Stratégie de test : Concevez une stratégie de test

Scénarios prévus

SC-1, création de comptes utilisateurs : Vérifie si l'on accède bien au formulaire d'inscription et vérifie son fonctionnement normal.

SC-2, Tester limites du formulaire d'inscription : Teste les cas extrêmes d'utilisation du formulaire d'inscription et vérifie qu'ils sont correctement gérés et que l'on ne valide pas l'inscription si certains champs ne sont pas remplis correctement.

SC-3, tester l'inscription d'homonymes : Permet de vérifier si dans son fonctionnement actuel, le formulaire d'inscription permet de gérer les homonymes. Si le test échoue, il faudra alors revoir le fonctionnement du formulaire.

SC-4, tester la connexion : Une fois qu'un compte a été créé, vérifie que l'on peut se connecter en mettant les bons identifiants. Vérifie également que la connexion échoue si l'on met les mauvais identifiants.

SC-5, tester l'authentification à deux facteurs : Vérifie le fonctionnement de l'authentification à double facteur. Vérifie aussi que l'on ne puisse pas se connecter si l'authentification à double facteur échoue.

SC-6, tester l'authentification biométrique : Vérifie le fonctionnement de l'authentification biométrique. Vérifie aussi que l'on ne puisse pas se connecter si l'authentification biométrique échoue.

SC-7, test API infos : Vérifie que l'API des infos bancaires fonctionne correctement avec les bons identifiants et que l'API renvoie une erreur si l'on effectue une requête avec les mauvais identifiants. Vérifie aussi le fonctionnement de l'application si l'API est inaccessible.

SC-8, test affichage infos tableau de bord : Vérifie que le tableau de bord (sans les transactions) affiche les bonnes informations.

SC-9, test API relevé : Vérifie que l'API qui permet d'obtenir les relevés fonctionne correctement avec les bons identifiants et que l'API renvoie une erreur si l'on effectue une requête avec les mauvais identifiants. Vérifie aussi le fonctionnement de l'application si l'API est inaccessible.

SC-10, tester téléchargement relevé : Permet de vérifier que l'on arrive bien à télécharger le relevé via l'application.

SC-11, test API transaction : Vérifie que l'API qui permet d'obtenir les informations de transactions fonctionne correctement avec les bons identifiants et que l'API renvoie une erreur si l'on effectue une requête avec les mauvais identifiants. Vérifie aussi le fonctionnement de l'application si l'API est inaccessible.

SC-12, test affichage transactions : Vérifie sur le tableau que les informations des transactions sont bien présentes et que l'on peut bien changer la période d'affichage des transactions.

SC-13, test rafraîchissement transactions : Test si l'on envoie bien une requête pour obtenir les dernières informations lorsque l'on clique sur le bouton rafraîchir.

SC-14, vérifier contenu transactions : Vérifie que les informations des transactions (Raison sociale, Montant, Date) sont du bon type et que ces informations correspondent bien aux données.

SC-15, suivi consommation vérifier affichage : Vérifie que le suivi de consommation en camembert contient les bonnes informations.

SC-16, suivi consommation vérifier contenu : Vérifie que les informations affichées dans le suivi de consommation correspondent bien aux données.

SC-17, Vérifier fonctionnement conseiller virtuel : Vérifie que le bouton Conseiller virtuel fonctionne correctement et renvoie vers les bons liens.

Méthodes de test adaptées

Tests manuels : La plupart des tests sont prévus pour être effectués de manière manuelle, mais les tests les plus récurrents seront automatisés.

Tests automatisés : Les tests les plus récurrents, comme l'inscription ou la connexion ainsi que les smoke tests, seront automatisés afin de gagner du temps.

Tests exploratoires : Je pense que les tests décrits actuellement couvrent la totalité des éléments à tester. Cependant, il peut s'avérer qu'à cause d'oubli ou bien de changements dans le fonctionnement de l'application, les tests actuels ne couvrent plus la totalité des fonctionnalités. Prendre du temps pour effectuer des tests exploratoires est alors justifié.

Tests end-to-end : Une partie des tests décrits sont déjà des tests end-to-end, notamment tous les tests qui demandent à effectuer CDT-2 en premier.

Smoke tests : Certains tests comme CDT-18, qui vérifie l'affichage des informations bancaires, sont nécessaires à effectuer avant d'effectuer les tests plus avancés. Étant donné que ces tests sont amenés à être répétés régulièrement vu qu'ils couvrent des fonctionnalités critiques, ils seront automatisés.

Tests de non-régression : Du temps à la fin de chaque sprint est prévu pour effectuer des tests de non-régression afin de vérifier que les fonctionnalités testées auparavant fonctionnent toujours.

Ressources nécessaires

Ressources humaines :

Sur ce projet, je pense que la présence d'une seule personne pour le test est suffisante.
En effet, à l'exception de quelques tâches telles que la vérification des données de transaction, la plupart des scénarios de test ne sont pas très longs à effectuer.
Ainsi, même en prenant en compte le fait de répéter plusieurs fois les tests, je pense qu'il y a assez de temps sur tout le projet pour qu'une seule personne puisse effectuer tous les tests.

Ressources matérielles :

Concernant l'environnement de tests, je préconise de travailler sur un duplicata de l'application sur un environnement en ligne dédié au test sur sa propre branche GitHub. Avec sa propre base de donnée afin de simuler aux mieux les conditions de l'application réelle.
Il faudra donc générer des jeux de fausses données telles que des fausses informations bancaires et transactions pour remplir cette base de donnée ainsi que des faux profils avec nom et prénom.
Il sera également nécessaire de générer un ensemble de documents et justificatifs d'identité dans le but de tester l'inscription.

Pour l'API, étant donné que le nombre de requêtes à effectuer est limité, il sera plus facile de mettre en place une mocking API telle que Mocki. Ainsi, il est possible d'effectuer autant de requêtes que l'on souhaite. De plus, il sera plus aisé de simuler le dysfonctionnement de l'API avec cette méthode.

Concernant la machine de la personne chargée des tests, elle devra être équipée d'un éditeur de code tel que VS Code et un outil d'automatisation de test tel que Cypress devra également être installé.

Un accès à l'outil de report de bugs utilisé par l'équipe devra également être fourni, que ce soit Jira, Notion ou autre.

Étapes clés de la stratégie

Voici les principales étapes de la stratégie de test sur un sprint :

1) Faire une review de la revue d'exigence afin de voir si 'il y a des questions restées sans réponse qui peuvent impacter les objectifs du sprint en cours. Cette première review peut avoir lieu pendant ou après la réunion pour parler des objectifs du sprint.

2) Une fois la review effectuée, mettre à jour la revue d'exigence (modifier ou ajouter des questions en fonction des réponses apportées lors de la review)

3) Effectuer les premiers scénarios de tests prévus sur le sprint

4) Une fois les tests effectués, rédiger un compte rendu des tests et rapporter les bugs sur l'outil de suivi de bugs

5) Effectuer les scénarios de tests suivant en attendant que les problèmes rapportés soient réglés

6) Faire des comptes rendus et rapporter les bugs pour les scénarios effectués

7) Effectuer à nouveau les tests sur les fonctionnalités qui ont reçu des corrections/modifications en adaptant les scénarios de tests si besoin

8) Effectuer à nouveau les étapes 5 à 7 jusqu'à avoir réalisé tous les tests prévus

9) Faire les tests de non-régression

10) Faire une réunion de bilan à la fin du sprint pour voir si les objectifs ont été atteints et préparer le sprint suivant

Exemple de la stratégie mise en place sur le sprint 1

	Semaine 1	**Semaine 2**	**Semaine 3**
Étape 1	Effectuer une review de la revue d'exigence globale	Si des modifications ont été apportées sur les éléments testés, refaire SC-1 et SC-2, sinon effectuer SC-3	Si des modifications ont été apportées sur les éléments testés, refaire SC-3 et SC-4, sinon effectuer SC-5
Étape 2	Mettre à jour la revue d'exigence en priorisant les questions sur les éléments concernant le sprint en cours	Reporter les bugs découverts et faire un compte rendu de SC-1 et SC-2 ou de SC-3	Reporter les bugs découverts et faire un compte rendu de SC-5
Étape 3	Effectuer SC-1	Effectuer SC-4	Effectuer SC-6
Étape 4	Reporter les bugs découverts et faire un compte rendu de SC-1	Reporter les bugs découverts et faire un compte rendu de SC-4	Reporter les bugs découverts et faire un compte rendu de SC-6
Étape 5	Effectuer SC-2	Débriefe de fin de semaine avec le reste de l'équipe	Effectuer les tests de non-régression
Étape 6	Reporter les bugs découverts et faire un compte rendu de SC-2		Débriefe de fin de semaine et de fin de sprint avec le reste de l'équipe
Étape 7	Débriefe de fin de semaine avec le reste de l'équipe		

Préconisations

Pour commencer, je recommande à toute l'équipe de développement d'effectuer une revue complète de la revue d'exigences au début du projet, car il y a certaines questions assez critiques qui doivent être répondues.

Tout d'abord, il y a le problème de l'API qui n'est pas sécurisée. En effet, l'adresse de l'API est en HTTP et non en HTTPS. Il s'agit donc de savoir comment il est possible que l'API ne soit pas en HTTPS et comment régler ce problème.

Un autre problème concerne la base de donnée. Tout d'abord, il n'est jamais mentionné quand et comment la base de données est mise ne place.
Ensuite, un point important concerne la façon dont les données sont stockées de manière sécurisée dans la base de données.
En effet, l'application gère un certain nombre de données assez sensibles. Notamment tous les justificatifs d'identité (carte d'identité, justificatif de domicile, etc).
Il est donc très important d'être vigilant sur la façon de transférer et stocker ces données.