Coffee Shop Project: Guide de Révision

Ce document explique le fonctionnement de ton projet React Native pour t'aider à préparer ton examen.

1. Architecture Globale

Le projet utilise **React Native** avec **TypeScript**. La gestion de l'état (les données de l'application) est centralisée dans un "Store" (Zustand).

Navigation (

App.tsx &

TabNavigator.tsx)

- **Stack Navigation (**

  App.tsx**)** : Gère le flux principal (Welcome -> Login/Register -> Application principale).

- **Tab Navigation (**

  TabNavigator.tsx**)** : La barre en bas qui permet de naviguer entre Home, Cart (Panier), Favorite (Favoris), et Profile.

---

2. Les Interfaces (Écrans)

Écran	Description
WelcomeScreen	L'écran d'accueil/introduction avec le bouton "Get Started".
Login / Register	Permettent à l'utilisateur de se connecter ou de créer un compte. Les données sont sauvegardées dans le Store.
HomeScreen	L'interface principale. Elle affiche les catégories de café, une barre de recherche, et la liste des produits (Cafés et Offres Spéciales).
DetailsScreen	S'affiche quand tu cliques sur un café. Tu peux choisir la taille (S, M, L), le niveau de sucre, et l'ajouter au panier ou aux favoris.
CartScreen	Affiche les articles ajoutés. Il calcule automatiquement le total et permet de choisir un mode de paiement (Visa, PayPal, etc.).
FavoritesScreen	Liste tous les cafés que tu as "liké" (coeur orange).
ProfileScreen	Affiche tes infos, ton historique de commandes, et le bouton de déconnexion.

---

3. Logique et État (Le Store)

Le fichier
src/store/store.ts est le "cerveau" de l'application.

- **Persistance** : Utilise `AsyncStorage` pour que tes favoris et ton panier ne disparaissent pas quand tu fermes l'application.
- **Actions** :
  - addToCart : Ajoute un article au panier.
  - calculateCartPrice : Calcule le prix total.
  - toggleDarkMode : Change le thème de l'app.
  - addToOrderHistoryListFromCart : Transforme le panier en une commande terminée.

---

4. Les Hooks React (Très important pour l'examen !)

`useState`

C'est pour l'état **local** (à l'intérieur d'un seul écran).

- **Pourquoi ?** Pour que l'écran se mette à jour dès qu'une donnée change.
- **Exemples dans ton projet :**
  - Dans

    HomeScreen.tsx : `searchText` (ce que l'utilisateur tape), `categoryIndex` (quelle catégorie est sélectionnée).

  - Dans

    DetailsScreen.tsx : `price` (la taille choisie), `sugarLevel` (le niveau de sucre).

  - Dans

    CartScreen.tsx : `selectedPaymentMethod`

`useEffect`

C'est pour les **effets de bord** (actions qui se lancent à un moment précis du cycle de vie).

- **Pourquoi ?** Pour faire quelque chose quand l'écran s'affiche pour la première fois (Mount) ou quand une variable change.
- **Exemples dans ton projet :**
  - Dans

    App.tsx : `useEffect(() => { BootSplash.hide(...) }, [])`. Ce code cache l'écran de démarrage dès que l'application est prête. Le tableau vide `[]` signifie que ça ne s'exécute **qu'une seule fois** au démarrage.

Cette ligne est un exemple parfait de l'utilisation du hook

`useState` de React. C'est presque certain qu'on te posera une question dessus à l'examen !

Voici le découpage de la ligne :

typescript
```
const [selectedPaymentMethod, setSelectedPaymentMethod] =
useState('Visa');
```

1.

`useState('Visa')`

- C'est l'appel au hook.
- `'Visa'` est la **valeur initiale**. Quand l'écran s'affiche pour la première fois, la méthode de paiement sélectionnée sera "Visa".

2.

`selectedPaymentMethod`

(La Variable)

- C'est la variable qui stocke la donnée actuelle.
- Tu l'utilises dans ton code pour savoir ce qu'il faut afficher (par exemple, mettre une bordure verte autour de l'icône Visa).

3.

`setSelectedPaymentMethod`

(La Fonction)

- C'est la fonction qui permet de **modifier** la valeur.
- **Règle d'or :** En React, on ne fait jamais `selectedPaymentMethod = 'PayPal'`. On doit obligatoirement utiliser `setSelectedPaymentMethod('PayPal')`.
- Pourquoi ? Parce que cette fonction dit à React : *"Hé, la donnée a changé, s'il te plaît, redessine l'écran (Re-render) pour que l'utilisateur voie le changement."*

4. Le crochet

`[ ]`

(Destructuration)

- `useState`

  renvoie toujours un tableau avec deux éléments : la valeur et la fonction. Les crochets servent juste à donner un nom à ces deux éléments d'un seul coup.

Résumé pour l'examen :

"*Cette ligne déclare une* ***variable d'état locale*** *appelée*

`selectedPaymentMethod`

avec 'Visa' comme valeur par défaut. Elle fournit aussi une fonction

`setSelectedPaymentMethod`

pour mettre à jour cette valeur et déclencher un rafraîchissement automatique de l'interface."

5. C'est quoi le "Store" ? (Zustand)

Imagine que l'application est un restaurant.

- Les **Screens** (écrans) sont les tables où les clients s'asseyent.
- Le **Store** est la **cuisine centrale**.

Pourquoi on en a besoin ?

Si tu ajoutes un café au panier dans l'écran
Details, comment l'écran
Cart sait-il qu'il doit l'afficher ? Sans Store, ce serait très compliqué de passer l'information d'un écran à l'autre. Le Store centralise tout :

1. **L'état (State)** : Les données actuelles (ex: la liste des cafés dans le panier).
2. **Les Actions** : Les fonctions pour changer les données (ex:

   addToCart).

IMPORTANT
Dans ton projet, on utilise **Zustand**. C'est une bibliothèque légère pour gérer ce Store. On utilise aussi `persist` pour que les données soient sauvegardées sur le téléphone même après avoir fermé l'appli.

---

6. C'est quoi "Lottie" ?

Lottie est un outil qui permet d'afficher des **animations** de haute qualité très facilement.

- **Le format** : Les animations sont des fichiers `.json`. C'est beaucoup plus léger qu'une vidéo ou un GIF.
- **Utilisation dans ton projet** :
  - Tu as un dossier `src/lottie`.
  - Exemples : L'animation du chargement (Loading) ou quand le panier est vide (`EmptyListAnimation.tsx`).
- **Avantage** : On peut contrôler l'animation (la lancer, l'arrêter, changer sa vitesse) directement avec du code.

---

7. Composants Clés

- **CoffeeCard** : Le petit rectangle qui affiche l'image, le nom et le prix du café sur la Home.
- **HeaderBar** : La barre en haut avec le titre ou le logo.
- **OrderHistoryCard** : Affiche une ancienne commande dans le profil.

TIP
Si on te demande comment le "Dark Mode" fonctionne : il utilise une variable `isDarkMode` dans le store. Chaque composant récupère ses couleurs via `getColors(isDarkMode)` pour s'adapter automatiquement.

10. Anatomie d'un Composant (`CoffeeCard.tsx`)

C'est un sujet classique d'examen : comment est structuré un fichier de composant ?

A. L'Interface (`interface CoffeeCardProps`)

Avant de créer le composant, on définit une **interface**.

- **C'est quoi ?** C'est un "contrat" ou un plan qui liste toutes les données (Props) que le composant doit recevoir pour fonctionner (nom, prix, image, etc.).
- **Pourquoi ?** Pour que TypeScript nous aide. Si tu oublies de donner le `price` à une `CoffeeCard`, TypeScript affichera une erreur immédiatement.

B. `React.FC<CoffeeCardProps>`

- `FC` signifie **Functional Component** (Composant Fonctionnel).
- **Les chevrons** `<...>` : On dit à React que ce composant fonctionnel utilise les propriétés définies dans l'interface `CoffeeCardProps`.

C. Le fonctionnement

1. **Réception des Props** : Le composant reçoit les infos du parent (souvent la `HomeScreen`).

2. **Affichage (Render)** : Il utilise du JSX (`View`, `Text`, `Image`) pour dessiner le petit rectangle du café.
3. **Actions** : Il possède des boutons (`TouchableOpacity`) qui appellent des fonctions comme `buttonPressHandler` pour ajouter au panier.

NOTE
Rappel pour l'examen : Ton projet a été simplifié. Il n'y a plus de "Beans" (Grains), tout est unifié sous le type **Coffee** pour plus de clarté.

11. `BGIcon` vs `GradientBGIcon`

Ces deux composants servent à afficher des icônes avec un arrière-plan, mais ils ont une petite différence visuelle.

A. `BGIcon.tsx` (Simple)

- **C'est quoi ?** Une icône dans un carré de couleur **unie** (flat color).
- **Usage** : Utilisé pour le bouton "+" (Ajouter au panier) dans la `CoffeeCard`.
- **Props** : Il a besoin de `BGColor` pour savoir quelle couleur unie afficher en fond.

B. `GradientBGIcon.tsx` (Premium)

- **C'est quoi ?** Une icône dans un carré avec un **dégradé** de couleurs (Linear Gradient).
- **Usage** : Utilisé pour les boutons de navigation (Flèche retour) ou le bouton "Favoris" dans les écrans de détails.
- **Pourquoi ?** Pour donner un aspect plus "premium" et moderne à l'interface, conformément aux standards de design actuels.
- **Technique** : Il utilise la bibliothèque `react-native-linear-gradient`.

---

12. C'est quoi un `LinearGradient` ?

C'est une technologie qui permet de créer une transition fluide entre deux ou plusieurs couleurs.

Les propriétés clés (Props) :

Dans ton fichier `GradientBGIcon.tsx`, tu as ce code :

```
<LinearGradient
  start={{x: 0, y: 0}} // Début (Haut à gauche)
  end={{x: 1, y: 1}}   // Fin (Bas à droite)
  colors={[COLORS.primaryGreyHex, COLORS.primaryBlackHex]}
>
```

1. **`colors`** : C'est une liste (tableau) de couleurs. L'effet passera de la première à la deuxième couleur.
2. **`start` et `end`** : Ils définissent la **direction** du dégradé (vertical, horizontal ou diagonal). Ici, `(0,0)` vers `(1,1)` crée un dégradé **diagonal**.

Pourquoi l'utiliser ?

- **Esthétique** : Ça donne du relief et de la profondeur à l'interface. Un bouton avec un dégradé paraît souvent plus "cliquable" et pro qu'un bouton plat.
- **Performance** : C'est généré par le téléphone directement, donc c'est beaucoup plus léger que d'utiliser une image de fond.

13. Composants vs Screens (Écrans)

C'est une question très fréquente à l'examen. Voici la différence avec une analogie :
Imagine que tu construis une maison en LEGO.

A. Les Composants (`src/components/`)

- **C'est quoi ?** Ce sont les **briques individuelles**.
- **Exemples** : Un bouton, une icône, la carte d'un café (`CoffeeCard`), la photo de profil (`ProfilePic`).
- **Caractéristique** : Ils sont coincés à un seul usage mais peuvent être **réutilisés**. Tu peux mettre la même `CoffeeCard` dans l'écran Home et dans l'écran Favoris. Ils ne "savent pas" où ils se trouvent, ils reçoivent juste des données pour les afficher.

B. Les Screens (`src/screens/`)

- **C'est quoi ?** C'est la **pièce entière** (Cuisine, Chambre).
- **Exemples** : `HomeScreen`, `CartScreen`, `DetailsScreen`.
- **Caractéristique** : Un screen est une page complète que l'utilisateur voit.
  Il **assemble** plusieurs composants pour former une interface. Un screen est relié à la **navigation** (on peut "aller" vers un screen).

En résumé :

"Un **Composant** est une petite pièce réutilisable et autonome. Un **Screen** est un assemblage de composants qui représente une page entière de l'application et qui est géré par le système de navigation."