Stratégie de test : Concevez une stratégie de test

Scénarios prévus

Indiquez la liste des scénarios que vous avez prévus.

SC-1. Inscription

Objectif :
Ce scénario a comme but de vérifier que le champ de date de naissance utilise un format cohérent (JJ/MM/AAAA), que la longueur maximale du champ "Mot de passe" est raisonnable (16 à 20 caractères) et que le champ "Prénom" est de type texte et non numérique.

SC-2. Pièces justificatives

Objectif :
Ce scénario a comme but de vérifier :
- que le format des documents à fournir est clairement défini
- que l'utilisateur est redirigé vers la page de connexion après avoir appuyé sur le bouton "Enregistrer" si toutes les documents sont correctement téléversées
- ce qui se passe si l'un des documents n'est pas téléversé.

SC-3. Authentifications additionnelles

Objectif :
Ce scénario a comme but de vérifier ce qui se passe :
- lors de la vérification double facteur si le service ne renvoie pas de connexion OK
- en cas d'échec de l'authentification biométrique.

SC-4. Appels API pour récupération des données bancaires

Objectif :
Ce scénario a comme but de vérifier :
- que l'appel à l'API est effectué via HTTPS pour assurer la sécurité des données.

- ce qui se passe en cas d'erreur lors de l'appel à l'API ou si l'API ne répond pas dans un délai raisonnable
- ce qui se passe si l'utilisateur clique sur le bouton “Rafraîchir” plusieurs fois en moins de cinq minutes et comment gérer ces appels pour respecter la limitation des cinq minutes

SC-5. Transactions

Objectif :
Ce scénario a comme but de vérifier :
- que les utilisateurs peuvent accéder aux données historiques des transactions sans stockage en BDD
- si le bouton de téléchargement s'appelle "Télécharger mes relevés de comptes" ou "Télécharger mes relevés"
- que le nombre de transactions affichées est limité à un nombre fixe par page (10, par exemple), avec la possibilité de charger plus de pages si nécessaire.

SC-6. Suivi de consommation

Objectif :
Ce scénario a comme but de spécifier les types de dépenses à inclure dans le graphique pour éviter toute ambiguïté et clarifier si la périodicité est paramétrable en fonction d'une date fixe ou personnalisable par l'utilisateur via un sélecteur de date.

SC-7. Connexion API pour le chat du conseiller virtuel

Objectif :
Ce scénario a comme but de spécifier le type de connexion API entre la bulle et le service de chat du conseiller virtuel et comment l'API gère le passage aux options de conseil.

Méthodes de test adaptées

Renseignez dans cette section les méthodes de tests qui seront utilisées.

1. Tests API

Les tests API sont essentiels pour vérifier les interactions entre les différents services et composants via leurs interfaces de programmation. Ils permettent de s'assurer que les échanges de données se déroulent comme prévu et que les réponses des services sont correctes et en temps voulu.

Par exemple, lors des interconnexions bancaires (TOM-10), ces tests vérifient que les appels à l'API de la Banque de France récupèrent correctement les informations des clients, telles que l'identifiant du client, le nom de la banque et les numéros de comptes.

Un autre exemple pertinent est la récupération des transactions des utilisateurs (TOM-13). Les tests API permettent de vérifier que les transactions sont correctement récupérées et affichées dans le tableau de bord de l'utilisateur. Ces tests garantissent que les données sont exactes, complètes et transmises de manière sécurisée.

2. Tests exploratoires

Les tests exploratoires sont des tests non structurés où le testeur explore librement l'application pour identifier les anomalies. Ils sont particulièrement utiles pour découvrir des problèmes non anticipés par les tests formels. Par exemple, pour la gestion des documents à fournir lors de la création de compte (TOM-1), les tests exploratoires permettent de vérifier que les utilisateurs peuvent téléverser correctement leurs documents, que ces derniers sont bien pris en compte par le système, et qu'il n'y a pas de bugs ou de comportements inattendus lors de ce processus.

3. Smoke tests

Les smoke tests sont utilisés pour vérifier rapidement que les fonctions principales d'une application fonctionnent après une nouvelle mise à jour ou modification. Avant de procéder à des tests plus détaillés, il est crucial de s'assurer que le système de base est opérationnel. Par exemple, après l'implémentation des fonctionnalités de création de compte utilisateur (TOM-1), ces tests vérifieront que les utilisateurs peuvent toujours créer des comptes, se connecter, et accéder aux fonctionnalités de base de l'application.

4. Tests end-to-end

Les tests end-to-end (E2E) examinent le système complet d'un point de vue utilisateur. Ils valident que tous les composants d'une application interagissent correctement ensemble et que les flux de bout en bout fonctionnent comme prévu. Pour le processus d'authentification double facteur (TOM-4), ces tests garantiront que l'ensemble du processus, depuis la connexion initiale jusqu'à l'accès aux services de l'application, fonctionne comme prévu.

5. Tests de non-régression

Enfin, les tests de non-régression visent à s'assurer que les modifications apportées au code n'ont pas introduit de nouveaux défauts dans les fonctionnalités existantes. Ils sont cruciaux pour maintenir la stabilité du système au fil des mises à jour. Par exemple, lors d'une mise à jour du système d'interconnexion bancaire (TOM-10), ces tests vérifieront que les anciennes fonctionnalités, comme la récupération des données bancaires des utilisateurs, continuent de fonctionner correctement après l'intégration des nouvelles modifications.

Ressources nécessaires

Listez toutes les ressources que vous comptez utiliser dans la campagne de test.

A. Ressources humaines

1. Testeurs (2-3 personnes)

- **Rôle et Responsabilités :**
  - **Testeur 1 : Tests fonctionnels et exploratoires**
    - Effectuer des tests fonctionnels pour vérifier les fonctionnalités de l’application.
    - Réaliser des tests exploratoires pour découvrir des problèmes non identifiés par les tests préexistants.
  - **Testeur 2 : Tests API et automatisation**
    - Réaliser des tests API pour vérifier les interactions entre l'application et les services externes.
    - Configurer et exécuter des tests automatisés avec des outils comme Cypress.
  - **Testeurs 3** : **Tests de non-régression et de fumée**
    - Se concentrer sur les tests de non-régression pour s'assurer que les nouvelles

fonctionnalités n'ont pas introduit de bugs dans les fonctionnalités existantes.
    - Réaliser des tests de fumée pour vérifier que les fonctionnalités principales fonctionnent après chaque itération.

2. Développeur (1 personne)

- **Rôle et Responsabilités :**
    - **Support technique** : Fournir un soutien technique pour résoudre les problèmes rencontrés par les testeurs, comme les erreurs d'intégration ou les bugs identifiés.
    - **Développement des fonctionnalités** : Développer de nouvelles fonctionnalités ou corriger des bugs en fonction des retours des testeurs.

3. Product Owner (1 personne)

- **Rôle et Responsabilités :**
    - **Définition des exigences** : S'assurer que les exigences sont claires et bien définies pour les testeurs.
    - **Priorisation des tâches** : Prioriser les tâches de test en fonction des besoins du produit et des objectifs du sprint.

4. Scrum Master (1 personne)

- **Rôle et Responsabilités :**
    - **Facilitation des processus Agile** : Faciliter les réunions Agile, comme les réunions de sprint et les revues de sprint, et s'assurer que l'équipe suit les pratiques Agile.
    - **Soutien de l'équipe** : Aider l'équipe à surmonter les obstacles et à maintenir le focus sur les objectifs du sprint.

B. Outils

- **Docker** : Crée des environnements de test isolés pour simuler l'environnement de production.
- **Éditeur de code** : Pour écrire et modifier les scripts de test.
- **Jira** : Pour la gestion des tâches, le suivi des bugs et la planification des sprints.
- **Cypress** : Pour les tests automatisés
- **Postman** : Pour tester les API et gérer les requêtes HTTP.
- **Librairies pour fausses données** : Générer des données de test réalistes pour les scénarios de test (**Faker.js** ou **Mockaroo)**
- **Swagger** : Pour la documentation des API.

C. Environnement de test

- **Environnement de test isolé** : Utilisation de Docker pour créer un environnement de test distinct de l'environnement de production.
- **Sous-domaine de test** : Un site en ligne dédié pour les tests afin de ne pas affecter le site en production.
- **Jeu de données de test** : Comptes fictifs comme `test@test.fr` avec un mot de passe `Test1`, et données simulées pour créer des scénarios de test.

Étapes clés de la stratégie

Planifiez les étapes clés de la stratégie de test.

Voici les principales étapes de notre approche, tenant compte de la méthodologie agile et de la nécessité de tests continus et systématiques.

1. Planification et Conception des Tests

1.1. Identification des Exigences et Scénarios de Test

- Définir clairement les exigences du projet en collaboration avec les parties prenantes.
- Traduire ces exigences en scénarios de test détaillés

1.2. Conception des Cas de Test

- Pour chaque scénario de test, concevoir des cas de test spécifiques.
- S'assurer que chaque exigence est couverte par au moins un cas de test.
- Préparer les données de test nécessaires (utilisation de jeux de données factices).

1.3. Préparation de l'Environnement de Test

- Configurer un environnement de test qui reflète au mieux l'environnement de production sans toutefois y accéder directement.
- Utiliser des outils comme Docker pour la gestion des environnements isolés et reproductibles.
- Mettre en place les outils nécessaires : éditeur de code, Postman, Cypress, Jira, etc.

2. Exécution des Tests

2.1. Tests en Continu

- Tester les nouvelles fonctionnalités dès qu'elles sont développées.
- Effectuer des tests API, exploratoires, smoke tests, end-to-end et de non-régression selon les besoins.

2.2. Tests en Fin de Sprint

- À la fin de chaque sprint, exécuter des tests complets pour vérifier toutes les nouvelles fonctionnalités et les corrections de bugs.
- Effectuer des tests de régression pour s'assurer que les modifications n'ont pas introduit de nouveaux défauts dans les fonctionnalités existantes.
- Collaborer avec les développeurs pour s'assurer que les bogues détectés sont corrigés avant le sprint suivant.

3. Gestion des Bugs

3.1. Enregistrement des Bugs

- Utiliser un outil de suivi des bogues comme Jira pour documenter et suivre les problèmes détectés.
- Fournir des rapports détaillés sur chaque bug, y compris les étapes pour reproduire le problème, les captures d'écran et les logs pertinents.

3.2. Révisions et Corrections

- Les développeurs corrigent les bugs identifiés à la fin de chaque sprint.
- Effectuer des tests de régression pour vérifier que les corrections n'ont pas introduit de nouveaux problèmes.

4. Documentation et Reporting

4.1. Documentation Continue

- Documenter chaque étape du processus de test, y compris les cas de test, les résultats et les analyses.
- Maintenir une documentation claire et accessible pour faciliter le suivi et la gestion des tests.

4.2. Rapports de Test

- Fournir des rapports de test détaillés à la fin de chaque sprint et avant chaque release majeure.
- Inclure des informations sur les tests effectués, les résultats obtenus, les bugs détectés et les recommandations.

Préconisations

Rédigez vos recommandations que vous pourriez émettre et qui permettraient de réduire les risques que vous avez identifiés, ou les recommandations pour traiter un sujet.

Pour améliorer l'application et réduire les risques identifiés, voici quelques recommandations clés :

➢ **Sécurisation des communications**

- **Recommandation :** Assurez-vous que toutes les communications entre le client et le serveur sont sécurisées par HTTPS pour protéger la confidentialité des informations.

➢ **Gestion des erreurs**

- **Recommandation :** Développez des mécanismes de gestion des erreurs clairs pour l'API, l'authentification biométrique, l'authentification à double facteur, lors de la téléversement de documents et en cas d'erreurs de connexion au conseiller virtuel.

➢ **Optimisation des filtrages et affichage des données**

- **Recommandation :** Ajoutez des filtres pour trier les transactions par date, montant ou type, et limitez le nombre de transactions affichées par page avec une option de pagination.

➢ **Organisation et utilisation des informations dans la base de données**

- **Recommandation** : Vérifiez comment les informations récupérées sont organisées, stockées et utilisées dans la base de données, y compris l'information de l'appareil ayant émis la demande.

- **Gestion de la périodicité et de l’accès aux données historiques**

  - **Recommandation :** Implémentez une fonctionnalité permettant aux utilisateurs de choisir une période pour consulter les données historiques des transactions.

- **Tests de Non-Régression et validation des nouvelles fonctionnalités**

  - **Recommandation :** Effectuez des tests de non-régression rigoureux à chaque sprint pour s’assurer que les nouvelles fonctionnalités ne créent pas de régressions dans les fonctionnalités existantes.

- **Documentation et suivi des bugs**

  - **Recommandation :** Documentez les cas de test, les résultats, et les anomalies dans un outil de gestion de projet comme Jira pour faciliter le suivi et la résolution des problèmes.