

Plan de test ICT304

Auteurs :

- **CHIMI KAMLO ROMARIC 21Q2453**
- **CHOUALA NODEM BOREL 21U2343**
- **NKONKAM WANDJI DAULIN CLAIR 21Q2435**
- **TCHEUTCHOUA CHRIST ORPHE 21Q2396**
- **ACHOTIA BORIS 20R2335**

Objectif

Ce document a pour objectif de définir les étapes et les procédures de test unitaire pour l'application mobile du restaurant, en utilisant le Framework Jets et suivant le modèle Arrange-Act-Assert (A-A-A). Les tests couvriront 04 sous-dossiers : basket, événements, promotions, et users.

Vue d'ensemble de l'application

L'application mobile permet aux utilisateurs de sélectionner des produits à partir d'un menu, de les ajouter à un panier, de consulter et d'appliquer des promotions, et de passer des commandes. Elle inclut également des fonctionnalités de gestion de profil et de suivi des commandes.

Objectifs des tests

Les objectifs des tests unitaires sont de :

- Vérifier que chaque fonction individuelle de l'application fonctionne correctement en isolation.
- Assurer que les composants de base sont fiables et robustes.
- De détecter et corriger les bogues précocement dans le cycle de développement.

- Faciliter les modifications futures en garantissant que les nouvelles modifications n'introduisent pas de régressions.

Approche des tests

Méthodes et techniques de test

- Utilisation de tests unitaires pour vérifier la logique des fonctions isolées.
- Adoption du modèle A-A-A pour structurer les tests de manière cohérente :
 - Arrange : Préparer les données et l'état initial nécessaires pour le test.
 - Act : Exécuter la fonction ou le module à tester.
 - Assert : Vérifier que le résultat obtenu correspond au résultat attendu.

Rôles et responsabilités

- Développeurs : ils sont responsables de l'écriture des tests unitaires.
- Testeurs : ils sont responsables de l'exécution des tests et de la validation des résultats.
- Chef de projet de test : Les chefs de projet sont responsables de la supervision et de la communication des résultats de test.

Calendrier des tests

Phase	Début	Fin	Durée
Planification	20 juin 2024	21 juin 2024	2 jours
Conception	22 juin 2024	24 juin 2024	3 jours
Exécution	25 juin 2024	26 juin 2024	2 jours
Reporting	27 juin 2024	28 juin 2024	1 jour

Environnement de test

- **Configuration requise:**

- Matériel: Ordinateurs avec les spécifications standard de développement.
- Logiciels: Node.js, Vitest, et les modules nécessaires pour les fonctions de promotion.
- Configurations réseau: Accès au dépôt de code et aux services de gestion des versions (GitHub).

Données de test

- **Source des données:**

- Les données de test seront générées en interne.

- **Type de données:**

- Valeurs pour les prix actuels, pourcentages de réduction, montants de réduction, identifiants d'utilisateur pour les codes de parrainage, codes de devise.

- **Contraintes:**

- Les données de test doivent couvrir une gamme de scénarios, y compris les limites inférieures et supérieures des valeurs acceptables.

Cas de test du sous-dossier : promotions

Fonction	ID	Cas de test	Donnees en entree	Résultat attendu	Statut
calculatePercentageDiscount	CPD-01	Retourne le prix réduit lorsque currentPrice est supérieur à minimumSpend.	{ discountPercentage: 20, currentPrice: 100, minimumSpend: 150 }	120 (pour une réduction de 20% sur 150).	Passé

calculatePercentageDiscount	CPD-02	Retourne le prix original lorsque <code>currentPrice</code> est inférieur à <code>minimumSpend</code> .	{ discountPercentage: 20, currentPrice: 100, minimumSpend: 80 }	80 (pour une réduction de 20% sur 80).	Passé
calculateMoneyOff	CMO-01	Retourne le prix après réduction lorsque <code>currentPrice</code> est supérieur à <code>minimumSpend</code> .	{ moneyOffAmount: 30, currentPrice: 100, minimumSpend: 150 }	120 (pour une réduction de 30 sur 150).	Passé
calculateMoneyOff	CMO-02	Retourne le prix original lorsque <code>currentPrice</code> est inférieur à <code>minimumSpend</code> .	{ moneyOffAmount: 30, currentPrice: 100, minimumSpend: 80 }	80 (pour une réduction de 30 sur 80).	Passé
generateReferralCode	GRC-01	Génère un code de parrainage contenant <code>userId</code> .	{ userId: '12345' }	Doit correspondre à la regex <code>#FRIEND-#\d{3}-#12345</code> .	Passé
applyDiscount	AD-01	Applique une réduction monétaire lorsque le type de réduction est <code>MONEYOFF</code> .	{ discountCode: 'DISCOUNT20', currentPrice: 150 }	130 (pour une réduction de 20 sur 150).	Passé
applyDiscount	AD-02	Applique une réduction en pourcentage lorsque le type de réduction est <code>PERCENTAGEOFF</code> .	{ discountCode: 'DISCOUNT20', currentPrice: 150 }	120 (pour une réduction de 20% sur 150).	Passé

applyDiscount	AD-03	Retourne le prix original lorsque la réduction n'est pas valide.	{ discountCode: 'DISCOUNT20', currentPrice: 150 }	150 (pour un code de réduction invalide).	Passé
applyDiscount	AD-04	Retourne le prix original lorsque le type de réduction est inconnu.	{ discountCode: 'DISCOUNT20', currentPrice: 150 }	150 (pour un type de réduction inconnu).	Passé
getExchangeRate	GER-01	Retourne le taux de change correct pour USD.	'USD'	{ originalCurrency: 'GBP', newCurrency: 'USD', exchangeRate: 1.25}.	Passé
getExchangeRate	GER-02	Retourne le taux de change correct pour EUR.	'EUR'	{ originalCurrency: 'GBP', newCurrency: 'EUR', exchangeRate: 1.18 }.	Passé
getExchangeRate	GER-03	Retourne le taux de change correct pour NZD.	'NZD'	{ originalCurrency: 'GBP', newCurrency: 'NZD', exchangeRate: 1.93}.	Passé
getExchangeRate	GER-04	Génère une erreur pour une devise non supportée.	'AUD'	Erreur Currency not supported.	Passé

Cas de test du sous-dossier : basket

Fonction	ID	Donnees en entree	Cas de test	Résultat attendu	Statut
----------	----	-------------------	-------------	------------------	--------

calculateTotal		<pre> basketItems=[] </pre>	Should return 0 if basket is empty	0	Passé
calculateTotal		<pre> basketItems: [new BasketItem({ ticketPrice: 50 }, 1)] </pre>	Should return the price of the single item in the basket	50	Passé
calculateTotal		<pre> basketItems: [new BasketItem({ ticketPrice: 50 }, 1), new BasketItem({ ticketPrice: 30 }, 1)] </pre>	Should return the sum of prices of all items in the basket	80	Passé
calculateTotal		<pre> basketItems: [new BasketItem({ ticketPrice: 50 }, 1), new BasketItem({ ticketPrice: 30 }, 1)], discount: 10 </pre>	Should apply the discount if provided	70	Passé
calculateTotal		<pre> basketItems: [new BasketItem({ ticketPrice: 50 }, 1), new BasketItem({ ticketPrice: 30 }, 1)], discount: 80 </pre>	Should return 0 if discount is equal to total price	0	Passé
calculateTotal		<pre> basketItems: [new BasketItem({ ticketPrice: 50 }, 1), new BasketItem({ ticketPrice: 30 }, 1)], discount: 100 </pre>	Should handle negative total price after discount	-20	Passé
showAdverts		<pre> user: { isPremium: true } </pre>	Should return false if the	false	Passé

			user is premium		
showAdverts		user: { isPremium: true }	Should return true if the user is not premium	true	Passé
searchBasket		basketItems: [new BasketItem({ name: 'Concert', ticketPrice: 50 }, 1), new BasketItem({ name: 'Rock Concert', ticketPrice: 60 }, 1)], searchQuery: 'concert'	Should return an empty array if basket is empty	[]	Passé
searchBasket		basketItems: [new BasketItem({ name: 'Concert', ticketPrice: 50 }, 1), new BasketItem({ name: 'Rock Concert', ticketPrice: 60 }, 1)], searchQuery: 'concert'	Should return an empty array if no items match the search query	[]	Passé
searchBasket		basketItems: [new BasketItem({ name: 'Concert', ticketPrice: 50 }, 1), new BasketItem({ name: 'Rock Concert', ticketPrice: 60 }, 1)], searchQuery: 'concert'	should return matching items based on the search query	[new BasketItem({ name: 'Concert', ticketPrice: 50 }, 1)]	Passé
searchBasket		basketItems: [new	Should return all matching	[new BasketItem({	Passé

		<pre> BasketItem({ name: 'Concert', ticketPrice: 50 }, 1), new BasketItem({ name: 'Rock Concert', ticketPrice: 60 }, 1)], searchQuery: 'concert </pre>	items for a partial search query	<pre> name: 'Concert', ticketPrice: 50 }, 1), new BasketItem({ name: 'Rock Concert', ticketPrice: 60 }, 1)] </pre>	
searchBasket		<pre> basketItems: [new BasketItem({ name: 'Concert', ticketPrice: 50 }, 1), new BasketItem({ name: 'Rock Concert', ticketPrice: 60 }, 1)], searchQuery: 'concert </pre>	Should be case insensitive in search	<pre> [new BasketItem({ name: 'Concert', ticketPrice: 50 }, 1)] </pre>	Passé
getBasketItem		<pre> basketItems: [new BasketItem({ id: 1, name: 'Concert' }, 1), new BasketItem({ id: 2, name: 'Football Match' }, 2)], event: { id: 2, name: 'Football Match' } </pre>	Should return null if the basket is empty	null	Passé
getBasketItem		<pre> basketItems: [new BasketItem({ id: 1, name: 'Concert' }, 1), new BasketItem({ id: 2, name: 'Football Match' }, 2)], event: { id: 2, name: </pre>	Should return null if the event is not in the basket	null	Passé

		'Football Match' }			
getBasketItem		<pre> basketItems: [new BasketItem({ id: 1, name: 'Concert' }, 1), new BasketItem({ id: 2, name: 'Football Match' }, 2)], event: { id: 2, name: 'Football Match' } </pre>	Should return the basket item if the event is in the basket	new BasketItem({ id: 1, name: 'Concert' }, 1)	Passé
getBasketItem		<pre> basketItems: [new BasketItem({ id: 1, name: 'Concert' }, 1), new BasketItem({ id: 2, name: 'Football Match' }, 2)], event: { id: 2, name: 'Football Match' } </pre>	Should return the correct basket item when multiple items are in the basket	new BasketItem({ id: 2, name: 'Football Match' }, 2)	Passé
createBasketItem		<pre> basketItems: [new BasketItem({ name: 'Pizza Margherita', id: 1 }, 1), new BasketItem({ name: 'Salade César', id: 2 }, 1)], newItem: { name: 'Burger', id: 3, ticketPrice: 8 } </pre>	Should serialize basket items to JSON format	[{ event: { id: 1, name: 'Event 1', ticketPrice: 50 }, ticketCount: 2 }, { event: { id: 2, name: 'Event 2', ticketPrice: 75 }, ticketCount: 1 }]	Passé
serializeBasketItemsToJson		<pre> basketItems: [new BasketItem({ id: 1, name: 'Event 1', ticketPrice: 50 }, 2), </pre>	Should handle empty basketItems array	[]	Passé

		new BasketItem({ id: 2, name: 'Event 2', ticketPrice: 75 }, 1)]			
serializeBasketItemsToJson		basketItems: [new BasketItem({ id: 1, name: 'Event 1', ticketPrice: 50 }, 2), new BasketItem({ id: 2, name: 'Event 2', ticketPrice: 75 }, 1)]	Should handle basketItems with null or undefined values	[{ event: { id: 1, name: 'Event 1', ticketPrice: 50 }, ticketCount: 2 }, null, undefined, { event: { id: 2, name: 'Event 2', ticketPrice: 75 }, ticketCount: 1 }]	Passé

Cas de test du sous-dossier : events

- event.js

fonction	I D	Cas de test	Donnees en entree	Résultat attendu	Stat ut
Event class		Should create an Event instance with correct properties	id, name, ticketPrice, totalTickets, ticketsRemaining, date	event.id = 1, event.name = "Concert", event.ticketPrice = 50, event.totalTickets = 100, event.ticketsRemai ning = 50, event.date instanceof Date	Pass é
isSoldOut	1	Should return true if ticketsRemaining is 0	event	true	Pass é
isSoldOut	2	Should return false if ticketsRemaining is not 0	event	false	Pass é
getTagLine	1	Should return "Event Sold Out!" if event is sold out	event, minimumTicketCo unt, isPopular	"Event Sold Out!"	Pass é

getTagLine	2	Should return correct message if tickets remaining are less than minimumTicketCount	event, minimumTicketCount, isPopular	"Hurry only 5 tickets left!"	Passé
getTagLine	3	Should return popular event message if event is popular	event, minimumTicketCount, isPopular	"This Event is getting a lot of interest. Don't miss out, purchase your ticket now!"	Passé
getTagLine	4	Should return general message if event is not popular	event, minimumTicketCount, isPopular	"Don't miss out, purchase your ticket now!"	Passé
createEvent	1	Should throw InvalidEventNameError if name is not a string or exceeds 200 characters	name, price, availableTickets	throw InvalidEventNameError	Passé
createEvent	2	Should throw InvalidEventPriceError if price is not a number or is negative	name, price, availableTickets	throw InvalidEventPriceError	Passé
createEvent	3	Should throw InvalidEventPriceError if availableTickets is not a number or less than 1	name, price, availableTickets	throw InvalidEventPriceError	Passé
createEvent	4	Should create an Event instance if all parameters are valid	name, price, availableTickets	<pre> event instanceof Event, event.name = "Concert", event.ticketPrice = 50, event.totalTickets = 100 </pre>	Passé

- **filter.js**

Fonction	ID	Cas de test	Donnees en entree	Résultat attendu	statut
today	1	Should return true if event date is today	{ date: new Date() }	true	Passé
today	2	Should return false if event	{ date: new Date(Date.now() + 86400000) }	false	Passé

		date is not today			
next7Days	1	Should return true if event date is within the next 7 days	{ date: new Date(Date.now() + 3 * 86400000) }	true	Passé
next7Days	2	Should return false if event date is beyond the next 7 days	{ date: new Date(Date.now() + 10 * 86400000) }	false	Passé
next30Days	1	Should return true if event date is within the next 30 days	{ date: new Date(Date.now() + 15 * 86400000) }	true	Passé
next30Days	2	Should return false if event date is beyond the next 30 days	{ date: new Date(Date.now() + 40 * 86400000) }	false	Passé

- **search.js**

Fonction	ID	Cas de test	Donnees en entree	Résultat attendu	statut
getEvents	1	Should filter events based on the search predicate	events = [{ name: "Concert", date: new Date() }, { name: "Festival", date: new Date(Date.now() + 86400000) }, { name: "Conference", date: new Date(Date.now() + 7 * 86400000) }]	filteredEvents.length = 1 filteredEvents[0].name = "Concert"	Passé
getEvents	2	Should return all events if predicate always returns true	events = [{ name: "Concert", date: new Date() }, { name: "Festival", date: new Date(Date.now() + 86400000) }, {	filteredEvents.length = events.length	Passé

			name: "Conference", date: new Date(Date.now() + 7 * 86400000) } }		
getEvents	3	Should return no events if predicate always	events = [{ name: "Concert", date: new Date() }, { name: "Festival", date: new Date(Date.now() + 86400000) }, { name: "Conference", date: new Date(Date.now() + 7 * 86400000) } }	filteredEvents.length = 0	Passé

Cas de test du sous-dossier : users

- **purchaseHistory.js**

Fonction	ID	Cas de test	Donnees en entree	Résultat attendu	statut
getPurchaseHistory	1	Should return a promise that resolves to an array of Purchase objects	userId = '123' mockPurchaseData = [{ event: 'Punk Goes Pop - 90s', tickets: 2, price: 40.00 }, { event: 'Adventures Live!', tickets: 5, price: 120.00 }, { event: 'Folk dance party!', tickets: 3, price: 75.00 }]	result = [new Purchase('Punk Goes Pop - 90s', 2, 40.00), new Purchase('Adventures Live!', 5, 120.00), new Purchase('Folk dance party!', 3, 75.00)]	Passé
getPurchaseHistory	2	Should transform the purchase data into an array of	purchaseData = [{ event: 'Punk Goes Pop - 90s', tickets: 2, price: 40.00 }, { event: 'Adventures Live!', tickets: 5, price: 120.00 }, {	result = [new Purchase('Punk Goes Pop - 90s', 2, 40.00), new Purchase('Adventures Live!', 5, 120.00), new	Passé

		Purchase objects	event: 'Folk dance party!', tickets: 3, price: 75.00 }}	Purchase('Folk dance party!', 3, 75.00)]	
--	--	------------------	---	--	--

- **account.js**

Fonction	ID	Cas de test	Donnees en entree	Résultat attendu	statut
isValidUserName	1	Should return false for an empty username	username = ''	result = false	Passé
isValidUserName	2	Should return false for a username without "@"	username = 'invaliduser.com'	result = false	Passé
isValidUserName	3	Should return true for a valid username	username = 'validuser@example.com'	result = true	Passé
isValidUserName	4	Should return false for a null username	username = null	result = false	Passé

Risques ET problèmes

- **Risques potentiels:**
 - Échec des tests dû à des changements non prévus dans les dépendances.
 - Données de test incorrectes ou insuffisantes.
- **Stratégies d'atténuation:**
 - Validation des données de test avant l'exécution.
 - Révisions de code régulières pour détecter les changements potentiels.

Rapports ET communication

- **Rapports de résultats:**
 - Fréquence: Quotidienne pendant la phase d'exécution des tests.
 - Format: Rapport de test détaillé avec les résultats des cas de test.
 - Parties prenantes: Développeurs, testeurs, chefs de projet.

Conclusion

- **Résumé:** Ce plan de test, utilisant le modèle A-A-A avec Jest, fournit une méthodologie rigoureuse pour assurer la qualité et la fiabilité de l'application mobile du restaurant. En suivant ce plan, l'équipe de développement pourra identifier et corriger les problèmes rapidement, garantissant une application robuste et prête pour le lancement.
- **Recommandations:** S'assurer que toutes les parties prenantes sont informées des résultats de test et que les tests sont exécutés dans un environnement contrôlé pour garantir la validité des résultats.