Plan de test - Projet « TODO IT Now »

Nom / Code projet : TODO IT Now - Gestionnaire de tâches
Référence : PT-TODO-001
Chef de projet : Chantal Dupont *(ou responsable désigné du projet)*
Service / Organisation : Département Informatique - Développement Progiciels

Version :	Auteur(s)	Description	Date
0.1	Alexandre LABORDE, Damien SENECHAL	Initialisation du document	22/04/2025

Introduction

Contexte général et déroulement
L'entreprise TODO IT Now développe un gestionnaire de tâches destiné aux managers et employés de l'entreprise. L'outil a pour objectif de faciliter l'organisation du travail et le suivi des projets d'équipe.

À travers l'interface web, un manager peut gérer les emplois du temps mensuels et hebdomadaires etc.. de ses collaborateurs. Chaque manager peut accéder uniquement aux "timesheets" (feuilles de temps) des membres de son équipe, ainsi qu'à son propre agenda (géré par son supérieur hiérarchique N+1)..

Dans ce contexte, la politique de tests de l'entreprise met l'accent sur la conformité, la qualité, la performance et l'expérience utilisateur. Le présent plan de test est élaboré en suivant une méthodologie de test standard (cycle en V).

Les tests seront intégrés au cycle de développement afin d'identifier et corriger les anomalies le plus tôt possible.

Objectifs des tests
Les objectifs principaux des tests pour **TODO IT Now** sont :

- **Conformité fonctionnelle :** Chaque fonctionnalité (gestion des timesheets, restrictions d'accès, demandes de modification, etc.) doit correspondre aux spécifications.

- **Qualité avant mise en production :** S'assurer que l'application est stable, performante avant son déploiement en production. Cela inclut la vérification de la performance (temps de réponse, charge supportée).

- **Utilisabilité et satisfaction des employés :** Vérifier que l'application est intuitive et utilisable par les employés de l'entreprise. Les interfaces utilisateur doivent être

validées via des tests utilisateurs.

Informations détaillées

Définition des tests effectués et leur but

Pour couvrir l'ensemble des objectifs ci-dessus, différents types de tests seront effectués. La démarche suivie pour construire ce plan de test consiste à définir plusieurs niveaux de test, chacun ayant un but précis :

- **Tests unitaires :** Vérifier le bon fonctionnement de chaque composant ou fonction individuelle de l'application.

- **Tests d'intégration :** Vérifier que les différents modules de l'application interagissent correctement entre eux une fois assemblés.

  Les tests d'intégration pourront être en partie automatisés à l'aide de Robot Framework (outil de test retenu) pour enchaîner des scénarios complets.

- **Tests de performance :** Évaluer les performances de l'application en conditions proches de la production. On mesurera les temps de réponse de l'interface (par ex. chargement du tableau de bord des timesheets), la capacité à supporter de nombreux utilisateurs simultanés, et l'utilisation des ressources (CPU, mémoire) côté serveur.

  Des tests de charge et de stress pourront être effectués (pouvant être automatisés via des scripts Robot Framework ou un outil spécialisé) pour identifier d'éventuels points de contention.

- **Tests de compatibilité :** Vérifier que l'application fonctionne correctement dans différents environnements utilisateurs. Étant donné qu'il existe une version web (navigateurs) et des versions mobiles (smartphones Android, iOS), il faudra tester la compatibilité avec plusieurs navigateurs (Chrome, Firefox, Safari, Edge…) et systèmes d'exploitation mobiles (différentes versions d'Android et d'iOS).

- **Tests utilisateurs (acceptation) :** Faire évaluer l'application par un panel d'utilisateurs réels (managers et employés) dans des scénarios proches de l'utilisation en production.

Outil de test – Robot Framework : L'outil principal choisi pour l'automatisation de nos tests est Robot Framework. L'utilisation de Robot Framework permettra d'uniformiser la démarche de test et de réutiliser les scripts à chaque nouvelle version de l'application.

Analyse de risque initiale

Identification des risques avant tests
Avant de définir le plan de test détaillé, une analyse des risques a été réalisée afin de cibler les aspects les plus critiques à tester. Voici les principaux **risques identifiés** et leur influence sur notre stratégie de test :

- **Risque de non-conformité fonctionnelle (Gravité (5/5)) :** Si certaines fonctionnalités clés ne respectent pas les exigences (par ex. un manager peut voir les données d'une autre équipe, ou une demande de congé n'est pas transmise correctement), cela compromettrait la confiance dans l'application.

  *Criticité élevée (20/20)***:** Concentration de tests unitaires et d'intégration sur toutes les règles de gestion (droits d'accès, workflows de demandes).

- **Risque de contre-performance ou de charge (Gravité (5/5)):** L'application pourrait ralentir ou planter sous une forte charge (ex. grand nombre d'employés se connectant en début de semaine pour voir leur planning).

  Criticité élevée (20/20) **:** Planification de tests de performance et de charge approfondis. Définition de critères d'acceptation stricts sur les temps de réponse (ex : <2 secondes pour charger un planning) et l'utilisation des ressources.

- **Risque d'incompatibilité ou d'inaccessibilité (Gravité (3/5)):** L'application pourrait ne pas fonctionner sur certaines configurations (navigateur obsolète, smartphone ancien) ou présenter des défauts d'affichage sur mobile.

  *Criticité moyenne (9/20)***:** Tests de compatibilité à réaliser sur un large panel d'environnements (navigateurs principaux).

- **Risque de mauvaise adhésion utilisateur (Gravité (3/5)):** Si l'ergonomie est insuffisante ou si l'application est complexe à utiliser, les employés pourraient la délaisser.

  *Criticité moyenne (9/20)***:** Prévoir des tests utilisateurs pilotés (beta-test en interne) pour identifier les problèmes d'UX/UI et les corriger avant déploiement.

- **Risque de retard de livraison dû à des bugs critiques (Gravité (4/5)):** Si des défauts majeurs sont découverts tardivement, cela peut retarder la mise en production.

*Criticité variable (8/20)***:** Suivi rigoureux des anomalies tout au long des phases de test, et exécution anticipée des tests sur les fonctionnalités critiques.

Liste des tests à réaliser et ressources nécessaires

Planification des tests et ressources
La liste ci-dessous récapitule les différents tests à effectuer pour couvrir les exigences du projet, et précise pour chacun les **ressources nécessaires**. L'ensemble de ces tests sera géré à l'aide de Robot Framework pour ceux qui sont automatisables, et manuellement pour les autres.

- **Tests unitaires** *:* Réalisés par les développeurs durant la phase de développement de chaque module. Outils de test unitaire. (ex. frameworks xUnit propres au langage utilisé)

  Temps alloué : les développeurs prévoient du temps dans chaque sprint (ou cycle) pour coder et exécuter les tests unitaires.

  Objectif de couverture : >80% du code critique couvert par des tests unitaires.

- **Tests d'intégration** *:* Menés par l'équipe développement.

  Environnement d'intégration dédié (serveur de test avec base de données).

  Outils : Robot Framework servira à automatiser des scénarios(par ex. création d'une tâche par un employé, validation par un manager).

  Personnel : 1-2 testeurs en charge de concevoir les cas d'intégration et d'analyser les résultats, en collaboration avec les développeurs pour le débogage.

  Temps : Prévoir une période de 3 jours par itération pour exécuter ces tests et traiter les anomalies découvertes.

- **Tests de performance** *:* Réalisés par un ingénieur test.

  Outils : utilisation de Robot Framework avec des librairies de test de performance, ou d'un outil spécialisé (tels JMeter, Gatling), pour simuler de multiples utilisateurs.

  Infrastructure : environnement de test performant comparable à la production, pour effectuer des tests de charge.

  Personnel : 1 testeur pour écrire et exécuter les scripts.

  Temps : Fenêtre dédiée pendant la phase de recette technique (par ex. 2 jours pour exécuter les scénarios de charge et 1 jour pour analyser et ajuster).

- **Tests de compatibilité** *:* Réalisés par l'équipe QA manuellement.

  Matériel : accès aux principaux navigateurs sur PC.

  Personnel : 1-2 testeurs pour exécuter une batterie de tests (vérifier le rendu de l'interface, le fonctionnement des fonctionnalités clés).

  Temps : quelques jours dédiés en fin de cycle de test pour parcourir toutes les configurations supportées officielles.

- **Tests utilisateurs** *:* Impliquent un groupe pilote de 5 à 10 employés de l'entreprise (managers et collaborateurs représentatifs) qui vont essayer l'application en conditions quasi réelles.

  Organisation : sessions de test encadrées par l'équipe projet.

  Outils : scénarios de tests guidés fournis aux utilisateurs et formulaires de feedback.

  Personnel : le responsable QA ou chef de projet assure le suivi des sessions.

  Temps : sur 1 à 2 jours, permettant aux utilisateurs de parcourir les fonctionnalités principales et de donner leur avis.

Infrastructure

Comment les tests seront-ils effectués ?
L'exécution des tests reproduira au mieux l'environnement de production :

- **Environnement de test (staging)** : Un serveur hébergera l'application TODO IT Now pour les besoins des tests. (par exemple, un jeu de collaborateurs et de managers fictifs, horaires, demandes déjà saisies). Cet environnement sera utilisé pour les tests d'intégration, de performance et de compatibilité. Il permettra de tester sans impacter les données réelles de l'entreprise.

- **Jeu de données de test** : Des données de **simulation** seront préparées (planning type, comptes utilisateurs de différentes catégories, etc.) afin de valider les fonctionnalités.

- **Automatisation via Robot Framework** : Les scénarios d'intégration et certains scénarios end-to-end sont codés sous forme de **tests Robot Framework**. Ces tests automatiques seront exécutés localement par les testeurs.

- **Outils de mesure de performance** : Sur l'environnement de test, des outils de monitoring (APM) seront activés pour mesurer les performances pendant les tests de charge (temps de réponse moyen, taux d'erreur HTTP, utilisation CPU/Mémoire du serveur). Si Robot Framework est utilisé pour orchestrer la charge, il sera complété par ces outils de mesure. Dans le cas de tests de charge plus poussés, un outil

dédié pourra générer du trafic simultané pendant que Robot Framework vérifie l'intégrité fonctionnelle sous charge.

- **Gestion des configurations pour compatibilité** : Pour les tests multi-navigateurs, on pourra utiliser des machines virtuelles ou un service de cloud testing pour disposer de différentes versions de navigateurs.
- **Rapports** : Avant chaque sprint de test, l'environnement sera réinitialisé. Les testeurs suivront des scénarios prédéfinis et renseigneront pour chaque cas de test le résultat dans un rapport de test. Les anomalies seront immédiatement enregistrées dans l'outil de suivi avec toutes les informations nécessaires. À la fin de la campagne, un rapport de synthèse sera produit, listant le nombre de tests passés, le nombre d'échecs, la liste des anomalies ouvertes et leur criticité, afin de décider des suites à donner (corrections, retests, etc.).

Cette infrastructure et ces processus garantissent que les tests pourront être répétés de manière cohérente et que les résultats seront fiables. L'équipe s'assurera également de la **sécurité** de l'environnement de test (les données utilisées ne doivent pas contenir de données personnelles réelles) et du maintien à jour de l'environnement (application déployée à jour des dernières versions livrées par les développeurs).

Scénarios de test et critères d'acceptation

Dans le cadre de ce plan de test, plusieurs scénarios de tests clés ont été imaginés pour valider les cas d'utilisation de l'application. Pour chacun, on définit les critères d'acceptation et les critères de rejet. Voici quelques scénarios représentatifs :

- **Scénario 1 : Consultation du planning d'équipe par un manager**
    - **Description :** Un manager se connecte à l'application web et accède à la liste des timesheets de son équipe pour la semaine en cours.
    - **Critères d'acceptation :** Le manager voit uniquement les membres de **son équipe** et pour chacun le planning correct de la semaine. Il peut ouvrir le détail d'un collaborateur et voit ses horaires journaliers. Aucune donnée d'une autre équipe n'est visible. L'affichage est rapide (<= 2s pour charger la liste).
    - **Critères de rejet :** Le test est considéré comme un échec si le manager peut voir des informations qui ne concernent pas son équipe (violation des règles d'accès), ou si des données attendues ne s'affichent pas (ex : planning vide alors qu'il devrait y avoir des entrées). Un temps de chargement excessif (> 5s) serait également noté comme un échec de performance.
- **Scénario 2 : Demande de modification d'un collaborateur et validation par le manager**

    - **Description :** Un employé (non-manager) se connecte et soumet une demande de modification de son planning (par ex. demander un congé ou un échange de créneau). Le manager de cet employé reçoit la demande et y répond.

    - **Critères d'acceptation :** Côté employé, une confirmation s'affiche une fois la demande envoyée. La demande apparaît bien dans la liste des demandes en attente du côté manager. Côté manager, le système notifie qu'une nouvelle demande est reçue. Le manager peut consulter le détail de la demande et choisir de l'approuver ou de la refuser. Après action du manager, l'employé voit le statut mis à jour (accepté/refusé) et le planning est mis à jour en réaction si c'est le planning est validé.

    - **Critères de rejet :** Le test échoue si le manager ne peut pas approuver/refuser, ou si après validation le changement n'est pas modifiée dans le planning de l'employé.

- **Scénario 3 : Gestion du planning d'un manager par son N+1**

    - **Description :** Un directeur (N+1) se connecte et modifie l'agenda d'un manager sous sa responsabilité (par ex. assigner une réunion ou bloquer une période).

    - **Critères d'acceptation :** Le directeur peut accéder au planning de ses managers subordonnés. La modification (rajout d'un événement sur un agenda) est correctement sauvegardée et le manager concerné voit la mise à jour sur son propre calendrier. Les permissions sont respectées : un directeur ne peut modifier que les agendas des managers de **son périmètre**.

    - **Critères de rejet :** Le test est en échec si le directeur n'a pas accès aux modifications, ou s'il peut par erreur modifier l'agenda d'un membre dont il devrait ne pas avoir accès.

- **Scénario 4 : Test de charge sur la consultation des plannings**

    - **Description :** Simuler 100 utilisateurs (managers et employés confondus) accédant simultanément à l'application un lundi matin pour consulter ou mettre à jour leurs plannings.

    - **Critères d'acceptation :** Le système supporte la charge sans erreur majeure. Le taux de réussite des connexions (consultation, soumission de demandes) est de 100% et les temps de réponse moyens restent au-dessous de < 3 secondes pour afficher une page). L'application ne plante pas et le serveur reste opérationnel (utilisation CPU/mémoire sous les seuils critiques).

    - **Critères de rejet :** Le test est un échec si des erreurs système apparaissent pour certains utilisateurs, ou si le temps de réponse moyen dégrade fortement (> 5-6s), ou si le serveur devient instable (saturation, crash).

- **Scénario 5 : Tests de compatibilité multi-navigateurs et mobiles**

    - **Description :** Exécuter un ensemble de cas de test de base (connexion, consultation planning, création de demande, etc.) sur différents navigateurs web et sur les applications mobiles Android/iOS.

    - **Critères d'acceptation :** Sur **chaque navigateur** testé (Chrome, Firefox, Safari, Edge), les fonctionnalités principales fonctionnent sans erreur et l'affichage est correct (pas de décalage majeur, textes lisibles, boutons cliquables). Sur mobile (dernières versions d'Android et iOS), l'application mobile s'installe et se lance sans problème, et les mêmes cas de test aboutissent correctement (par ex. un employé peut faire une demande depuis son smartphone et le manager la voir sur son interface web, confirmant l'interopérabilité).

    - **Critères de rejet :** Toute **anomalie critique** sur un environnement donné entraîne un échec. Par exemple, si sur Safari la page reste bloquée, ou si sur un mobile Android une fonctionnalité provoque un crash de l'application, le test est considéré comme échoué pour cette configuration. Des problèmes d'affichage mineurs peuvent être tolérés s'ils n'impactent pas l'utilisation, mais devront être corrigés ultérieurement.

Ces scénarios de test couvrent les chemins critiques de l'application. Ils seront détaillés dans des **cas de test** individuels avec des étapes pas-à-pas dans notre gestionnaire de tests. Chaque cas de test possède un identifiant, une description, les prérequis, les étapes d'exécution, les résultats attendus et un statut (à réussir ou échoué). Les critères d'acceptation mentionnés ci-dessus correspondent aux résultats attendus, tandis que les critères de rejet indiquent les conditions considérées comme des anomalies à corriger.

Pendant l'exécution, tout écart par rapport aux critères d'acceptation sera consigné et donnera lieu soit à une correction logicielle suivie d'une revalidation, soit, si l'anomalie est mineure et acceptable temporairement, à un **risque résiduel** documenté pour décision d'aller en production (voir section suivante).

Analyse de risque finale (risque résiduel)

Après l'exécution de l'ensemble des tests prévus dans ce plan, une **analyse de risque finale** sera réalisée pour évaluer le **risque résiduel** avant le déploiement. Cette analyse examine les résultats des tests et les anomalies éventuellement ouvertes à la fin de la campagne :

- Si **tous les tests critiques sont réussis** et qu'aucune anomalie majeure n'est en cours, le risque de défaillance majeure en production est considéré comme **faible**. Le projet peut être validé pour passage en production en toute confiance. Les quelques anomalies mineures restantes (cosmétiques ou améliorations mineures d'ergonomie) sont alors documentées pour être corrigées dans une version future

sans bloquer le lancement.

Stratégie de test - Projet « TODO IT Now »

Nom / Code projet : TODO IT Now - Gestionnaire de tâches
Référence : ST-TODO-001
Chef de projet : Chantal Dupont *(ou responsable désigné du projet)*
Service / Organisation : Département Informatique – Développement Progiciels

Version	Auteur(s)	Description	Date
0.1	Alexandre LABORDE, Damien SENECHAL	Initialisation du document	22/04/2025

Objectifs et Contexte

Contexte du projet
TODO IT Now est une application composée d'une interface web et d'applications mobiles, qui permet aux managers de planifier le travail de leurs équipes et aux employés de consulter et demander des modifications de leur planning. des règles d'accès spécifiques seront obligatoires (chaque manager gère uniquement son équipe, chaque employé ne voit que ses données). Le but du jeu est de fournir un outil efficace pour améliorer l'organisation interne, sans introduire de complexité ou de risques (erreurs de planning, problèmes de droits d'accès, etc.).

La **stratégie de test** a pour objectif deux objectifs majeurs :

- **Conformité des fonctionnalités :** s'assurer que toutes les fonctionnalités correspondent aux besoins exprimés et aux particularités.

- **Qualité et stabilité :** vérifier que l'application soit technique et complèt avant sa mise en production, en trouvant et corrigeant les bugs potentiels ..

Généralités

Vecteur directeurs et méthodologie
Les principaux principes guidant la stratégie de test du projet TODO IT Now sont :

- **Tester le plus tôt possible** : Prévenir et chercher les défauts pour réagir le plus tôt possible.

- **Prioritisation par les risques** : Le choix de l'importance/la concentration de choix des tests sont proportionnés en fonction de la criticité des fonctionnalités et des risques identifiés. Les fonctionnalités dites les plus critiques pour le métier ou

présentant le plus grand risque recevront un focus en termes de tests

- **Automatisation pragmatique** : On cherche à automatiser les tests répétitifs et régressifs à l'aide de Robot Framework, afin d'accélérer les cycles de test et d'augmenter la fiabilité des résultats.

En termes de méthodologie, la stratégie de test fonctionne dans un cycle de développement Agile. À chaque sprint de développement, les activités de test correspondantes sont planifiées. Le projet suit un cycle en V, alors la stratégie prévoit une phase de tests unitaires en parallèle du codage, une phase de tests d'intégration après l'assemblage du logiciel.

Informations détaillées

Démarche pour construire la stratégie
La construction de cette stratégie de test repose sur l'analyse des exigences. Chaque exigence a été examinée pour déterminer comment la vérifier..

1. **Choix des outils et processus** supportant la stratégie. Le choix de **Robot Framework** pour l'automatisation a été entériné en raison de sa compatibilité avec nos besoins (tests web et mobiles, langage de script lisible, intégration CI facile). De même, l'utilisation d'un outil de suivi des bugs et de gestion des tests (Jira avec module XRay, TestLink) fait partie de la stratégie pour assurer la traçabilité.

2. **Planification macro** des activités de test. Sur la base du planning projet, des jalons de test ont été placés (par ex. fin Sprint 5: tests d'intégration module X terminés; Sprints 6-7: campagne de tests système; etc.). Cette planification assure que les tests s'enchaînent de manière logique et que les dépendances (ex: test de performance après stabilisation fonctionnelle) sont respectées.

Périmètre de la stratégie de test
Le périmètre couvert par la stratégie inclut l'ensemble des composants de l'application TODO IT Now : le **frontend web**, le **backend serveur** et la **base de données** associée. Tous les modules (gestion des utilisateurs, gestion des plannings, notifications de demandes, etc.) sont couverts. Sont également inclus les aspects transverses comme les performances, la sécurité basique (contrôle d'accès) et l'ergonomie. En revanche, ne sont pas couverts explicitement par cette stratégie les tests de sécurité avancés (tests d'intrusion) ou la conformité légale (par ex. RGPD) .

Analyse des risques

La stratégie de test s'appuie sur une analyse des risques susceptibles d'affecter la qualité du produit ou le déroulement des tests, et leur assigne des niveaux de gravité et de probabilité. Voici un résumé des risques principaux et des réponses:

- **Fonctionnalité non conforme aux attentes** *:* Certaines fonctionnalités livrées ne correspondent pas exactement aux besoins métier (erreurs dans la logique de planning, règles d'approbation de demandes incorrectes).

  Impact : Important pour l'utilisateur final (mauvaises données, erreurs de planning).

  Stratégie : Des tests de validation fonctionnelle seront conçus pour chaque fonctionnalité critique. Les tests utilisateurs en fin de parcours permettront de lever les ambiguïtés restantes.

- **Défauts critiques découverts tardivement** *:* Des anomalies graves (ex : crash de l'application lors de l'utilisation simultanée par plusieurs personnes) pourraient n'apparaître que lors de la mise en production si les tests ne les détectent pas.

  Impact : Élevé (interruption de service, correction en urgence, coût).

  Stratégie : Mettre en place des tests de charge et de robustesse en environnement contrôlé pour reproduire les conditions de production.

- **Problèmes de performance ou d'évolutivité** *:* L'application fonctionne bien avec peu d'utilisateurs mais se dégrade en situation réelle (plus d'utilisateurs, données volumineuses sur plusieurs mois).

  Impact : Élevé (ralentissements, refus de l'outil par impatience, surcharge des équipes de support).

  Stratégie : Prévoir une campagne de tests de performance progressive (monter en charge graduellement jusqu'aux limites), et optimiser dès que des signes de faiblesse sont détectés. La stratégie inclut potentiellement une **optimisation en continu** : les résultats de chaque test de performance influenceront le sprint suivant (ajout de tâches de performance tuning si besoin).

Répartition des tests par niveau

La stratégie de test répartit les tests à effectuer selon différents niveaux ou phases, en définissant pour chacun l'objectif, les types de tests associés, et les responsabilités. Cette section décrit la hiérarchisation des tests du plus bas niveau (unitaire) jusqu'au plus haut (acceptation utilisateur) :

- **Tests unitaires (niveau 1)***:* Valider individuellement chaque composant logiciel (fonctions, méthodes, modules) en isolation selon une approche TDD ou au moins en même temps que l'implémentation.

  Responsable : Équipe de développement (chaque développeur garantit les tests unitaires de son code).

  Outils : Frameworks unitaires du langage (JUnit, NUnit, etc.) pour l'automatisation, exécution intégrée au pipeline CI (par exemple, push du code au lancement automatique des tests unitaires).

- **Tests d'intégration (niveau 2)** *:* Vérifier les interactions entre plusieurs composants intégrés. Après assemblage des modules (frontend-backend-base de données), ces tests visent à détecter des problèmes d'interface, de communication ou de cohérence des données.

  Responsable *:* Équipe QA

  Outils : Robot Framework est utilisé pour automatiser des scénarios d'intégration complexes impliquant, par exemple, une suite d'actions utilisateur sur l'interface web et la vérification des résultats en base de données.

- **Tests système / fonctionnels (niveau 3)** *:* Valider le système complet dans un environnement simulant la production, sur l'ensemble de ses exigences fonctionnelles et non-fonctionnelles. Ici, on englobe les tests de performance, les tests de compatibilité et d'autres tests système

  Responsable : Équipe QA

  Outils : Outils de test de charge (Robot Framework + librairies, JMeter), parc de navigateurs et devices pour compatibilité.

- **Tests d'acceptation utilisateur (niveau 4)***:* Obtenir la validation du produit par les utilisateurs finaux ou le client métier dans des conditions réelles ou quasi-réelles. Des scénarios de test « métier » sont exécutés par des utilisateurs pilotes (managers, employés) pour s'assurer que l'application répond bien aux besoins opérationnels.

  Responsable : Utilisateurs finaux encadrés par l'équipe projet

Outils : Formulaires de feedback, éventuellement scripts de test manuels décrits étape par étape.

Approbations

Les personnes ci-dessous approuvent ce document :

- **Le chef de projet (nom)** : *Chantal DUPONT* (signature) – Date : 22/04 */2025*
- **Le sponsor (nom)** : *Olivier ROBESPIERRE* (signature) – Date : 22/04 */2025*
- **Le responsable QA / Intégration (nom)** : *Jacques MEUNIER* (signature) – Date :22/04 */2025*
- **Le directeur informatique (nom)** : *Daniel CHÂTEAU* (signature) – Date : 22/04 */2025*