

Do you want to add a new property to Article so that you can access/update Comment objects from it - e.g. $article->getComments()? (yes/no) [yes]:
> yes

A new property will also be added to the Article class so that you can access the related Comment objects from it.

New field name inside Article [comments]:
>

Do you want to activate orphanRemoval on your relationship?
A Comment is "orphaned" when it is removed from its related Article.
e.g. $article->removeComment($comment)

NOTE: If a Comment may *change* from one Article to another, answer "no".

Do you want to automatically delete orphaned App\Entity\Comment objects (orphanRemoval)? (yes/no) [no]:
> yes

updated: src/Entity/Comment.php
updated: src/Entity/Article.php

```

illustrent par un exemple concret de relation ManyToOne entre les entités Bet (pari) et User :

Ajout d'une relation ManyToOne avec Doctrine et API Platform

1. Crédation de la relation dans l'entité

Dans une relation ManyToOne, c'est toujours l'entité du côté « Many » (ici, Bet) qui porte la clé étrangère et donc la définition de la relation. Cela signifie que chaque Bet est associé à un seul User, tandis qu'un User peut avoir plusieurs Bet.

On génère cette relation via la ligne de commande suivante :

```
php bin/console make:entity
```

On sélectionne l'entité à modifier (Bet)

On ajoute un nouveau champ (par exemple, user)

On indique que ce champ est une relation vers l'entité User de type ManyToOne

On précise si le champ peut être nul ou non, et si on veut autoriser la suppression en cascade

Cela génère automatiquement le code suivant dans Bet.php :

```
php
// src/Entity/Bet.php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use ApiPlatform\Metadata\ApiResource;

#[ORM\Entity]
#[ApiResource]
class Bet
{
    // ...

    #[ORM\ManyToOne(targetEntity:
User::class, inversedBy: 'bets', cascade:
['remove'])]
    private ?User $user = null;

    // getters et setters...
}
```

Et dans l'entité User, on aura :

```
php
// src/Entity/User.php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use
Doctrine\Common\Collections\ArrayCollecti
on;
use
Doctrine\Common\Collections\Collection;

#[ORM\Entity]
class User
{
    // ...

    #[ORM\OneToMany(mappedBy: 'user',
targetEntity: Bet::class)]
    private Collection $bets;

    public function __construct()
    {
        $this->bets = new ArrayCollection();
    }

    // getters et setters...
}
```

2. Migration de la base de données

Après avoir ajouté la relation, on génère et applique la migration :

```
bash
php bin/console
make:migration
php bin/console
doctrine:migrations:migrate
```

3. Test et visualisation via l'API

API Platform expose automatiquement la relation dans la documentation Swagger à /api/.

On pourra ainsi créer un Bet en associant un User via son IRI (identifiant de ressource), et retrouver la relation dans les réponses de l'API.

6.1.1. Spécificités techniques : Frontend

6.1.1.1. React Native

Aujourd'hui, il existe plusieurs approches pour développer des applications mobiles.

Les **applications natives** sont conçues spécifiquement pour un système d'exploitation donné (iOS ou Android), en utilisant les langages propres à chaque plateforme, comme Swift pour iOS ou Kotlin/Java pour Android. Cette méthode garantit la meilleure performance et un accès complet aux fonctionnalités matérielles du téléphone, mais elle implique de développer et de maintenir une base de code distincte pour chaque OS, ce qui augmente les coûts et les délais de développement.

À l'opposé, le **développement cross-platform** permet de créer une application unique, compatible avec plusieurs systèmes d'exploitation à partir d'une seule base de code. Des frameworks comme Ionic, Cordova, Flutter ou React Native facilitent cette approche. Elle offre un gain de productivité important et réduit les coûts, car il n'est plus nécessaire de dupliquer le travail pour chaque plateforme. Cependant, les applications cross-platform traditionnelles peuvent présenter des limites : elles n'accèdent pas toujours à toutes les fonctionnalités natives du téléphone et leurs performances peuvent être inférieures à celles des applications natives.

Pour combiner les avantages des deux mondes, nous avons choisi de développer "Côté Pote" avec **Expo React Native**. Expo est une plateforme open source qui s'appuie sur React Native et permet de créer des applications mobiles iOS et Android à partir d'un même code JavaScript ou TypeScript, sans avoir à manipuler

directement les couches natives. Expo propose un environnement de développement géré, des outils de prévisualisation en temps réel et une bibliothèque riche de composants et d'API, ce qui accélère le développement et simplifie la maintenance.

Contrairement à d'autres solutions cross-platform, **Expo React Native permet de générer des applications réellement natives**, qui utilisent les mêmes API que celles développées avec Xcode ou Android Studio. Cela garantit une expérience utilisateur fluide et permet d'accéder à de nombreuses fonctionnalités des appareils mobiles, tout en conservant la rapidité et la simplicité d'un développement multiplateforme.

En résumé, ce choix technologique nous permet de maximiser la productivité, de réduire la complexité et d'offrir une expérience utilisateur de qualité, sans avoir à apprendre plusieurs langages natifs ni sacrifier l'accès aux fonctionnalités essentielles des smartphones.

6.1.1.2. Expo

Pour mettre en place un environnement de développement React Native — comme pour la plupart des frameworks front-end modernes — il est indispensable d'installer **Node.js**, une plateforme JavaScript orientée vers les applications réseau asynchrones. Node.js s'accompagne de **npm** (Node Package Manager), l'outil incontournable pour gérer, installer et publier des bibliothèques JavaScript au sein de l'écosystème [Node.js](#).

Pour créer un projet React Native, nous avons choisi d'utiliser la librairie **Expo CLI**. Expo fournit un SDK complet ainsi qu'un ensemble d'outils et de services facilitant la création, le test et le déploiement d'applications mobiles avec React Native.

Expo permet, par exemple, grâce à son utilitaire de développement « watchman », de surveiller les fichiers du projet et de recompiler le code à la volée. On peut ainsi visualiser instantanément les modifications sur différents émulateurs iOS/Android ou directement sur un appareil mobile, ce qui améliore considérablement le confort de développement.

Étapes de création d'un projet Expo React Native :

1. Pré-requis :

On installe Node.js (version LTS recommandée) et npm sur notre machine.

2. Initialisation du projet :

On exécute la commande suivante pour créer un nouveau projet Expo :

```
bash  
npx create-expo-app my-app
```

ou avec Yarn :

```
bash  
yarn create expo-app my-app
```

Cette commande génère la structure de base du projet et installe toutes les dépendances nécessaires.

3. Démarrage du serveur de développement :

On se rend dans le dossier du projet, puis on lance :

```
bash  
npm start
```

ou

```
bash  
npx expo start
```

Cela démarre le serveur de développement Expo, qui affiche un QR code à scanner avec l'application Expo Go sur votre smartphone pour tester l'application en temps réel.

4. Gestion des dépendances :

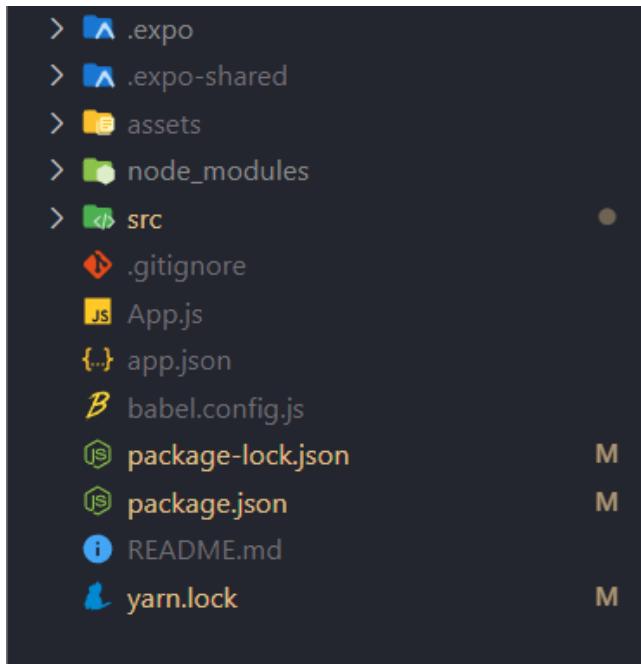
Toutes les librairies et dépendances du projet sont listées dans le fichier `package.json`. Grâce à npm ou Yarn, il est facile d'installer, de mettre à jour ou de supprimer des

modules selon les besoins du projet.

En résumé :

L'utilisation de Node.js, npm et Expo CLI permet de démarrer rapidement un projet React Native, de gérer efficacement les dépendances, et de profiter d'un environnement de développement moderne, multiplateforme et évolutif.

Une fois cette opération faite, on se retrouve avec une architecture de ce genre :



```
> 📂 .expo
> 📂 .expo-shared
> 📁 assets
> 📂 node_modules
> 📁 src
  ⚡ .gitignore
  🎨 App.js
  {..} app.json
  ⚡ babel.config.js
  🎯 package-lock.json          M
  🎯 package.json               M
  🗂 README.md
  🎣 yarn.lock                 M
```

Ce qui est pratique une fois de plus avec ce type de framework, on peut aisément utiliser et télécharger des librairies pour pouvoir s'en servir par la suite sur notre projet, on peut retrouver simplement dans le fichier "composer.json" toutes les librairies et dépendances que l'on a sur notre projet :

```
② package.json > 49 private
1  {
2    "name": "app-cotepote",
3    "version": "1.0.0",
4    "main": "node_modules/expo/AppEntry.js",
5    "scripts": {
6      "start": "expo start",
7      "android": "expo start --android",
8      "ios": "expo start --ios",
9      "web": "expo start --web",
10     "eject": "expo eject"
11   },
12   "dependencies": {
13     "@expo-google-fonts/dev": "^0.2.2",
14     "@expo-google-fonts/poppins": "^0.2.2",
15     "@react-native-async-storage/async-storage": "1.15.17",
16     "@react-native-community/masked-view": "0.1.11",
17     "@react-navigation/bottom-tabs": "6.0.9",
18     "@react-navigation/drawer": "6.3.1",
19     "@react-navigation/native": "6.0.6",
20     "@react-navigation/stack": "6.0.11",
21     "axios": "0.25.0",
22     "buffer": "6.0.3",
23     "expo": "44.0.0",
24     "expo-app-loading": "1.3.0",
25     "expo-constants": "13.0.1",
26     "expo-font": "10.0.4",
27     "expo-google-fonts": "0.0.8",
28     "expo-image-picker": "12.0.1",
29     "expo-secure-store": "11.1.0",
30     "expo-status-bar": "1.2.0",
31     "react": "17.0.1",
32     "react-dom": "17.0.1",
33     "react-native": "0.64.3",
34     "react-native-gesture-handler": "2.1.0",
35     "react-native-keyboard-aware-scroll-view": "0.9.5",
36     "react-native-reanimated": "2.3.1",
37     "react-native-safe-area-context": "3.3.2",
38     "react-native-screens": "3.10.1",
39     "react-native-web": "0.17.1",
40     "save": "2.4.0"
41   },
42 }
```

6.1.1.3. React Navigation

La navigation entre les différentes vues dans une application React Native est très différente de celle d'un site web classique, où l'on utilise des liens "href" pour passer d'une page à l'autre. Dans React Native, on utilise la bibliothèque populaire **React Navigation** pour gérer le routage et la navigation entre les écrans, ainsi que le partage de données entre eux.

Pour naviguer entre les écrans, on utilise le **StackNavigator**, qui fonctionne comme une pile : chaque nouvel écran est ajouté au sommet de la pile, et lorsque l'on appuie sur le bouton "Retour", les écrans sont retirés du haut de la pile.

Pour installer React Navigation, il suffit d'exécuter la commande suivante dans le terminal :

```
bash
npm install
@react-navigation/native
```

Ensuite, on installe également le module `@react-navigation/stack` et ses dépendances pour gérer la navigation en pile.

Dans l'application Côté Pote, la navigation principale se fait grâce à une **TabBar** située en bas de l'écran, qui permet de passer facilement d'un écran principal à un autre. Pour que ces composants fonctionnent, ils doivent être placés à l'intérieur du composant `<NavigationContainer>`, fourni par React Navigation, que l'on place généralement à la racine de l'application dans le fichier `App.js`.

Le `<NavigationContainer>` gère l'ensemble de l'arbre de navigation et les différents états de navigation, tout en offrant des fonctionnalités adaptées à chaque plateforme.

Après avoir mis en place le `<NavigationContainer>`, on utilise la fonction `createBottomTabNavigator` pour créer la TabBar. Cette fonction retourne un objet contenant les composants `Navigator` et `Screen`. Chaque `<Tab.Screen>` correspond à un onglet en bas de l'écran, permettant de changer d'écran principal.

Tous les `<Tab.Screen>` doivent être inclus dans un composant parent `<Tab.Navigator>`, avec au minimum deux propriétés :

- `component` : le composant fonctionnel à afficher pour cet écran.
- `name` : le nom utilisé pour naviguer vers cet écran.

```

const Tab = createBottomTabNavigator();

const Tabs = () => [
  const [pic, setPic] = useState("");
  const profilePic = async () => {
    let mag = await UserInfoService.getAllUserInfo();
    if (mag) {
      setPic(JSON.parse(mag).fileurl);
    } else {
      return;
    }
  };
]

profilePic();

return (
  <Tab.Navigator
    screenOptions={{
      headerShown: false,
      tabBarStyle: {
        backgroundColor: "black",
        height: 100,
        borderTopColor: "black",
        ...styles.tab,
      },
      tabBarShowLabel: false,
    }}
  >
  <Tab.Screen
    name="Home"
    component={AcceuilScreen}
    options={{
      tabBarIcon: ({ focused }) => (
        <View
          style={{ alignItems: "center", justifyContent: "center", top: 5 }}
        >
          <Image
            source={require("../assets/icons/home-icon.png")}
            resizeMode="contain"
            style={{
              width: 40,
              height: 40,
              tintColor: focused ? "white" : "grey",
            }}
          />
        </View>
      ),
    }}
  />
)

```

Ensuite, pour chaque onglet, on peut créer des écrans supplémentaires à l'aide de la fonction `createNativeStackNavigator`. Cela permet de créer une pile d'écrans à l'intérieur d'un onglet et de gérer la navigation entre eux grâce aux fonctionnalités événementielles fournies par React Native.

```

import React, { useEffect } from "react";
import "react-native-gesture-handler";
import { NavigationContainer } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/stack";
import NewBetScreen from "../screens/NewBetScreen";
import CreateBetScreen from "../screens/CreateBetScreen";
import SearchBetScreen from "../screens/SearchBetScreen";

const { Navigator, Screen } = createStackNavigator();

const BetNavigator = () => {
  return (
    // <NavigationContainer>
    <Navigator screenOptions={{ headerShown: false }}>
      <Screen name="CreateBet" component={CreateBetScreen} />
      <Screen name="SearchBetById" component={SearchBetScreen} />
      <Screen name="NewBet" component={NewBetScreen} />
    </Navigator>
    // </NavigationContainer>
  );
};

export { BetNavigator };

```

6.1.1.4. Axios : Interaction avec la base de données

Présentation d'Axios

Axios est une librairie JavaScript permettant de réaliser des requêtes HTTP de manière simple et efficace, notamment pour interagir avec des bases de données via des API. Elle fonctionne aussi bien côté client (navigateur) que côté serveur (Node.js), et repose sur le principe des Promesses pour gérer les réponses asynchrones.

Avantages d'Axios

- Gestion simplifiée des requêtes HTTP** : Axios propose des méthodes dédiées pour chaque verbe HTTP (`get`, `post`, `put`, `delete`, etc.), ce qui facilite l'écriture et la lecture du code.
- Support natif des réponses** : Contrairement à l'API `fetch`, Axios gère automatiquement la conversion des réponses JSON et permet d'exploiter directement les données une fois la Promesse résolue, sans étape de parsing supplémentaire.
- Gestion centralisée des erreurs et des en-têtes** : Axios offre une gestion

plus intuitive des erreurs réseau et permet de configurer facilement les en-têtes, l'authentification, ou encore les paramètres globaux des requêtes.

Exemple d'utilisation

Pour effectuer une requête avec Axios, il faut d'abord importer le module dans le fichier concerné :

```
javascript
import axios from
```

Requête GET (lecture de données)

```
javascript
axios.get('https://api.monsite.com/ressource')
  .then(response => {
    // Les données sont accessibles via
    response.data
    console.log(response.data);
  })
  .catch(error => {
    // Gestion des erreurs
    console.error(error);
  });
});
```

Dans cet exemple, la méthode get prend en paramètre l'URL de la ressource à interroger.

Requête POST (écriture de données)

```

javascript
axios.post('https://api.monsite.com/ressource',
{
  nom: 'Exemple',
  valeur: 42
})
.then(response => {
  console.log(response.data);
})
.catch(error => {
  console.error(error);
});

```

Ici, la méthode post prend en paramètre l’URL de la ressource et un objet représentant le corps de la requête (body), qui sera envoyé au serveur.

Résumé des méthodes principales

Méthode Axios	Description	Paramètres principaux
axios.get	Récupérer des données	URL, options
axios.post	Créer/ajouter des données	URL, données à envoyer, options
axios.put	Modifier/remplacer des données	URL, nouvelles données, options
axios.delete	Supprimer des données	URL, options

L’utilisation d’Axios permet d’interagir facilement avec une API pour lire, écrire,

modifier ou supprimer des données, tout en bénéficiant d'une gestion simplifiée des réponses et des erreurs.

: <https://blog.logrocket.com/axios-vs-fetch-best-http-requests>

```
await axios
  .get(
    "https://cotepte.nadir-ziane.students-laplateforme.io/api/users/" + id,
  {
    headers: {
      "Content-type": "application/json",
    },
  }
)
.then(async (response) => {
  if (response.data.id) {
    console.log(response.data.fileurl);
    let fileurl = response.data.fileurl;
    setPic(
      "https://cotepte.nadir-ziane.students-laplateforme.io" + fileurl
    );
    console.log(pic);
  } else {
    console.log("erreur, pas de retour de l'api");
  }
})
.catch(async (e) => {
  console.log(e);
});
```

7. Présentation du jeu d'essai

7.1. Parcours utilisateur pour la connexion d'un user

Je vais vous présenter des extraits de code illustrant les différentes étapes du processus de connexion d'un utilisateur.

Le point d'entrée de l'application se situe dans le fichier **App.js**. C'est à partir de ce composant principal que s'effectue le rendu conditionnel de la Tab Bar ou de la stack de navigation, en fonction de l'état d'authentification de l'utilisateur.

- **Si l'utilisateur est connecté**, l'application affiche la Tab Bar, permettant d'accéder aux différentes sections principales.
- **Si l'utilisateur n'est pas authentifié**, une stack de navigation spécifique s'affiche, composée de trois vues dédiées au parcours de connexion.

L'onglet « Connexion » est alors le premier écran de la pile, servant de point de départ au processus d'authentification de l'utilisateur.

```
return (
  <AuthContext.Provider value={authContext}>
    {state.userToken !== null ? (
      <>
        <NavigationContainer>
          <Navigator screenOptions={{ headerShown: false }}>
            <Screen name="Tab" component={Tabs} />
          </Navigator>
        </NavigationContainer>
      </>
    ) : (
      <>
        <NavigationContainer>
          <Navigator screenOptions={{ headerShown: false }}>
            <Screen name="Connexion" component={ConnexionScreen} />
            <Screen name="Inscription" component={InscriptionScreen} />
            <Screen name="Loading" component={LoadingScreen} />
          </Navigator>
        </NavigationContainer>
      </>
    )
  }
</AuthContext.Provider>
);
```

Cette page, ainsi que les suivantes, sont des enfants d'une stack de navigation créée à l'aide de la fonction `createStackNavigator` dans l'application. Lors de l'utilisation de cette fonction, deux composants principaux sont mis à disposition : **Navigator** et **Screen**. J'importe également le composant `NavigationContainer`, qui englobe le `Navigator` et tous les écrans (`Screen`) de la stack.

Chaque composant `<Screen>` possède plusieurs propriétés, dont deux essentielles :

- **name** : qui permet d'identifier l'écran et de naviguer vers celui-ci.
- **component** : qui spécifie le composant React à afficher à l'écran.

Par défaut, l'écran affiché en premier est l'écran de **Connexion**. Ce dernier présente les champs de saisie pour l'email et le mot de passe. Les valeurs saisies dans ces champs seront utilisées pour effectuer un appel API, afin de vérifier les identifiants de l'utilisateur auprès du backend



```

const authContext = React.useMemo(
() => ({
  signIn: async (data) => {
    let token;
    await axios
      .post(
        "https://cotepote.nadir-ziane.students-laplateforme.io/api/login",
        {
          username: data.login,
          password: data.password,
        },
        {
          headers: {
            "Content-type": "application/json",
          },
        }
      )
      .then(async (response) => {
        if (response.data?.token) {
          await SecureStore.setItemAsync("token", `${response.data.token}`);
          token = response.data.token;
          await SecureStore.setItemAsync("data", `${response.data}`);
          UserService.getUser();
          await SecureStore.setItemAsync("isOk", "1");
        } else {
        }
      })
      .catch(async (e) => {
        console.log(e);
        await SecureStore.setItemAsync("error", "pas connecté");
        Alert.alert("Erreur", "Identifiant ou mot de passe incorrect", [
          {
            text: "OK",
          },
        ]);
      });
    dispatch({ type: "SIGN_IN", token: token });
  },
}),

```

J'effectue cette requête à l'intérieur d'un hook useMemo, ce qui me permet de mémoriser et de renvoyer une valeur optimisée. J'utilise ensuite cette valeur comme paramètre pour la méthode React.createContext().

En enveloppant tous mes composants avec ce contexte, je rends accessibles, de manière globale, les données, méthodes ou variables nécessaires à l'application. Ainsi, je n'ai plus à me soucier du passage de données entre composants parents et enfants, ce qui simplifie grandement le partage d'informations et de fonctionnalités à travers toute l'application.

```

import * as React from 'react';

const AuthContext = React.createContext();

export default AuthContext;

```

```

return (
  <AuthContext.Provider value={authContext}>
    {state.userToken !== null ? (
      <>
        <NavigationContainer>
          <Navigator screenOptions={{ headerShown: false }}>
            <Screen name="Tab" component={Tabs} />
          </Navigator>
        </NavigationContainer>
      </>
    ) : (
      <>
        <NavigationContainer>
          <Navigator screenOptions={{ headerShown: false }}>
            <Screen name="Connexion" component={ConnexionScreen} />
            <Screen name="Inscription" component={InscriptionScreen} />
            <Screen name="Loading" component={LoadingScreen} />
          </Navigator>
        </NavigationContainer>
      </>
    )}
  </AuthContext.Provider>
);

```

Stockage sécurisé avec SecureStore

Pour stocker les données sensibles sur le mobile de l'utilisateur, j'utilise la librairie **SecureStore** d'Expo. Cette bibliothèque permet d'enregistrer des paires clé/valeur de manière chiffrée et sécurisée, aussi bien sur Android que sur iOS. Les principales méthodes sont :

- `setItemAsync(key, value)` pour enregistrer une donnée
- `getItemAsync(key)` pour la récupérer,
- `deleteItemAsync(key)` pour la supprimer.

Le stockage se fait de façon asynchrone et chaque valeur stockée est chiffrée localement, ce qui garantit la confidentialité des informations utilisateurs, comme les tokens d'authentification ou les identifiants.

Les hooks React

Les **hooks** sont des fonctions introduites dans React pour apporter aux composants fonctionnels des fonctionnalités avancées, auparavant réservées aux composants de type classe. Voici ceux que j'utilise principalement :

useEffect

- Permet de gérer les cycles de vie d'un composant fonctionnel.
- Sert à déclencher des actions (comme une requête API ou une mise à jour du stockage) après le rendu du composant ou lors d'un changement d'état ou de props.

useState

- Permet de déclarer des variables d'état locales à un composant fonctionnel.
- Lorsqu'un état est modifié avec la fonction associée, le composant est automatiquement re-rendu pour refléter la nouvelle valeur.

Gestion avancée de l'état avec useReducer

Pour gérer des états plus complexes ou imbriqués, j'utilise le hook **useReducer**. Il s'inspire du modèle de réduction de Redux et se révèle particulièrement utile lorsque :

- L'état de l'application est composé de plusieurs propriétés liées.
- Les mises à jour de l'état dépendent d'actions précises.
- La logique de modification de l'état devient trop complexe pour être gérée avec plusieurs useState.

Fonctionnement :

- On définit une fonction « réducteur » (`reducer`) qui prend l'état courant et une action, puis retourne le nouvel état.
- On initialise l'état avec une valeur de départ.
- On utilise la fonction `dispatch` pour envoyer des actions au réducteur.

Exemple :

```
javascript
```

```
const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      throw new Error();
  }
}

const [state, dispatch] = useReducer(reducer, initialState);
```

Ce modèle rend la gestion de l'état plus prévisible et structurée, surtout dans des applications où plusieurs variables d'état doivent évoluer ensemble

```

export default function App() {
  const [state, dispatch] = React.useReducer(
    (prevState, action) => {
      switch (action.type) {
        case "RESTORE_TOKEN":
          return {
            ...prevState,
            userToken: action.token,
            isLoading: false,
          };
        case "SIGN_IN":
          return {
            ...prevState,
            isSignout: false,
            userToken: action.token,
          };
        case "SIGN_OUT":
          return {
            ...prevState,
            isSignout: true,
            userToken: null,
          };
      }
    },
    {
      isLoading: true,
      isSignout: false,
      userToken: null,
    }
);

```

Comme illustré précédemment dans l'exemple de mon appel API, à la fin de l'exécution du script, j'utilise la méthode `dispatch` pour mettre à jour l'état global de l'application. Je transmets en paramètre un type d'action, ici "`SIGN_IN`", ainsi que la donnée associée, en l'occurrence le token d'authentification. Ainsi, la valeur de `userToken` est mise à jour avec le token reçu.

Si `userToken` est différent de `null`, l'utilisateur est considéré comme authentifié. Je lui donne alors accès à la navigation principale de l'application, en basculant de la stack de navigation dédiée à l'authentification vers la TabBar qui regroupe toutes les vues accessibles à un utilisateur connecté. Cette logique est gérée à l'aide d'une expression ternaire, un opérateur JavaScript permettant de choisir rapidement entre deux valeurs selon une condition :

`condition ? exprSiVrai : exprSiFaux`

Pour éviter les problèmes liés à l'asynchronisme de JavaScript lors des appels API, j'utilise les mots-clés `async` et `await` dans mes fonctions asynchrones. Cela permet de garantir l'ordre d'exécution des requêtes, d'améliorer la lisibilité du code et de faciliter le traitement des réponses de l'API.

En ce qui concerne la navigation entre deux vues, comme par exemple entre la page de connexion et celle d'inscription, il suffit de passer la propriété

{navigation} à mon composant. Cela me permet d'utiliser les méthodes de navigation fournies par React Navigation pour changer d'écran facilement.

```
export default function connexion({ navigation }) {
  const { signIn } = useContext(AuthContext);

  let [fontsLoaded] = useFonts({ Poppins_200ExtraLight });
  const [login, setLogin] = useState("");
  const [password, setPassword] = useState("");

  const goTo = () => {
    navigation.navigate("Inscription");
  };
}
```

Ensuite, je stocke dans une constante l'appel à la méthode navigation.navigate("nom_du_screen"). Ainsi, lorsque j'exécute cette constante, la vue portant le nom « Inscription » sera affichée à l'écran. Cela me permet de déclencher la navigation vers l'écran d'inscription de manière simple et réutilisable dans mon code.

```
<Pressable
  onPress={() => {
    goTo();
  }}
  style={styles.button1}>
```

Navigation avec Pressable et gestion du retour dans React Navigation

Utilisation de Pressable pour naviguer

Dans React Native, la balise Pressable permet de créer des boutons interactifs. Pour naviguer vers la vue "Inscription" lors d'un appui, il suffit d'utiliser la méthode navigation.navigate("Inscription") à l'intérieur de la fonction de rappel du bouton. Par exemple :

```
jsx
import { Pressable, Text } from 'react-native';

<Pressable onPress={() =>
navigation.navigate("Inscription")}>
  <Text>Aller à l'inscription</Text>
</Pressable>
```

À chaque appui sur ce bouton, l'utilisateur sera redirigé vers l'écran d'inscription.

Bouton de retour avec React Navigation

Pour implémenter un bouton de retour, React Navigation propose la méthode `navigation.goBack()`. Cette méthode permet de revenir à l'écran précédent dans la pile de navigation, quelle que soit la structure de votre navigation (Stack, Tab, etc.). Exemple d'utilisation :

```
jsx
<Pressable onPress={() => navigation.goBack()}>
  <Text>Retour</Text>
</Pressable>
```

Couverture des cas courants par React Navigation

La librairie React Navigation couvre de nombreux cas de figure pour la navigation mobile :

- Aller vers un écran spécifique : `navigation.navigate("NomEcran")`
- Revenir en arrière : `navigation.goBack()`
- Remplacer l'écran courant : `navigation.replace("NomEcran")`
- Réinitialiser la pile de navigation : `navigation.reset({...})`
- Passer des paramètres lors de la navigation

Cela permet de gérer facilement la navigation, les transitions et les retours dans toutes les situations courantes d'une application mobile

```
const goTo = () => {
  navigation.goBack();
};
```

Concernant la mise en place de ma user interface, il ne s'agit pas de HTML à proprement dit mais de JSX, cela ressemble à du HTML :

Mise en place de l'interface utilisateur : le rôle du JSX

Lorsque l'on développe une interface utilisateur avec React Native, on n'utilise pas du HTML classique, mais une syntaxe appelée **JSX** (JavaScript XML). Visuellement, le JSX ressemble beaucoup au HTML, mais il s'agit en réalité d'une extension de syntaxe JavaScript qui permet de décrire l'interface directement dans le code JavaScript.

Qu'est-ce que le JSX ?

- **JSX** permet d'écrire des éléments d'interface utilisateur sous forme de balises proches du HTML, tout en intégrant la puissance de JavaScript.
- Par exemple, au lieu d'écrire :

```
xml  
<h1>Bonjour, tout le monde !</h1>
```

en HTML, on écrira en JSX :

```
jsx  
<Text>Bonjour, tout le monde !</Text>
```

En React Native, les balises HTML classiques (`<div>`, ``, etc.) sont remplacées par des composants spécifiques comme `<View>`, `<Text>`, ou `<Image>`, qui correspondent à des éléments natifs sur chaque plateforme mobile.

Différences principales entre JSX et HTML

Aspect	JSX (React/React Native)	HTML classique
Syntaxe	Resssemble à du HTML, mais s'intègre à JS	Balises HTML standard
Composants	Utilise <code><View></code> , <code><Text></code> , <code><Image></code> , etc.	Utilise <code><div></code> , <code><p></code> , <code></code> , etc.
Intégration JS	Expressions JS entre {} dans les balises	JS séparé, souvent via <code><script></code>

Attributs	camelCase (onPress, className)	minuscules (onclick, class)
Rendu dynamique	Directement dans le JSX avec JS	Manipulation du DOM nécessaire
Transpilation	Nécessite Babel pour convertir en JS natif	Interprété nativement par le navigateur

Avantages du JSX

- **Intégration directe avec JavaScript** : on peut insérer des variables, des fonctions ou des expressions JS à l'intérieur des balises grâce aux accolades {}.
- **Composants réutilisables** : JSX encourage la création d'interfaces sous forme de petits composants modulaires et réutilisables.
- **Sécurité** : React échappe automatiquement les valeurs insérées dans le JSX, limitant les risques d'injection de code malveillant.
- **Lisibilité** : La syntaxe proche du HTML rend la structure de l'interface plus facile à lire et à maintenir.

Exemple concret en React Native

```
jsx
import React from 'react';
import { View, Text } from 'react-native';

const MonComposant = () => (
  <View>
    <Text>Bienvenue sur mon application !</Text>
  </View>
);
```

Ici, `<View>` joue un rôle similaire à une `<div>` en HTML, et `<Text>` remplace les balises de texte comme `<p>` ou `<h1>`.

À retenir

- Le JSX n'est pas du HTML, même s'il y ressemble beaucoup : il s'agit d'une syntaxe JavaScript qui permet de décrire l'interface utilisateur de façon

déclarative et dynamique.

- En React Native, on utilise des composants spécifiques à la plateforme mobile plutôt que les balises HTML classiques.
- L'utilisation du JSX facilite la création, la maintenance et la réutilisation des interfaces dans les applications mobiles modernes.

En résumé :

Le JSX est la pierre angulaire de la création d'interfaces en React Native. Il reprend la simplicité du HTML tout en offrant la puissance et la flexibilité du JavaScript, mais il s'appuie sur des composants spécifiques à l'écosystème mobile.

```
<KeyboardAwareScrollView showsVerticalScrollIndicator={false}>
  <View style={styles.container}>
    <Text style={styles.title}>Coté Pote</Text>
    <Text style={styles.txt}>L'application de pari entre potes</Text>
    <TextInput
      style={styles.ipt}
      onChangeText={setLogin}
      value={login}
      placeholder="Email"
      placeholderTextColor="#CFCFCF"
    />
    <TextInput
      style={styles.ipt2}
      onChangeText={setPassword}
      secureTextEntry={true}
      value={password}
      placeholder="Mot de passe"
      placeholderTextColor="#CFCFCF"
    />
    <Pressable
      onPress={() => {
        connexion();
      }}
      style={styles.button}
    >
```

Pour le style, ce n'est toujours pas du CSS :

Avec React Native, je stylise mon application en utilisant JavaScript. Tous les composants de base acceptent un accessoire nommé style. Les noms et les valeurs de style correspondent généralement au fonctionnement de CSS sur le Web, sauf que les noms sont écrits en utilisant la camelCase, par exemple backgroundColor plutôt que background-color...

Il faut commencer par importer "StyleSheet" depuis la librairie react-native et ensuite déclaré une constante où l'on va stocker la méthode "StyleSheet.Create({})" et à l'intérieur déclarer toute les classes dont j'ai besoins :

Styliser une application React Native

Contrairement au web où l'on utilise le CSS, le style dans React Native se fait entièrement en JavaScript. Tous les composants de base acceptent une propriété `style` qui permet de leur appliquer des styles personnalisés.

Quelques points essentiels :

- Les noms de propriétés de style sont en camelCase (par exemple, `backgroundColor` au lieu de `background-color`).
- Les valeurs de style sont souvent similaires à celles du CSS, mais il existe des différences et des limitations propres à l'environnement mobile.
- Les styles sont définis sous forme d'objets JavaScript, ce qui permet de profiter de la puissance du langage pour créer des styles dynamiques ou conditionnels.

Utilisation de StyleSheet

Pour organiser les styles, il est recommandé d'utiliser l'utilitaire `StyleSheet` fourni par la bibliothèque `react-native`. Voici la démarche standard :

1. Importer StyleSheet :

```
javascript
import { StyleSheet } from
'react-native';
```

2. Créer un objet de styles :

```
javascript

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#fff',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    color: '#333',
  },
});
```

3. Appliquer les styles à un composant :

```
jsx

<View style={styles.container}>
  <Text style={styles.title}>Bienvenue sur mon application !</Text>
</View>
```

Grâce à cette approche, les styles sont centralisés, réutilisables et optimisés pour les performances sur mobile.

À retenir

- JSX facilite la création d'interfaces en React Native, mais les balises diffèrent de celles du HTML classique.
- Le style se fait en JavaScript, via la propriété `style` et l'utilitaire `StyleSheet`.
- Les noms de propriétés sont en camelCase et la documentation officielle propose des ressources pour trouver les équivalents HTML/React Native.

Références :

<https://reactnative.dev/docs/components-and-apis>

<https://reactnative.dev/docs/style>

<https://blog.logrocket.com/styles-in-react-native/>

```
const styles = StyleSheet.create({
  container: {
    marginLeft: 20,
    marginTop: 250,
  },
  title: {
    fontFamily: "Poppins_700Bold",
    fontSize: 45,
    fontWeight: "bold",
    color: "white",
  },
  txt: {
    fontFamily: "Poppins_200ExtraLight",
    fontSize: 15,
    fontWeight: "bold",
    color: "white",
  },
  ipt: {
    height: 50,
    width: 335,
    borderWidth: 0.25,
    borderColor: "white",
    borderRadius: 5,
    padding: 10,
    marginTop: 50,
    color: "white",
  },
})
```

et ensuite pour attribuer du style à un élément dans mon code je le fais de la manière suivante :

```
<Text style={styles.text1}>'inscrire</Text>
```

8. Extrait d'une recherche à partir de site anglophone

Recherche de solutions techniques : l'exemple d'ImagePicker

Au cours du développement de ce projet, j'ai été confronté à de nombreux défis techniques qui ont nécessité des recherches approfondies pour trouver des solutions adaptées. La majorité des réponses à mes questions ont été trouvées sur des sites anglophones spécialisés, en particulier Stack Overflow, qui s'est révélé être une ressource précieuse pour le débogage et la résolution de problèmes complexes liés à React Native et à ses bibliothèques.

Exemple concret : débogage avec la bibliothèque ImagePicker

L'un des cas les plus marquants concernait l'intégration de la bibliothèque **ImagePicker**, utilisée pour permettre à l'utilisateur de sélectionner ou prendre des photos au sein de l'application. Lors de la mise en œuvre de cette fonctionnalité, j'ai rencontré un problème : la sélection d'images ne fonctionnait pas correctement sur certaines plateformes, ou des erreurs apparaissaient lors de l'accès à la galerie ou à la caméra. Pour résoudre ce souci, je me suis tourné vers Stack Overflow, où de nombreux développeurs partagent leurs expériences et solutions pour des problèmes similaires. Les étapes de résolution recommandées incluaient :

- **Vérification des permissions** : S'assurer que les autorisations nécessaires sont bien déclarées dans les fichiers de configuration (`AndroidManifest.xml` pour Android, `Info.plist` pour iOS), notamment pour l'accès à la galerie et à la caméra.
- **Compatibilité des versions** : Confirmer que la version d'ImagePicker utilisée était bien compatible avec la version de React Native du projet, car certains bugs sont spécifiques à certaines versions ou plateformes.
- **Nettoyage et reconstruction du projet** : Après modification des configurations ou des dépendances, il était conseillé de nettoyer le cache du projet et de relancer la compilation pour s'assurer que les changements étaient bien pris en compte.
- **Consultation des issues GitHub** : Lire les discussions sur le dépôt GitHub d'ImagePicker, où des utilisateurs partagent des correctifs ou des contournements efficaces pour des problèmes similaires.

Illustration d'une solution trouvée

Par exemple, pour un problème d'accès à la galerie sur Android 12, la solution trouvée sur Stack Overflow consistait à ajouter une section `<queries>` dans le fichier `AndroidManifest.xml`, même si Android Studio affichait un avertissement. Cette modification a permis de restaurer le fonctionnement de la sélection d'images sur les appareils concernés.

Cette expérience illustre l'importance de la veille technique et des ressources communautaires, en particulier Stack Overflow, dans le développement d'applications mobiles. Les échanges et solutions partagés par la communauté permettent souvent de débloquer rapidement des situations complexes, même pour des problèmes spécifiques à des bibliothèques comme ImagePicker.

<https://stackoverflow.com/questions/tagged/expo-imagepicker>

stackoverflow Products Search... 1

Results from the 2022 Developer Survey are [here](#).

Home PUBLIC Questions Tags Users Companies COLLECTIVES Explore Collectives TEAMS Stack Overflow for Teams – Start collaborating and sharing organizational knowledge. Create a free Team Why Teams?

how to select image from expo-image-picker react native

Asked 3 months ago Modified 3 months ago Viewed 328 times

0 I try to select image from gallery and show it on Image component. I read the expo documentation and follow the step . when I try it on web it's work. but when I try it on android phone , after I select image and crop my app redownloading . I don't know where make mistake

• my code

```
import React, { useState, useEffect } from 'react';
import * as ImagePicker from 'expo-image-picker';
import { Button, Image, View, Platform } from 'react-native';

export default function GalleryComponenet() {
  const [image, setImage] = useState(null);

  useEffect(() => {
    (async () => {
      if (Platform.OS !== 'web') {
        const { status } = await ImagePicker.requestMediaLibraryPermissionsAsync();
        if (status !== 'granted') {
          alert('Sorry, Camera roll permissions are required to make this work');
        }
      }
    })();
  }, []);

  const chooseImage = async () => {
```

The Overflow Blog

- Experts from Stripe and Waymo explain how to craft great documentation (Ep. 455)
- Asked and answered: the results for the 2022 Developer survey are here!

Featured on Meta

- Announcing the arrival of Valued Associate #1214: Dalmarus
- Testing new traffic management tool
- Ask Wizard Test Results and Next Steps
- Trending: A new answer sorting option
- Updated button styling for vote arrows: currently in A/B testing

Hot Meta Posts

- Add a magic link to <https://stackoverflow.com/editing-help>

Conclusion

Ce projet m'a permis de mettre en œuvre, de façon concrète et structurée, l'ensemble des compétences attendues dans le référentiel du diplôme. De la phase de conception à la mise en production, j'ai pu aborder toutes les étapes clés du développement d'une application mobile moderne : analyse des besoins, modélisation de la base de données, conception de l'interface utilisateur, développement d'une API sécurisée et intégration des recommandations en matière de sécurité et de conformité (RGPD, accessibilité).

La diversité des technologies utilisées (React Native, Expo, Symfony/API Platform, MySQL) ainsi que l'organisation du travail en mode collaboratif m'ont permis de développer mon autonomie, ma rigueur et ma capacité à résoudre des problèmes techniques variés. Les difficultés rencontrées ont été surmontées grâce à une veille technologique active et à l'exploitation des ressources communautaires, notamment Stack Overflow.

Ce dossier illustre non seulement la maîtrise des aspects techniques et méthodologiques du développement d'applications multicouches, mais aussi l'importance de l'adaptabilité, de la documentation et du respect des bonnes pratiques professionnelles.

En définitive, cette expérience a renforcé ma motivation à évoluer dans le domaine du développement logiciel et constitue une étape clé dans mon parcours professionnel.