PLAN DE TEST DU PROJET

JavaScript Unit Testing With Vi-test

LISTE DES PARTICIPANTS :

- AWATI MAFOUO SUZY IVANA : 21T2306
- AMBASSA GUY MATTHIEU : 20U2654
- NGAH NOMO GUY ROGER : 20U2610
- KENMOE SIMO BORIS : 19M2066

SOMMAIRE

1. Introduction générale

Ce document présente le plan de test pour l'application mobile d’un restaurant local. L'application, actuellement en phase de test après une phase initiale de développement, a pour objectif principal de permettre aux utilisateurs de créer un panier de produits et de bénéficier de promotions.

1.1) Objectifs du plan de test :

- Définir la portée des tests et les cas de test à exécuter.
- Identifier les ressources nécessaires et établir un calendrier pour les tests.
- Décrire les critères de réussite et d'échec des tests.
- Assurer la qualité et le bon fonctionnement de l'application avant sa mise en production.

1.2) Portée des tests :

Ce plan de test couvre les tests unitaires des fonctions javascript de l'application, contenues dans les dossiers suivants :

- Basket : Fonctionnalités liées au panier d'achat
- Error-handling : Gestion des erreurs et exceptions au sein de l'application.
- Évents : Gestion des événements utilisateur
- Promotions : Application des promotions, calcul des réductions
- Users : Fonctionnalités liées aux utilisateurs

1.3) Méthodologie :

La méthodologie de test utilisée sera principalement **le test unitaire,** chaque fonction du code source étant testée individuellement pour s'assurer de son bon fonctionnement.

Livrables : Scripts de tests unitaires pour chaque fonction.

2. Calendrier des tests

Nous avons réalisé un exemple de calendrier de test avec des durées fixes :

Phase 1 : Préparation (Semaine 1)

Jour 1-2 : Révision des exigences fonctionnelles et non-fonctionnelles, des spécifications techniques et du plan de test.

Jour 3-4 : Préparation de l'environnement de test, installation des outils et des données de test.

Jour 5 : Réunion d'équipe pour la revue du plan de test et la répartition des tâches.

Phase 2 : Exécution des Tests (Semaines 2-3)

Semaine 2 : Exécution des tests unitaires pour chaque module (events, users, basket, promotion...).

Semaine 3 : Exécution des tests d'intégration pour vérifier l'interaction entre les modules.

Semaine 4 : Exécution des tests fonctionnels pour valider le comportement du système selon les exigences.

Phase 3 : Analyse et Correction (Semaine 4)

Jour 1-3 : Analyse des résultats des tests, identification des anomalies et des bugs.

Jour 4-5 : Correction des anomalies par l'équipe de développement, re test des corrections.

3. Environnement de tests

a. Environnement de Développement :

Éditeur de code : Visual Studio Code (VS Code)

Langage de programmation : JavaScript

Gestionnaire de paquets : npm

Outil de construction : Vite

Framework de test : Vite Test

b. Structure du projet :

js: Contient les codes sources des fonctions à tester.

Test : Contient les fichiers de test.

vitest.config.js et setup.js : Fichier de configuration de Vite.

package.json: Fichier de gestion des dépendances et des scripts.

4. Analyse des Cas de tests

4.1) Fichier User.test.js

a. Classe User :

Objectif : Vérifier la création d'une instance de la classe User avec les propriétés correctes.

Cas de Test :

- it ('should create a new User instance with correct properties'):

Entrée : new User (1, 'test user')
Sortie attendue :
user.id est égal à 1.
user.username est égal à 'testuser'.
user.isPremium est égal à false.
Objectif : Vérifier que les propriétés de l'instance User sont correctement initialisées.
Remarques :
Ce test vérifie la construction de base de la classe User.

b. Fonction userExists:

Objectif : Vérifier le fonctionnement de la fonction userExists pour déterminer si un nom d'utilisateur existe dans le système.

Cas de Test :

- it('should return true if username exists') :

Entrée : 'newuser1@pluralsight.com'

Sortie attendue : true
Objectif : Vérifier que la fonction retourne true si le nom d'utilisateur existe dans la base de données.

- it ('should return false if username does not exist'):

Entrée : 'nonexistentuser@example.com'
Sortie attendue : false
Objectif : Vérifier que la fonction retourne false si le nom d'utilisateur n'existe pas dans la base de données.
Remarques :
Ce test vérifie les deux cas possibles de la fonction userExists: utilisateur existant et utilisateur inexistant.

c. Fonction createUserId:

Objectif : Vérifier le fonctionnement de la fonction createUserId pour générer un identifiant d'utilisateur unique.
Cas de Test :

- it('should return a unique user ID') :

Entrée : Aucune
Sortie attendue : 2 (en supposant que l'identifiant précédent était 1)
Objectif : Vérifier que la fonction génère un identifiant unique qui n'est pas déjà utilisé.

Remarques : Ce test vérifie la génération d'un identifiant unique.

4.2) Fichier basket.test.js

a. calculateTotal

Objectif : Vérifier le calcul du prix total du panier en fonction des articles et d'un éventuel rabais.

Cas de Test :

- it('should return 0 if basketItems is empty'):

Entrée : basketItems = []
Sortie Attendue : 0
Objectif : Vérifier que le prix total est nul si le panier est vide.

- it ('should return the price of the first item if basketItems has only one item'):

Entrée : basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2)]
Sortie Attendue : 100
Objectif : Vérifier que le prix total est égal au prix du seul article du panier multiplié par sa quantité.

- it ('should return the total price of all items in basketItems'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2), new BasketItem(new MockEvent(2, 'Play', 30), 3)]
Sortie Attendue : 190
Objectif : Vérifier que le prix total est la somme des prix de tous les articles du panier multipliés par leurs quantités.

- it ('should return the total price minus the discount if discount is provided'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2), new BasketItem(new MockEvent(2, 'Play', 30), 3)], discount = 10
Sortie Attendue : 180
Objectif : Vérifier que le prix total est réduit du rabais si celui-ci est fourni.

b. showAdverts
Objectif : Vérifier si les publicités doivent être affichées en fonction du statut premium de l'utilisateur.

Cas de Test :

- it('should return false if user is premium') :

Entrée : user = { isPremium: true }
Sortie Attendue : false
Objectif : Vérifier que les publicités ne sont pas affichées si l'utilisateur est premium.

- it('should return true if user is not premium') :

Entrée : user = { isPremium: false}
Sortie Attendue : true
Objectif : Vérifier que les publicités sont affichées si l'utilisateur n'est pas premium.

c. searchBasket

Objectif : Vérifier la recherche d'articles dans le panier en fonction du nom de l'événement.

Cas de Test :

- it ('should return an empty array if basketItems is empty'):

Entrée : basketItems = [], searchQuery = 'Concert'
Sortie Attendue : []
Objectif : Vérifier que la recherche retourne un tableau vide si le panier est vide.

- it ('should return an array containing the matching basketItem if searchQuery matches event name'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2), new BasketItem(new MockEvent(2, 'Play', 30), 3)], searchQuery = 'Concert'
Sortie Attendue : [new BasketItem(new MockEvent(1, 'Concert', 50), 2)]

Objectif : Vérifier que la recherche retourne un tableau contenant l'article correspondant si le nom de l'événement correspond à la requête de recherche.

- it ('should return an array containing all matching basketItems if multiple events match searchQuery'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2), new BasketItem(new MockEvent(2, 'Play', 30), 3), new BasketItem(new MockEvent(3, 'Concert Hall', 40), 1)], searchQuery = 'Concert'
Sortie Attendue : [new BasketItem(new MockEvent(1, 'Concert', 50), 2), new BasketItem(new MockEvent(3, 'Concert Hall', 40), 1)]
Objectif : Vérifier que la recherche retourne un tableau contenant tous les articles correspondant si plusieurs noms d'événements correspondent à la requête de recherche.

- it ('should return an empty array if no event name matches searchQuery'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2), new BasketItem(new MockEvent(2, 'Play', 30), 3)], searchQuery = 'Opera'
Sortie Attendue : []

Objectif : Vérifier que la recherche retourne un tableau vide si aucun nom d'événement ne correspond à la requête de recherche.

d. getBasketItem
Objectif : Vérifier la récupération d'un article du panier en fonction de l'identifiant de l'événement.

Cas de Test :

- it('should return null if basketItems is empty'):

Entrée : basketItems = [], event = new MockEvent(1, 'Concert', 50)
Sortie Attendue : null
Objectif : Vérifier que la fonction retourne null si le panier est vide.

- it ('should return the BasketItem if event.id matches'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2), new BasketItem(new MockEvent(2, 'Play', 30), 3)], event = new MockEvent(1, 'Concert', 50)
Sortie Attendue : new BasketItem(new MockEvent(1, 'Concert', 50), 2)
Objectif : Vérifier que la fonction retourne l'article correspondant si l'identifiant de l'événement correspond.

- it('should return null if no event.id matches'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2), new BasketItem(new MockEvent(2, 'Play', 30), 3)], event = new MockEvent(3, 'Opera', 60)
Sortie Attendue : null
Objectif : Vérifier que la fonction retourne null si aucun identifiant d'événement ne correspond.

e. createBasketItem
Objectif : Vérifier la création d'un nouvel article dans le panier.
Cas de Test :

- it ('should return a new BasketItem if event is not already in basketItems'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2)], event = new MockEvent(2, 'Play', 30), ticketCount = 3
Sortie Attendue : new BasketItem(new MockEvent(2, 'Play', 30), 3)
Objectif : Vérifier que la fonction retourne un nouvel article si l'événement n'est pas déjà dans le panier.

- it ('should return null if event is already in basketItems'):

Entrée: basketItems = [new BasketItem(new MockEvent(1, 'Concert', 50), 2)], event = new MockEvent(1, 'Concert', 50), ticketCount = 3

Sortie Attendue : null
Objectif : Vérifier que la fonction retourne null si l'événement est déjà dans le panier.

f. serializeBasketItemsToJson
Objectif : Vérifier la sérialisation des articles du panier en JSON.
Cas de Test :

- it ('should return an empty array if basketItems is empty'):

Entrée : basketItems = []
Sortie Attendue : []
Objectif : Vérifier que la fonction retourne un tableau vide si le panier est vide.

4.3) Fichier exeption.test.js

Vérifier que les classes d'erreurs personnalisées (InvalidEventNameError, InvalidEventPriceError, InvalidReferralCodeError, InvalidUsernameError, UserHasAccountError) sont correctement instanciées et que le message d'erreur personnalisé est correctement défini.

Cas de Test
3.1) InvalidEventNameError

- **it ('should create an InvalidEventNameError with a custom message'):**

Entrée : errorMessage = 'Invalid event name'
Sortie Attendue : error.message = 'Invalid event name'

3.2) InvalidEventPriceError

- **it ('should create an InvalidEventPriceError with a custom message'):**

Entrée : errorMessage = 'Invalid event price'
Sortie Attendue : error.message = 'Invalid event price'

3.3) InvalidReferralCodeError

- **it ('should create an InvalidReferralCodeError with a custom message'):**

Entrée : errorMessage = 'Invalid referral code'
Sortie Attendue: error.message = 'Invalid referral code'

3.4) InvalidUsernameError

- **it ('should create an InvalidUsernameError with a custom message'):**

Entrée : errorMessage = 'Invalid username'
Sortie Attendue : error.message = 'Invalid username'

3.5) UserHasAccountError

- **it ('should create a UserHasAccountError with a custom message'):**

Entrée: errorMessage = 'User already has an account'
Sortie Attendue : error.message = 'User already has an account'

Remarques

Chaque test utilise une assertion expect(error.message).toBe(errorMessage) pour vérifier que le message d'erreur de l'instance de

l'erreur est bien égal au message personnalisé fourni lors de l'instanciation.

4.4) Fichier Exchange.test.js

Objectif : Vérifier le bon fonctionnement de la fonction getExchangeRate pour obtenir le taux de change correct pour différentes devises et gérer les cas d'erreur.

Cas de Test

- it ('should return the correct exchange rate for USD')

Entrée: currencyCode = 'USD', mockCallback = vi.fn()

Sortie Attendue : mockCallback est appelé avec l'objet { originalCurrency: 'GBP', newCurrency: 'USD', exchangeRate: 1.25}

Objectif : Vérifier que le taux de change correct est retourné pour la devise USD.

- it ('should return the correct exchange rate for EUR')

Entrée : currencyCode = 'EUR', mockCallback = vi.fn()

Sortie Attendue : mockCallback est appelé avec l'objet { originalCurrency: 'GBP', newCurrency: 'EUR', exchangeRate: 1.18}

Objectif : Vérifier que le taux de change correct est retourné pour la devise EUR.

- it ('should return the correct exchange rate for NZD')

Entrée : currencyCode = 'NZD', mockCallback = vi.fn()

Sortie Attendue : mockCallback est appelé avec l'objet { originalCurrency: 'GBP', newCurrency: 'NZD', exchangeRate: 1.93}

Objectif : Vérifier que le taux de change correct est retourné pour la devise NZD.

- it ('should throw an error if the currency is not supported')

Entrée : currencyCode = 'JPY', mockCallback = vi.fn()

Sortie Attendue : Une erreur 'Currency not supported' est levée.

Objectif : Vérifier que la fonction gère correctement les devises non prises en charge.

Remarques

Les tests utilisent vi.fn() pour simuler la fonction de rappel mockCallback et expect(mockCallback).toHaveBeenCalledWith(...) pour vérifier que la fonction a été appelée avec les bons arguments.Le test de la gestion des erreurs utilise await expect(getExchangeRate(currencyCode, mockCallback)).rejects.toThrowError(...) pour vérifier que la fonction lève une erreur spécifique.

4.5) Fichier promotions.test.js

Cas de Test

a. calculatePercentageDiscount

- **it ('should apply the discount if the current price meets the minimum spend')**

Entrée : percentage = 10, minimumSpend = 50, currentPrice = 60

Sortie Attendue : discountedPrice = 5.4

Objectif : Vérifier que la réduction est appliquée correctement si le prix actuel atteint le minimum de dépenses.

- **it ('should not apply the discount if the current price is below the minimum spend')**

Entrée : percentage = 10, minimumSpend = 50, currentPrice = 40

Sortie Attendue : discountedPrice = 40

Objectif : Vérifier que la réduction n'est pas appliquée si le prix actuel est inférieur au minimum de dépenses.

b. calculateMoneyOff

- **it ('should apply the discount if the current price meets the minimum spend')**

Entrée : discount = 5, minimumSpend = 50, currentPrice = 60

Sortie Attendue : discountedPrice = 55

Objectif : Vérifier que la réduction est appliquée correctement si le prix actuel atteint le minimum de dépenses.

- **it ('should not apply the discount if the current price is below the minimum spend')**

Entrée : discount = 5, minimumSpend = 50, currentPrice = 40

Sortie Attendue : discountedPrice = 40

Objectif : Vérifier que la réduction n'est pas appliquée si le prix actuel est inférieur au minimum de dépenses.

c. generateReferralCode

- **it('should generate a unique referral code')**

Entrée : userId = 123

Sortie Attendue : referralCode correspond au format #FRIEND-#\d {3} -#123

Objectif : Vérifier que la fonction génère un code de parrainage unique avec le format correct.

d. applyDiscount

- **it ('should apply a MONEYOFF discount if the discount code is valid')**

Entrée: discountCode = 'TEST-MONEYOFF', currentTotal = 100, mockDiscountData = { isValid: true, type: 'MONEYOFF', value: 10, minSpend: 50 }

Sortie Attendue : discountedTotal = 90

Objectif : Vérifier que la réduction MONEYOFF est appliquée correctement si le code de réduction est valide.

- **it ('should apply a PERCENTAGEOFF discount if the discount code is valid')**

Entrée: discountCode = 'TEST-PERCENTAGEOFF', currentTotal = 100, mockDiscountData = { isValid: true, type: 'PERCENTAGEOFF', value: 10, minSpend: 50}

Sortie Attendue : discountedTotal = 90

Objectif : Vérifier que la réduction PERCENTAGEOFF est appliquée correctement si le code de réduction est valide.

- **it ('should return the original total if the discount code is invalid')**

Entrée: **discountCode = 'INVALID-CODE', currentTotal = 100, mockDiscountData = { isValid: false}**

Sortie Attendue : discountedTotal = 100

Objectif : Vérifier que le total original est retourné si le code de réduction est invalide.

4.6) Fichier event.test.js

Cas de Test

a. isSoldOut

- it('should return true if ticketsRemaining is 0')

Entrée: event = new Event(1, 'Concert', 10, 10, 0, new Date())

Sortie Attendue : isSoldOut(event) = true

Objectif : Vérifier que la fonction retourne true si le nombre de billets restants est égal à 0.

- it ('should return false if ticketsRemaining is not 0')

Entrée: event = new Event(1, 'Concert', 10, 10, 5, new Date())

Sortie Attendue : isSoldOut(event) = false

Objectif : Vérifier que la fonction retourne false si le nombre de billets restants est différent de 0.

b. getTagLine

- it ('should return "Event Sold Out!" if event is sold out')

Entrée: event = new Event (1, 'Concert', 10, 10, 0, new Date ()), minimumTicketCount = 5, isPopular = false

Sortie attendue : getTagLine(event, minimumTicketCount, isPopular) = 'Event Sold Out!'

Objectif : Vérifier que le slogan correct est retourné si l'événement est épuisé.

- it ('should return "Hurry only X tickets left!" if ticketsRemaining is less than minimumTicketCount')

Entrée: event = new Event (1, 'Concert', 10, 10, 2, new Date ()), minimumTicketCount = 5, isPopular = false

Sortie Attendue : getTagLine(event, minimumTicketCount, isPopular) = 'Hurry only 2 tickets left!'

Objectif : Vérifier que le slogan correct est retourné si le nombre de billets restants est inférieur au minimum.

- it ('should return "Hurry only 1 ticket left!" if ticketsRemaining is 1')

Entrée: event = new Event (1, 'Concert', 10, 10, 1, new Date ()), minimumTicketCount = 5, isPopular = false

Sortie Attendue : getTagLine(event, minimumTicketCount, isPopular) = 'Hurry only 1 ticket left!'

Objectif : Vérifier que le slogan correct est retourné si le nombre de billets restants est égal à 1.

- it('should return "This Event is getting a lot of interest. Don\'t miss out, purchase your ticket now!" if event is popular and ticketsRemaining is greater than minimumTicketCount')

Entrée : event = new Event(1, 'Concert', 10, 10, 10, new Date()), minimumTicketCount = 5, isPopular = true

Sortie Attendue : getTagLine(event, minimumTicketCount, isPopular) = 'This Event is getting a lot of interest. Don\'t miss out, purchase your ticket now!'

Objectif : Vérifier que le slogan correct est retourné si l'événement est populaire et le nombre de billets restants est supérieur au minimum.

- it ('should return "Don\'t miss out, purchase your ticket now!" if event is not popular and

ticketsRemaining is greater than minimumTicketCount')

Entrée: event = new Event (1, 'Concert', 10, 10, 10, new Date ()), minimumTicketCount = 5, isPopular = false

Sortie Attendue : getTagLine(event, minimumTicketCount, isPopular) = 'Don\'t miss out, purchase your ticket now!'

Objectif : Vérifier que le slogan correct est retourné si l'événement n'est pas populaire et le nombre de billets restants est supérieur au minimum.

c. createEvent

- it('should create a new Event object with valid parameters')

Entrée : name = 'Concert', price = 10, availableTickets = 10

Sortie Attendue : event est une instance de Event, event.name = 'Concert', event.ticketPrice = 10, event.totalTickets = 10

Objectif : Vérifier que la fonction crée un nouvel objet Event avec les paramètres valides.

- it ('should throw an InvalidEventNameError if name is not a string or exceeds 200 characters')

Entrée : name = 123, price = 10, availableTickets = 10

Sortie Attendue : Une erreur InvalidEventNameError est levée.

Objectif : Vérifier que la fonction lève une erreur si le nom n'est pas une chaîne de caractères ou dépasse 200 caractères.

- it ('should throw an InvalidEventPriceError if price is not a number or less than 0')

Entrée : name = 'Concert', price = '10', availableTickets = 10

Sortie Attendue : Une erreur InvalidEventPriceError est levée.

Objectif : Vérifier que la fonction lève une erreur si le prix n'est pas un nombre ou est inférieur à 0.

- it ('should throw an InvalidEventPriceError if availableTickets is not a number or less than 1')

Entrée : name = 'Concert', price = 10, availableTickets = '10'

Sortie Attendue : Une erreur InvalidEventPriceError est levée.

Objectif : Vérifier que la fonction lève une erreur si le nombre de billets disponibles n'est pas un nombre ou est inférieur à 1.

4.7) Fichier filters.test.js

Vérifier le bon fonctionnement des fonctions de filtrage des événements, notamment today, next7Days et next30Days.

Cas de Test

a. today

- it('should return true if event date is today')

Entrée : event = new Event(1, 'Concert', 10, 10, 10, todayDate) (où todayDate est la date d'aujourd'hui)
Sortie Attendue : today(event) = true
Objectif : Vérifier que la fonction retourne true si la date de l'événement est aujourd'hui.

- it ('should return false if event date is not today')

Entrée : event = new Event (1, 'Concert', 10, 10, 10, yesterdayDate) (où yesterdayDate est la date d'hier)
Sortie Attendue : today(event) = false
Objectif : Vérifier que la fonction retourne false si la date de l'événement n'est pas aujourd'hui.

b**. next7Days**

- it ('should return true if event date is within the next 7 days')

Entrée : event = new Event (1, 'Concert', 10, 10, 10, tomorrowDate) (où tomorrowDate est la date de demain)
Sortie Attendue : next7Days(event) = true
Objectif : Vérifier que la fonction retourne true si la date de l'événement est dans les 7 prochains jours.

- it ('should return false if event date is more than 7 days in the future')

Entrée : event = new Event (1, 'Concert', 10, 10, 10, nextWeekDate) (où nextWeekDate est la date dans 8 jours)
Sortie Attendue : next7Days(event) = false

Objectif : Vérifier que la fonction retourne false si la date de l'événement est plus de 7 jours dans le futur.

- it ('should return false if event date is in the past')

Entrée : event = new Event(1, 'Concert', 10, 10, 10, yesterdayDate) (où yesterdayDate est la date d'hier)
Sortie Attendue : next7Days(event) = false
Objectif : Vérifier que la fonction retourne false si la date de l'événement est dans le passé.

3. next30Days

- it ('should return true if event date is within the next 30 days')

Entrée : event = new Event(1, 'Concert', 10, 10, 10, nextMonthDate) (où nextMonthDate est la date dans 15 jours)
Sortie Attendue : next30Days(event) = true
Objectif : Vérifier que la fonction retourne true si la date de l'événement est dans les 30 prochains jours.

- it('should return false if event date is more than 30 days in the future')

Entrée : event = new Event (1, 'Concert', 10, 10, 10, nextMonthDate) (où nextMonthDate est la date dans 31 jours)
Sortie Attendue : next30Days(event) = false
Objectif : Vérifier que la fonction retourne false si la date de l'événement est plus de 30 jours dans le futur.

- it ('should return false if event date is in the past')

Entrée : event = new Event (1, 'Concert', 10, 10, 10, yesterdayDate) (où yesterdayDate est la date d'hier)
Sortie Attendue : next30Days(event) = false
Objectif : Vérifier que la fonction retourne false si la date de l'événement est dans le passé.

4.8) Fonction search.test.js

Vérifier le bon fonctionnement de la fonction getEvents, qui filtre une liste d'événements en fonction d'un prédicat de recherche.

Cas de Test

- .it ('should return an empty array if no events are provided')

Entrée : events = [], searchPredicate = () => true (prédicat toujours vrai)
Sortie Attendue : filteredEvents = [] (tableau vide)
Objectif : Vérifier que la fonction retourne un tableau vide si aucun événement n'est fourni.

- it ('should return all events if the search predicate is always true')

Entrée :
events = [new Event(1, 'Concert 1', 10, 10, 10, new Date()), new Event(2, 'Concert 2', 20, 20, 20, new Date()),]
searchPredicate = () => true (prédicat toujours vrai)

Sortie Attendue : filteredEvents = events (le tableau d'événements d'entrée)
Objectif : Vérifier que la fonction retourne tous les événements si le prédicat de recherche est toujours vrai.

- it ('should return a filtered array of events based on the search predicate')

Entrée :

events = [new Event(1, 'Concert 1', 10, 10, 10, new Date()), new Event(2, 'Concert 2', 20, 20, 20, new Date()), new Event(3, 'Theater Show', 30, 30, 30, new Date()),]
searchPredicate = (event) => event.name.includes('Concert') (prédicat qui vérifie si le nom de l'événement contient "Concert")

Sortie Attendue :

filteredEvents = [new Event(1, 'Concert 1', 10, 10, 10, new Date()), new Event(2, 'Concert 2', 20, 20, 20, new Date()),]

Objectif : Vérifier que la fonction retourne un tableau filtré d'événements en fonction du prédicat de recherche.

5. Risques et Enjeux

Risques et Enjeux	Description	Impact potentiel	Stratégies d'atténuation
Couverture de code insuffisante	Les tests ne couvrent pas tous les cas d'utilisation et les scénarios possibles pour	Des bugs peuvent passer inaperçus, ce qui pourrait affecter la stabilité et la fiabilité	Utiliser des outils de couverture de code pour mesurer la couvertur

	chaque fichier.	de l'application.	e des tests, prioriser les tests des fonctions critiques
Tests fragiles et dépendants	Les tests sont trop dépendants de l'implémentation interne des fonctions, ce qui les rend fragiles et difficiles à maintenir	Les tests peuvent échouer à cause de changements mineurs dans le code, ce qui peut ralentir le développement et créer des faux positifs.	Utiliser des mocks et des stubs pour isoler les dépendances, écrire des tests qui vérifient le comportement externe des fonctions

- Risques et Enjeux spécifiques aux fichiers :

user.test.js : on note le risque de ne pas tester les validations et les autorisations des utilisateurs, ce qui pourrait mener à des problèmes de sécurité.
basket.test.js : ne pas tester les calculs du panier, les promotions et les taxes pourrait mener à des erreurs de facturation.

exception.test.js : on a le risque de ne pas tester tous les types d'exceptions et leurs traitements, ce qui pourrait mener à des erreurs non gérées.
exchange.test.js : ne pas tester toutes les conversions de devises et les taux de change.
promotion.test.js : nous avons le Risque de ne pas tester toutes les conditions et les applications des promotions
event.test.js : Risque de ne pas tester la gestion des événements et les interactions avec les utilisateurs.
filter.test.js : ne pas tester tous les types de filtres et leurs combinaisons, peut conduire à des résultats de recherche incorrects.
search.test.js : on note le Risque de ne pas tester les algorithmes de recherche et les résultats, qui pourrait mener à des résultats incomplets ou inexacts.

6. Conclusion

En conclusion le projet de développement d'une application mobile pour d'un restaurant a nécessité une

attention particulière portée à la phase de test, cruciale pour garantir la qualité et la fiabilité du produit final. Notre plan de test, axé sur des tests unitaires complets des fonctions Javascript de l'application, a permis d'identifier et de corriger les erreurs potentielles avant la mise en production. La méthodologie rigoureuse adoptée, combinée à l'utilisation d'outils performants comme Vite Test, a garanti l'efficacité et la rapidité d'exécution des tests. Les résultats obtenus à l'issue de cette phase de test ont permis de valider le bon fonctionnement de l'application et son adéquation aux besoins des utilisateurs.