PLAN DE TEST

Plan de Test Détaillé pour les Fonctions du Panier

Fonction `calculateTotal`

1. **Test: Calculer le total pour un seul article du panier**
   - **Arrange:**
     - Créer un tableau `basketItems` contenant un seul `BasketItem` avec un prix défini.
   - **Act:**
     - Appeler la fonction `calculateTotal(basketItems)`.
   - **Assert:**
     - Vérifier que le total est égal au prix de l'unique article dans `basketItems`.
2. **Test: Calculer le total pour plusieurs articles du panier**
   - **Arrange:**
     - Créer un tableau `basketItems` contenant plusieurs `BasketItem` avec des prix définis.
   - **Act:**
     - Appeler la fonction `calculateTotal(basketItems)`.
   - **Assert:**
     - Vérifier que le total est égal à la somme des prix des articles dans `basketItems`.
3. **Test: Appliquer une remise au total**
   - **Arrange:**
     - Créer un tableau `basketItems` contenant plusieurs `BasketItem` avec des prix définis.
     - Définir une remise `discount`.
   - **Act:**
     - Appeler la fonction `calculateTotal(basketItems, discount)`.
   - **Assert:**
     - Vérifier que le total est égal à la somme des prix des articles moins la remise.

Fonction `showAdverts`

1. **Test: Retourne `false` pour les utilisateurs premium**
   - **Arrange:**
     - Créer un objet `user` avec la propriété `isPremium` à `true`.
   - **Act:**
     - Appeler la fonction `showAdverts(user)`.
   - **Assert:**
     - Vérifier que la fonction retourne `false`.
2. **Test: Retourne `true` pour les utilisateurs non premium**
   - **Arrange:**
     - Créer un objet `user` avec la propriété `isPremium` à `false`.

- **Act:**
  - Appeler la fonction showAdverts(user).
- **Assert:**
  - Vérifier que la fonction retourne true.

Fonction searchBasket

1. **Test: Retourne les articles du panier correspondant à la requête de recherche**
   - **Arrange:**
     - Créer un tableau basketItems contenant plusieurs BasketItem.
     - Définir une requête de recherche searchQuery correspondant à un ou plusieurs articles.
   - **Act:**
     - Appeler la fonction searchBasket(basketItems, searchQuery).
   - **Assert:**
     - Vérifier que les résultats contiennent les articles correspondant à searchQuery.
2. **Test: Retourne un tableau vide si aucun article ne correspond à la requête de recherche**
   - **Arrange:**
     - Créer un tableau basketItems contenant plusieurs BasketItem.
     - Définir une requête de recherche searchQuery qui ne correspond à aucun article.
   - **Act:**
     - Appeler la fonction searchBasket(basketItems, searchQuery).
   - **Assert:**
     - Vérifier que les résultats sont un tableau vide.

Fonction getBasketItem

1. **Test: Retourne l'article du panier avec l'événement spécifié**
   - **Arrange:**
     - Créer un objet event et un tableau basketItems contenant plusieurs BasketItem.
   - **Act:**
     - Appeler la fonction getBasketItem(basketItems, event).
   - **Assert:**
     - Vérifier que la fonction retourne l'article correspondant à l'événement.
2. **Test: Retourne null si l'événement n'est pas trouvé dans le panier**
   - **Arrange:**
     - Créer un objet event qui n'est pas dans le panier et un tableau basketItems contenant plusieurs BasketItem.
   - **Act:**
     - Appeler la fonction getBasketItem(basketItems, event).
   - **Assert:**
     - Vérifier que la fonction retourne null.

Fonction `createBasketItem`

1. **Test: Crée un nouvel article du panier si l'événement n'est pas déjà dans le panier**
    - **Arrange:**
        - Créer un objet `event` et un tableau `basketItems` sans cet événement.
        - Définir le nombre de tickets requis `requiredTickets`.
    - **Act:**
        - Appeler la fonction `createBasketItem(basketItems, event, requiredTickets)`.
    - **Assert:**
        - Vérifier que la fonction retourne un nouvel article du panier avec l'événement spécifié et le nombre de tickets requis.
2. **Test: Retourne `null` si l'événement est déjà dans le panier**
    - **Arrange:**
        - Créer un objet `event` qui est déjà dans le panier et un tableau `basketItems` contenant cet événement.
        - Définir le nombre de tickets requis `requiredTickets`.
    - **Act:**
        - Appeler la fonction `createBasketItem(basketItems, event, requiredTickets)`.
    - **Assert:**
        - Vérifier que la fonction retourne `null`.

Fonction `serializeBasketItemsToJson`

1. **Test: Sérialise les articles du panier en tableau JSON**
    - **Arrange:**
        - Créer un tableau `basketItems` contenant plusieurs `BasketItem`.
    - **Act:**
        - Appeler la fonction `serializeBasketItemsToJson(basketItems)`.
    - **Assert:**
        - Vérifier que la longueur du tableau JSON est égale à la longueur de `basketItems`.
        - Vérifier que chaque élément du tableau JSON est égal à l'article correspondant dans `basketItems`.

Tests pour la classe `Event`

Test 1.1 : Création d'une instance de `Event`

- **Description** : Vérifie si une instance de `Event` est correctement créée avec les valeurs fournies.
- **Étapes** :
    - Arrange : Créer les valeurs pour l'ID, le nom, le prix du ticket, le nombre total de tickets, les tickets restants et la date.
    - Act : Créer une instance de `Event` avec ces valeurs.
    - Assert : Vérifier que les propriétés de l'instance `Event` correspondent aux valeurs fournies.
- **Cas de test** :
    - Vérifier que `event.id` est égal à `id`.
    - Vérifier que `event.name` est égal à `name`.
    - Vérifier que `event.ticketPrice` est égal à `ticketPrice`.

    - Vérifier que event.totalTickets est égal à totalTickets.
    - Vérifier que event.ticketsRemaining est égal à ticketsRemaining.
    - Vérifier que event.date est égal à date.

Tests pour `isSoldOut`

Test 2.1 : Retourne `true` si `ticketsRemaining` est 0

- **Description** : Vérifie si la fonction retourne true lorsque les tickets restants sont 0.
- **Étapes** :
    - Arrange : Créer une instance de Event avec ticketsRemaining égal à 0.
    - Act : Appeler isSoldOut avec cette instance.
    - Assert : Vérifier que le résultat est true.

Test 2.2 : Retourne `false` si `ticketsRemaining` est supérieur à 0

- **Description** : Vérifie si la fonction retourne false lorsque les tickets restants sont plus de 0.
- **Étapes** :
    - Arrange : Créer une instance de Event avec ticketsRemaining supérieur à 0.
    - Act : Appeler isSoldOut avec cette instance.
    - Assert : Vérifier que le résultat est false.

Tests pour `getTagLine`

Test 3.1 : Retourne "Event Sold Out!" si l'événement est complet

- **Description** : Vérifie si la fonction retourne "Event Sold Out!" lorsque l'événement est complet (ticketsRemaining est 0).
- **Étapes** :
    - Arrange : Créer une instance de Event avec ticketsRemaining égal à 0.
    - Act : Appeler getTagLine avec cette instance, un minimumTicketCount et isPopular false.
    - Assert : Vérifier que le résultat est "Event Sold Out!".

Test 3.2 : Retourne "Hurry only X tickets left!" si les tickets restants sont moins que `minimumTicketCount`

- **Description** : Vérifie si la fonction retourne "Hurry only X tickets left!" lorsque les tickets restants sont moins que minimumTicketCount.
- **Étapes** :
    - Arrange : Créer une instance de Event avec ticketsRemaining moins que minimumTicketCount.
    - Act : Appeler getTagLine avec cette instance, un minimumTicketCount et isPopular false.
    - Assert : Vérifier que le résultat est "Hurry only X tickets left!".

Test 3.3 : Retourne un slogan populaire si `isPopular` est `true`

- **Description** : Vérifie si la fonction retourne un slogan populaire lorsque isPopular est true.
- **Étapes** :

    - Arrange : Créer une instance de Event avec ticketsRemaining supérieur à minimumTicketCount.
    - Act : Appeler getTagLine avec cette instance, un minimumTicketCount et isPopular true.
    - Assert : Vérifier que le résultat est "This Event is getting a lot of interest. Don't miss out, purchase your ticket now!".

Test 3.4 : Retourne un slogan standard sinon

- **Description** : Vérifie si la fonction retourne un slogan standard lorsque l'événement n'est pas complet, a assez de tickets et n'est pas populaire.
- **Étapes** :
    - Arrange : Créer une instance de Event avec ticketsRemaining supérieur à minimumTicketCount.
    - Act : Appeler getTagLine avec cette instance, un minimumTicketCount et isPopular false.
    - Assert : Vérifier que le résultat est "Don't miss out, purchase your ticket now!".

Tests pour **createEvent**

Test 4.1 : Créer un événement si les entrées sont valides

- **Description** : Vérifie si un événement est créé correctement avec des entrées valides.
- **Étapes** :
    - Arrange : Définir un nom, un prix et des tickets disponibles valides.
    - Act : Appeler createEvent avec ces valeurs.
    - Assert : Vérifier que l'instance retournée est une instance de Event et que ses propriétés correspondent aux valeurs fournies.

Test 4.2 : Lève **InvalidEventNameError** si le nom n'est pas une chaîne ou dépasse 200 caractères

- **Description** : Vérifie si une exception InvalidEventNameError est levée lorsque le nom est invalide.
- **Étapes** :
    - Arrange : Définir un nom invalide (non chaîne ou dépasse 200 caractères).
    - Act & Assert : Appeler createEvent et vérifier qu'une exception InvalidEventNameError est levée.

Test 4.3 : Lève **InvalidEventPriceError** si le prix n'est pas un nombre ou est inférieur à 0

- **Description** : Vérifie si une exception InvalidEventPriceError est levée lorsque le prix est invalide.
- **Étapes** :
    - Arrange : Définir un prix invalide (non nombre ou inférieur à 0).
    - Act & Assert : Appeler createEvent et vérifier qu'une exception InvalidEventPriceError est levée.

Test 4.4 : Lève **InvalidEventPriceError** si **availableTickets** n'est pas un nombre ou est inférieur à 1

- **Description** : Vérifie si une exception InvalidEventPriceError est levée lorsque les tickets disponibles sont invalides.
- **Étapes** :
    - Arrange : Définir availableTickets invalide (non nombre ou inférieur à 1).
    - Act & Assert : Appeler `

Tests pour **getExchangeRate**

But: Vérifier que la fonction getExchangeRate appelle correctement le fournisseur de taux de change et retourne la réponse attendue.

Cas de Test:

1. **Test d'appel et de réponse réussie**:
    - **Description**: Vérifier que la fonction appelle le fournisseur de taux de change avec le bon code de devise et retourne la réponse attendue.
    - **Préconditions**: currencyCode = 'USD', expectedExchangeRate = 1.25
    - **Entrée**: currencyCode = 'USD'
    - **Sortie Attendue**: { originalCurrency: 'GBP', newCurrency: 'USD', exchangeRate: 1.25 }

Tests pour **calculatePercentageDiscount**

But: Vérifier que la fonction applique correctement un pourcentage de réduction si le montant minimum est atteint.

Cas de Test:

1. **Réduction appliquée si le minimum est atteint**:
    - **Description**: Vérifier que la réduction de pourcentage est appliquée si le montant minimum est atteint.
    - **Préconditions**: percentage = 20, minimumSpend = 100, currentPrice = 150
    - **Entrée**: percentage = 20, minimumSpend = 100, currentPrice = 150
    - **Sortie Attendue**: 120

2. **Réduction non appliquée si le minimum n'est pas atteint**:
    - **Description**: Vérifier que la réduction de pourcentage n'est pas appliquée si le montant minimum n'est pas atteint.
    - **Préconditions**: percentage = 20, minimumSpend = 200, currentPrice = 150
    - **Entrée**: percentage = 20, minimumSpend = 200, currentPrice = 150
    - **Sortie Attendue**: 150

Tests pour **calculateMoneyOff**

But: Vérifier que la fonction applique correctement une réduction fixe en argent si le montant minimum est atteint.

Cas de Test:

1. **Réduction appliquée si le minimum est atteint**:
    - **Description**: Vérifier que la réduction en argent est appliquée si le montant minimum est atteint.
    - **Préconditions**: discount = 20, minimumSpend = 100, currentPrice = 150
    - **Entrée**: discount = 20, minimumSpend = 100, currentPrice = 150
    - **Sortie Attendue**: 130
2. **Réduction non appliquée si le minimum n'est pas atteint**:

- **Description**: Vérifier que la réduction en argent n'est pas appliquée si le montant minimum n'est pas atteint.
- **Préconditions**: discount = 20, minimumSpend = 200, currentPrice = 150
- **Entrée**: discount = 20, minimumSpend = 200, currentPrice = 150
- **Sortie Attendue**: 150

Tests pour **generateReferralCode**

But: Vérifier que la fonction génère un code de parrainage au format attendu.

Cas de Test:

1. **Génération du code de parrainage**:
    - **Description**: Vérifier que le code de parrainage généré correspond au format attendu.
    - **Préconditions**: userId = 12345
    - **Entrée**: userId = 12345
    - **Sortie Attendue**: Format correspondant à #FRIEND-#\d{3}-#12345

Tests pour **applyDiscount**

But: Vérifier que la fonction applique correctement les réductions en argent et en pourcentage, et gère les cas où la réduction est invalide.

Cas de Test:

1. **Application correcte de la réduction en argent**:
    - **Description**: Vérifier que la réduction en argent est appliquée correctement.
    - **Préconditions**: discountCode = 'SAVE20', currentTotal = 150
    - **Mock**: getDiscount.mockResolvedValue({ isValid: true, type: 'MONEYOFF', value: 20, minSpend: 100 })
    - **Entrée**: discountCode = 'SAVE20', currentTotal = 150
    - **Sortie Attendue**: 130

2. **Application correcte de la réduction en pourcentage**:
    - **Description**: Vérifier que la réduction en pourcentage est appliquée correctement.
    - **Préconditions**: discountCode = 'SAVE20', currentTotal = 150
    - **Mock**: getDiscount.mockResolvedValue({ isValid: true, type: 'PERCENTAGEOFF', value: 20, minSpend: 100 })
    - **Entrée**: discountCode = 'SAVE20', currentTotal = 150
    - **Sortie Attendue**: 120

3. **Retourne le total actuel si la réduction est invalide**:

    - **Description**: Vérifier que le total reste inchangé si la réduction n'est pas valide.
    - **Préconditions**: discountCode = 'SAVE20', currentTotal = 150
    - **Mock**: getDiscount.mockResolvedValue({ isValid: false, type: 'MONEYOFF', value: 20, minSpend: 100 })
    - **Entrée**: `discountCode

Tests pour la Classe `User`

But

Vérifier que la classe `User` fonctionne comme prévu en initialisant correctement les propriétés.

Cas de Test

1. **Création d'un utilisateur avec les propriétés données**:
    - **Description**: Vérifier que la création d'un utilisateur initialise correctement les propriétés id, username et isPremium.
    - **Préconditions**: Aucune
    - **Entrée**: id = 1, username = 'testuser'
    - **Sortie Attendue**: user.id est 1, user.username est 'testuser', user.isPremium est false

Tests pour la Fonction `userExists`

But

Vérifier que la fonction `userExists` retourne la valeur correcte en fonction de l'existence de l'utilisateur.

Cas de Test

1. **Utilisateur existant**:
    - **Description**: Vérifier que la fonction retourne `true` pour un utilisateur existant.
    - **Préconditions**: Aucune
    - **Entrée**: username = "newuser1@pluralsight.com"
    - **Sortie Attendue**: true

2. **Utilisateur inexistant**:

- **Description**: Vérifier que la fonction retourne `false` pour un utilisateur non existant.
- **Préconditions**: Aucune
- **Entrée**: username = "nonexistentuser@pluralsight.com"
- **Sortie Attendue**: false

Tests pour la Fonction `createUserId`

But

Vérifier que la fonction `createUserId` génère correctement un nouvel identifiant utilisateur.

Cas de Test

1. **Retourne un nouvel identifiant utilisateur**:
    - **Description**: Vérifier que la fonction retourne un nouvel identifiant utilisateur (dans ce cas, 2).
    - **Préconditions**: Aucune
    - **Entrée**: Aucune
    - **Sortie Attendue**: 2