Stratégie de test

Scénarios prévus

Fonctionnalité	Exigence	Criticité	Objectif
Création de compte	TOM-6	Moyenne	S'assurer que les champs 'Nom', 'Prénom', 'Date de naissance', 'Mot de passe' et le bouton 'Suivant' sont affichés correctement, valident les entrées, et que les informations valides sont conservées après avoir cliqué sur 'Suivant'.
Téléversement des documents	TOM-7	Moyenne	Contrôler le bon fonctionnement du téléversement via 'Parcourir', l'enregistrement et l'affichage des fichiers, ainsi que la redirection et le maintien des données après un clic sur 'Annuler'.
Authentification	TOM-8	Critique	Vérifier que l'utilisateur passe par une double authentification lors de la première connexion, et une authentification simple pour les suivantes.
Authentification biométrique	TOM-9	Critique	Valider le fonctionnement complet de l'authentification biométrique sur ordinateur et mobile.
Récupération des données bancaires	TOM-10	Critique	Tester la récupération des données bancaires via API Banque de France, leur enregistrement en base, et leur actualisation après chaque appel.
Tableau de bord	TOM-11	Critique	S'assurer de l'affichage correct des données bancaires par banque sur le tableau de bord, et de leur mise à jour après clic sur 'Rafraîchir'.
Téléchargement des relevés de compte	TOM-12	Critique	Tester la présence et le fonctionnement du bouton de téléchargement des relevés PDF, s'assurer que l'API fonctionne sans stocker d'information localement.
Consultation des transactions	TOM-13	Critique	Vérifier que les transactions sont bien affichées dans le tableau de la banque concernée, sans stockage en base de données.
Consultation du suivi de consommation	TOM-14	Moyenne	Tester l'affichage d'un graphique regroupant les dépenses par type sur une période donnée.
Conseiller virtuel	TOM-15	Mineure	Tester l'interaction avec le conseiller virtuel depuis la bulle et vérifier la redirection vers la prise de rendez-vous après une sélection.

Méthode de test

Scénario fonctionnel (TOM-x / SC-x)	Méthode préconisée	Pourquoi ?
Création de compte (TOM-6 / SC-1)	**Automatisation**	Le parcours est rejoué à chaque build : champs multiples, règles de validation et flux d'e-mails. Automatiser permet de couvrir rapidement un grand nombre de combinaisons et d'attraper les régressions dès l'intégration continue.
Téléversement des justificatifs (TOM-7 / SC-2)	**Automatisation**	Le traitement (upload, contrôle de format, stockage, statut) ne dépend pas d'une appréciation visuelle fine ; un script peut vérifier intégrité du fichier, codes-retour serveur et persistance en base bien plus vite qu'un humain.
Authentification double facteur (TOM-8 / SC-3)	**Automatisation**	Les scénarios 2FA sont sensibles à la sécurité et très sujets à régression quand on touche au back-end. Simuler l'envoi de code et la validation côté API garantit un feedback immédiat et reproductible, indispensable pour la conformité.
Authentification biométrique (TOM-9 / SC-4)	**Manuel**	Le résultat dépend du capteur matériel, du système d'exploitation et de divers facteurs environnementaux (luminosité, empreinte partielle, etc.). Un testeur humain sur appareils réels détecte des problèmes que l'automatisation hardware-agnostique ignore.
Récupération des données bancaires (API) (TOM-10 / SC-5)	**Automatisation**	Pur appel API : logique, stateless, facilement mockable. Les assertions sur codes HTTP, temps de réponse et structure JSON s'intègrent idéalement à une suite d'intégration, offrant une couverture large à coût marginal.
Tableau de bord / rafraîchissement (TOM-11 / SC-6)	**Automatisation**	Fonction centrale déclenchée souvent ; la logique est déterministe : clic → requête → rendu. Un test UI automatisé (avec captures DOM) détecte toute régression de rafraîchissement ou de latence sans mobilisation humaine.
Téléchargement des relevés PDF (TOM-12 / SC-7)	**Manuel**	Vérifier le rendu visuel, la pagination et l'accessibilité d'un PDF reste complexe à scripter de façon fiable. Un testeur humain détecte plus rapidement des artefacts (fontes manquantes, tableaux tronqués) que des parsers génériques.
Consultation des transactions (TOM-13 / SC-8)	**Manuel**	Le test consiste surtout à juger la lisibilité, l'ordre chronologique et la cohérence métier (libellés, montants). Ces aspects requièrent une validation visuelle et contextuelle que l'automatisation couvre mal.
Suivi de consommation / graphique (TOM-14 / SC-9)	**Manuel**	Les graphiques doivent être compréhensibles, esthétiques et accessibles. Les outils d'automatisation vérifient la présence d'un SVG ou d'un canvas, mais pas la pertinence du design ni la

Scénario fonctionnel (TOM-x / SC-x)	Méthode préconisée	Pourquoi ?
		lisibilité des valeurs ; l'œil humain reste la référence.
Conseiller virtuel (TOM-15 / SC-10)	**Manuel**	Les dialogues générés par IA produisent des chemins quasi infinis et un langage naturel imprévisible. Les tests exploratoires manuels repèrent incohérences, ton inadéquat ou blocages conversationnels que les scripts ne peuvent anticiper.

Ressources nécessaires

La mise en œuvre efficace de la stratégie de test repose sur la mobilisation de ressources humaines, matérielles et logicielles spécifiques. Une organisation rigoureuse de ces moyens est indispensable pour garantir la qualité, la fiabilité et la conformité du produit final.

Ressources humaines

- **Testeurs (x2)**
  Deux testeurs spécialisés interviendront sur l'ensemble du cycle de tests. Ils seront en charge des tests fonctionnels, de l'ergonomie (UX/UI), ainsi que des vérifications côté API. Leur expertise devra couvrir à la fois les tests manuels (pour les validations contextuelles et visuelles) et les tests automatisés (pour les scénarios récurrents et critiques).
- Prévoir un renfort QA (ou développeurs en pair-testing) durant les Sprints 5-6 pour couvrir l'exploratoire et la performance sans réduire la couverture.

Ressources matérielles et logicielles

- **Outil de gestion de projet et de tests** : *JIRA*
  Permet de centraliser les scénarios de test, de suivre leur exécution, de tracer les anomalies et d'assurer une transparence complète avec l'équipe projet.

- **Outil de test d'API** : *Postman*
  Utilisé pour tester manuellement les endpoints des APIs bancaires, vérifier les formats de réponse, simuler des erreurs et valider la robustesse des services.

- **Environnement de test applicatif** : *Docker*
  Fournit une version stable de l'application TOMSEN dans un conteneur isolé, facilitant les tests d'intégration et les validations fonctionnelles sans impacter l'environnement de production.

- **Base de données de test**
  Une base isolée contenant des données réalistes mais fictives est indispensable pour simuler les comportements bancaires sans compromettre de vraies données.

- **Navigateurs et équipements**

1. Navigateurs : Chrome, Firefox, Edge, Safari (pour la compatibilité UI).
2. Mobiles : Smartphones Android et iOS (pour tester la biométrie et la responsivité).

- **Outil de versioning** : *GitHub*
  Pour assurer le suivi des versions, la traçabilité des correctifs et la collaboration entre testeurs et développeurs.

- **Outil d'automatisation de tests** : *Cypress*
  Il permettra d'automatiser les scénarios critiques, notamment ceux liés à l'authentification, au tableau de bord, et à la récupération des données bancaires.

- **Outil de communication d'équipe** : *Zoom*
  Utilisé pour les réunions de coordination, les démos de tests, et les revues collectives d'anomalies.

Etapes clés de la stratégie

Nous adoptons une démarche **Agile Scrum** : le projet est découpé en sprints de deux semaines, chacun livrant un incrément testé et potentiellement déployable.
- **À la fin de chaque sprint**, toute l'équipe se réunit pour la *Sprint Review* (démo, feedback métier) puis la *Rétrospective* (amélioration continue) et prépare le *Sprint Planning* suivant.
- **Chaque matin**, développeurs et testeurs tiennent un *daily stand-up* de quelques minutes ; chacun partage ce qui a été fait, ce qui est prévu et les éventuels obstacles, afin d'ajuster immédiatement le travail et la stratégie de test.

Cette cadence garantit un feedback régulier, une visibilité permanente sur l'avancement et une collaboration étroite entre développement et QA.

Sprint 1 (semaines 1-2) – *Fondations*
- Lecture intégrale des spécifications, identification des risques et élaboration complète de la stratégie de test.
- Première version du cahier de recette (CDR) : flux critiques uniquement.
- **Fin de sprint 1 :** écriture et exécution d'un **squelette d'automatisation "smoke"** (lancement appli, login, page d'accueil). On place déjà la suite dans le pipeline CI pour obtenir un feu vert/rouge quotidien dès la semaine 3.

Sprint 2 (semaines 3-4) – *Conception finalisée + auto lot #2*
- CDR porté à 100 % (tous scénarios, jeux de données préparés).
- Poursuite du développement côté équipe Dev.
- **Fin de sprint 2 :** lot #2 de scripts automatisés : création de compte, API bancaires, 2FA. Exécution immédiate dans la CI ; la non-régression continue est officiellement active.

Sprint 3 (semaines 5-6) – *Premiers tests manuels*
- Les premières user-stories livrées sont couvertes par le CDR mis à jour en continu.
- **Semaine 6 :** démarrage des **tests manuels systémiques** (exécution du CDR sur l'incrément réel).
- Lot #3 d'automatisation écrit et lancé en clôture de sprint : il fiabilise les corrections issues des tests manuels.

Sprint 4 (semaines 7-8) – *Croisement manuel / auto + exploratoire 1re passe*
- Manuel : couverture complète de tous parcours disponibles.
- Automatisation : lot #4 (tableau de bord, rafraîchissement, téléchargements PDF).
- **Fin de sprint 4 :** première **session exploratoire ciblée** (biométrie sur terminaux réels) pour dénicher des cas limites matériels.

Sprint 5 (semaines 9-10) – *Exploratoire renforcé & performance*
- Exploratoires et manuels sur : graphiques de consommation, chatbot, rendus PDF (accessibilité).

- Tests de performance sur l'API bancaire
- Lot #5 d'automatisation intègre les correctifs exploratoires + asserts temps de réponse.

Sprint 6 (semaines 11-12) – *Stabilisation & Go Live*
- Exécution complète de la non-régression (lots #1 → #5) chaque jour.
- Re-passage manuel ciblé sur fonctionnalités critiques modifiées tardivement.
- Rédaction du bilan de recette et procès-verbal.
- Buffer de quelques jours pour corriger les ultimes anomalies, puis **mise en production** à la fin de la semaine 12 après le Go/No-Go.

Préconisation

Axe	Pourquoi ?	Risque si ignoré	Préconisation
Synchronisation QA / Dev	Planning serré en sprints ; tout retard Dev décale la fenêtre de test.	Exécution bâclée ou reportée, régressions non détectées.	• Revue hebdo Dev-QA. • Priorisation *critique* sur les scénarios SC-1, 2, 3, 5, 6. • Buffer de 2 jours avant GoLive pour absorber les décalages.
Automatisation des parcours critiques	Nos parcours SC-1, 2, 3, 5, 6 sont déterministes et rejoués à chaque build.	Scripts prêts trop tard, pas intégrés au pipeline CI/CD.	• Concevoir les scripts Cypress/Postman dès la **semaine 2**. • POC technique validé **fin sprint 1**. • Intégrer l'exécution sur chaque merge-request.
Campagne de non-régression "double niveau"	Garantit la stabilité continue et finale.	Effets de bord découverts en production.	• **NR continue** : exécution quotidienne du socle auto. • **NR finale** : campagne complète au **sprint 6** sur tous scénarios critiques. • Se référer au tableau* « ***Où renforcer la non-régression ?*** ».
Exploratoires ciblés	Certaines fonctionnalités (biométrie, PDF, graphiques, chatbot) nécessitent une appréciation humaine.	Défauts UX ou données visuelles non captés par l'auto.	• Planifier des sessions exploratoires durant les dernières semaines avant le sprint final. (cf. tableau* « ***Où ajouter des tests exploratoires ?*** »). • Conserver les notes d'observation dans Notion.
Gestion proactive des anomalies	Backlog d'anomalies mal suivi = fin de sprint désorganisée.	Perte de visibilité, re-tests multiples.	• Tableau partagé (Notion) avec statut, sévérité, owner. • QA référent valide chaque correctif avant clôture ticket.
Documentation et traçabilité	Traçabilité = gage de qualité + transfert de connaissance.	Cas non rejoués, décisions non justifiées.	• Mettre à jour en continu : cahier de recette, journal d'exécution, rapport NR. • Hébergement centralisé (Drive/Confluence).
Alignement des environnements	Écart pré-prod / prod = 1re source de bugs latents.	Fonction valide en test, KO en prod.	• Parité config + données réalistes. • Vérifier les accès API tierces dès le sprint 1.

Axe	Pourquoi ?	Risque si ignoré	Préconisation
			• Maintenir un mock contractuel de l'API Banque de France (OpenAPI + Prism) et exécuter des tests de contrat à chaque build pour détecter toute dérive de schéma.
Sécurité applicative	Application bancaire exposée à Internet ; obligations réglementaires (DSP2, RGPD).	faille XSS/SQLi, perte de données, sanctions.	• Analyse SAST/SCA automatisée à chaque build (ex. OWASP Dependency-Check, SonarQube). DAST hebdo (OWASP ZAP) + rapport dans la CI. Pentest externe entre Sprints 4 et 5 avec suivi des correctifs.

*Où renforcer la **non-régression** ?

Scénario	Pourquoi une suite de non-régression ?	Couverture idéale
1 – Création de compte	Parcours critique et exécuté à chaque build ; toute régression bloque l'onboarding.	Tests UI + API : validation de chaque règle de champ, envoi d'e-mail, inscription en base.
2 – Téléversement de justificatifs	Flux répétitif ; on veut éviter de re-casser la vérification de format ou les droits d'accès.	API / back-office : upload, antivirus mocké, contrôle MIME, persistance.
3 – Authentification 2FA	Haut risque de régression (timing, jetons expirés) et impact sécurité.	Tests API end-to-end : génération du code, délai d'expiration, scenarios « code déjà utilisé ».
5 – Récupération des données bancaires (API)	Intégration externe : toute modification de schéma JSON ou de mapping doit être détectée instantanément.	Tests de contrat (schema) + performance (< 500 ms) + assertions métier.
6 – Tableau de bord / rafraîchissement	Action fréquente et visible ; la moindre régression se répercute sur tous les utilisateurs connectés.	Test UI chronométré : clic « Rafraîchir », contrôle DOM, event WebSocket, timing.

*Où ajouter des **tests exploratoires** ?

Scénario	Pourquoi un test exploratoire ?	Quand l'exécuter ?
4 – Authentification biométrique	L'expérience dépend de facteurs réels : capteurs, luminosité, empreintes partielles. Les exploratoires dévoilent des cas limites impossibles à simuler (doigts humides, capteur rayé…).	À chaque nouvelle version d'OS / firmware et après l'ajout de nouveaux terminaux au parc.
7 – Téléchargement des relevés PDF	Les testeurs peuvent déceler des artefacts visuels (polices manquantes, tableaux décalés, contraste insuffisant) ou des problèmes d'accessibilité que les scripts ignorent.	Après chaque refonte de template PDF ou mise à jour de la librairie de génération.

Scénario	Pourquoi un test exploratoire ?	Quand l'exécuter ?
8 – Consultation des transactions	Vérifier la lisibilité (libellés, montants, tri), la cohérence métier et la navigation (pagination, filtres) gagne à être exploré par des humains qui vont essayer des chemins inattendus.	À la fin de chaque sprint comportant des changements UI ou règles de comptabilisation.
9 – Suivi de consommation / graphiques	Les graphes doivent rester compréhensibles même avec des jeux de données atypiques (pics, valeurs négatives). Un exploratoire mettra la jauge à l'épreuve d'outliers.	Lorsqu'on change la librairie de charting ou les agrégations back-end.
10 – Chatbot / conseiller virtuel	Le langage naturel crée des branches de dialogue quasi infinies ; seuls des tests exploratoires peuvent piocher des tournures imprévues, vérifier la tonalité ou détecter des boucles.	En continu : nouvelle version de modèle LLM, enrichment du corpus, nouvelle langue supportée.