Retour au dossier de maintenance	
Sujets connexes	Cartographie fonctionnelle de LoGeAs Informations sur la version en cours Proc#145 : Procédure de Gestion des Versions Proc#168 - Chaîne de qualification et de livraison Proc#167 - Procédure de mise en ligne d'une version du client « lourd » Proc#169 - Cycle de release LoGeAs Web Proc#170 - Guide pratique — Gestion des versions LoGeAs Web

Méthodologie des tests

Présentation

Ce document est un document de référence stable, commun à toutes les versions de LoGeAs Web.
Il décrit la méthodologie des tests appliqués au logiciel, les garanties qu'ils apportent, leurs limites et les modalités de lecture et de validation des rapports qualité de version.
Les résultats des tests sont présentés dans chaque rapport qualité de version, accessible depuis l'écran **« Rapports de version »** de la cartographie fonctionnelle.

1. Objet

Ce document constitue une pièce du dossier de certification NF Logiciel. Il répond à trois questions principales :

- **Quels sont les différents types de tests réalisés et comment se complètent-ils ?** (sections 2 à 4)
- **Comment les tests sont-ils organisés et exécutés ?** (section 5)
- **Comment lire et valider un rapport qualité de version ?** (sections 6 et 7)

2. Les tests unitaires

2.1. Définition

Un test unitaire vérifie le comportement d'une unité de code isolée, telle qu'une fonction, une méthode ou un composant.
Les dépendances réelles (serveur, base de données ou autres écrans) sont remplacées par des simulations (*mocks*).
Les tests unitaires s'exécutent en quelques millisecondes. Cette rapidité permet d'en réaliser plusieurs centaines et de les relancer à chaque modification du code.

2.2. Ce que les tests unitaires permettent de vérifier

- **Détection des régressions :** une modification qui altère un comportement précédemment vérifié entraîne l'échec du test correspondant.
- **Documentation vivante :** chaque test décrit un comportement attendu. Cette documentation reste vérifiable puisqu'elle est exécutée automatiquement.
- **Sécurisation des évolutions :** les tests permettent de réorganiser ou de modifier un traitement tout en vérifiant qu'il continue à produire les résultats attendus.

2.3. Limites des tests unitaires

Les tests unitaires ne garantissent pas l'absence de défauts.

- **Périmètre limité :** ils ne vérifient que les cas effectivement couverts par les tests.
- **Absence de validation de l'assemblage réel :** une fonction peut fonctionner correctement de manière isolée et présenter des dysfonctionnements lorsqu'elle est reliée au véritable serveur ou aux autres composants de l'application.
- **Limites de la couverture de code :** le taux de couverture mesure la proportion de code exécutée par les tests, mais ne constitue pas une mesure directe de la qualité. Une ligne exécutée n'est pas nécessairement une ligne dont le résultat a été vérifié.

2.4. Listes, compte rendu d'exécution

https://cartographie-fonctionnelle.logeas-web.fr/UniversTests?niveau=1

3. Les tests d'interface (E2E)

3.1. Définition

Un test de bout en bout (*End-to-End* ou E2E) vérifie le comportement de l'application complète, du point de vue d'un utilisateur.
Un navigateur est piloté automatiquement par **Playwright**.
Il charge l'interface réelle, effectue des actions (clics, saisies de texte, navigation) et vérifie les résultats effectivement affichés à l'écran.
Ces tests utilisent les véritables composants de l'application, les échanges réseau, les serveurs, la base de données et le mécanisme d'authentification. Aucune de ces couches n'est simulée.
Contrairement aux tests unitaires, qui isolent les composants pour permettre une exécution rapide, les tests E2E vérifient leur fonctionnement conjoint dans des conditions proches de l'utilisation réelle.

3.2. Ce que les tests d'interface permettent de vérifier

- **Détection des régressions d'intégration :** un test échoue lorsque l'assemblage réel (interface, service, appel serveur et base de données) ne fonctionne plus, même si chaque composant fonctionne individuellement.
- **Validation des parcours utilisateur :** les tests vérifient qu'un enchaînement d'actions

(connexion, création d'une fiche, réponse à un ticket, etc.) produit le résultat attendu.
- **Détection des défauts d'intégration :** certains défauts ne peuvent être révélés que par une exécution avec les véritables serveurs. Ils pourraient être masqués par des simulations.

3.3. Limites des tests d'interface

- **Durée d'exécution :** un scénario peut prendre plusieurs secondes, voire plusieurs minutes. Il n'est donc pas possible de couvrir autant de cas qu'avec les tests unitaires.
- **Dépendance à l'environnement réel :** les délais réseau, l'état partagé des données et les variations de temps d'exécution peuvent introduire une certaine instabilité. Un test qui échoue puis réussit lors d'une nouvelle exécution doit être analysé et non ignoré.
- **Périmètre du scénario :** la réussite d'un scénario ne garantit que le bon fonctionnement du parcours effectivement testé.

Les tests d'interface constituent donc un complément indispensable aux tests unitaires, mais ne les remplacent pas.

3.4. Listes, compte rendu d'exécution

Dans LoGeAs on en distinct deux catégories :

- Les tests d'interface proprement dit : https://cartographie-fonctionnelle.logeas-web.fr/UniversTests?niveau=21
- les tests de stress et de charge : https://cartographie-fonctionnelle.logeas-web.fr/UniversTests?niveau=4

4. Tests automatiques

4.1. Définition

A chaque “Génération des écritures”, opération qui consiste à transformer dans LoGeAs les saisies comptables en écritures comptable, mais aussi lors du charge de la base des tests sont réalisés. Leur vocations est de vérifier la bonne santé des données que l'origine soit une erreur de développement (par exemple : une personne non lié à une famille) ou une erreur de l'utilisateur (par exemple : une écriture comptable non équilibrée)

4.2. Listes, compte rendu d'exécution

https://cartographie-fonctionnelle.logeas-web.fr/UniversTests?niveau=3

5. Complémentarité des tests

Les tests unitaires et les tests d'interface répondent à des objectifs différents et complémentaires.

Type de test	**Périmètre vérifié**	**Vitesse d'exécution**	**Outils et emplacement dans LoGeAs Web**
Test unitaire	Une unité de code isolée, avec des dépendances simulées	Quelques millisecondes	Jest, dépôts `logeas-web` (application) et `logeas-lib` (bibliothèques partagées)
Test d'interface (E2E)	L'application complète, pilotée par un navigateur réel	De quelques secondes à plusieurs minutes	Playwright, dossier `tests/` du dépôt `logeas-web`
Test de charge et de stress	L'application complète, pilotée par un navigateur réel, mais dans des volumes limites	De quelques minutes à quelleques heures	Playwright, dossier `tests-chargestress/` du dépôt `logeas-web`
Test de “génération”	Réaliser automatiquement dans le courant de l'usage du logiciel	De quelques secondes à plusieurs minutes	divers dossiers

Les tests unitaires permettent de vérifier rapidement un grand nombre de cas.
Les tests d'interface en vérifient un nombre plus limité, mais dans des conditions réelles d'utilisation.
les test de génération vérifie dans l'usage les problèmes pouvant survenir de la part du logiciel et/ou de l'utilisateur

Aucun de ces types de tests ne remplace :

- la revue humaine du code ;
- la validation fonctionnelle de l'application.

6. Organisation et exécution des tests

6.1. Une exécution complète par version

Un script unique permet d'exécuter automatiquement l'ensemble des tests d'une version, dans l'ordre suivant :

- Démarrage des serveurs locaux dédiés aux tests. Les serveurs de production ne sont jamais utilisés.
- Exécution des tests unitaires des deux dépôts, avec mesure de la couverture de code.
- Exécution des tests d'interface sur une base de données de test créée spécifiquement pour cette exécution.
- Production du rapport qualité de version et analyse des résultats.

Cette procédure permet de disposer d'un rapport consolidé des différents types de tests réalisés sur une version candidate.

6.2. Organisation des tests d'interface

Les scénarios de tests d'interface sont organisés en groupes exécutés successivement :

- Mise en place de l'environnement.
- Connexion.
- Administration.
- Fichier.
- Tickets.
- Fin de l'exécution : sauvegarde et archive fiscale.

Les règles d'exécution sont les suivantes :

- Si la mise en place de l'environnement échoue, les groupes suivants ne sont pas exécutés.
- Un échec dans un groupe n'empêche pas l'exécution des autres groupes.
- Chaque fichier de scénario comporte l'identifiant du test correspondant dans la cartographie fonctionnelle.

Par exemple, le fichier `03-T92-reponse-ticket-client.spec.ts` correspond au test numéro 92 de la cartographie.
Le résultat de chaque scénario est enregistré automatiquement dans le suivi des tests de la cartographie fonctionnelle.

7. Lecture d'un rapport qualité de version

Chaque rapport qualité de version présente les informations nécessaires à l'identification du code testé, à l'interprétation des résultats et à leur validation.

7.1. Version

La version indiquée correspond à la **version candidate**, c'est-à-dire à la prochaine version bêta que deviendra le code testé.
Il ne s'agit donc pas nécessairement de la dernière version bêta déjà livrée.

7.2. Clé du code

La clé du code identifie précisément les éléments ayant fait l'objet des tests :

- les versions exactes (commits) des deux dépôts ;
- l'empreinte des paquets de la bibliothèque effectivement installés.

Un rapport qualité de version ne vaut que pour la clé de code qu'il identifie.
Deux situations particulières sont signalées :

- **Modifications locales non commitées :** le code testé comporte des modifications qui ne correspondent exactement à aucun commit.
- **Code modifié pendant le run :** le code a été modifié entre le début et la fin de l'exécution des tests.

Ces informations sont importantes pour apprécier la représentativité des résultats par rapport au code destiné à être livré.

7.3. Origine de l'exécution

Le rapport distingue deux origines possibles :

- **Qualification :** exécution réalisée par le script de qualification, avec vérification des dépôts et reconstruction de la bibliothèque. Seule cette origine est admise pour la livraison d'une version bêta.
- **Run complet :** exécution complète réalisée dans le cadre du travail courant, sans constituer nécessairement une qualification officielle.

7.4. Verdict

Le verdict synthétise le résultat de l'exécution :

- **Tous les tests passent :** tous les tests exécutés sont réussis.
- **Échecs :** un ou plusieurs tests ont échoué.
- **Run incomplet :** une ou plusieurs familles de tests n'ont pas pu être exécutées.

Le verdict doit être interprété en tenant compte des éventuelles suites non exécutées et des tests instables.

7.5. Suites de tests unitaires non exécutées

Une suite de tests unitaires non exécutée correspond à un fichier de tests qui n'a pas pu démarrer, par exemple en raison d'une erreur de compilation ou d'un plantage.
Il ne s'agit pas d'un test en échec, mais d'un ensemble de tests qui n'a effectué aucune vérification.
Cette situation doit être analysée au même titre qu'un échec de test.

7.6. Tests d'interface instables

Un test d'interface est considéré comme instable lorsqu'un scénario échoue lors d'une première exécution, puis réussit lors d'une nouvelle exécution.
Cette situation doit faire l'objet d'une analyse afin d'identifier une éventuelle instabilité de l'environnement ou un défaut intermittent.

7.7. Sections d'analyse

Le rapport comporte plusieurs sections d'analyse :

- Synthèse.
- Analyse des résultats.
- Défauts.
- Écarts.
- Réserves.

Ces sections sont rédigées à partir des résultats des tests, de l'historique du code et du rapport de la version précédente.
Elles font ensuite l'objet d'une relecture et d'une validation par l'équipe.

7.8. Historique des rapports

Les rapports historiques repris des anciens rapports de release présentent uniquement les chiffres des tests unitaires.
Ils ne comportent ni analyse ni validation. Il convient donc de tenir compte de cette différence lors de leur consultation et de leur comparaison avec les rapports qualité actuels.

8. Validation des rapports qualité de version

8.1. Principe de validation

Chaque rapport qualité de version est validé dans l'écran **« Rapports de version »** de la cartographie fonctionnelle.
La validation est réalisée par un utilisateur connecté et comporte les informations suivantes :

- la décision : validé, validé avec réserves ou refusé ;
- l'auteur de la validation ;
- la date de validation ;
- un commentaire éventuel ou obligatoire selon la décision.

La validation porte sur une exécution précise des tests et sur le code correspondant.

8.2. Nouvelle exécution des tests

Si les tests sont relancés sur une même version, un nouveau rapport est généré.
Ce nouveau rapport doit faire l'objet d'une validation distincte.
Les rapports précédents restent consultables et sont marqués comme **« remplacés »**.

8.3. Publication du rapport

Tout rapport validé, avec ou sans réserves, est publié simultanément à sa validation.

8.4. Conditions de validation avec réserves

Une validation avec réserves n'est admise que si chaque échec est rattaché :

- à un ticket identifié ;
- ou à un échec déjà connu et documenté.

Tout échec nouveau par rapport au dernier rapport validé entraîne un refus de validation.

Un commentaire est obligatoire pour toute validation avec réserves ou tout refus.

9. Chaîne de qualification et de livraison

9.1. Principe général

Aucune version ne peut être livrée sans disposer d'un rapport qualité validé correspondant exactement au code concerné.
La chaîne de qualification et de livraison comprend les étapes suivantes.

9.2. Étape 1 : qualification

La qualification est réalisée à l'aide du script `qualifier-version`. Les conditions suivantes doivent être réunies :

- les deux dépôts doivent être positionnés sur leur branche de version ;
- aucune modification locale non commitée ne doit être présente ;
- les dépôts doivent être à jour et leurs modifications poussées ;
- la bibliothèque doit être reconstruite à partir de son dernier commit.

Une fois ces conditions vérifiées, le script exécute l'ensemble des tests et produit le rapport qualité de la version candidate.

9.3. Étape 2 : validation du rapport

Le rapport de qualification doit être validé dans l'écran **« Rapports de version »** de la cartographie fonctionnelle.
Cette validation constitue un préalable obligatoire à la production de la version bêta.

9.4. Étape 3 : livraison de la version bêta

La production de la version bêta est réalisée par le script `livrer-beta`.
Cette étape constitue une **porte bloquante** : la bêta ne peut être produite que si un rapport de qualification validé correspond exactement au code présent dans les deux dépôts.
La livraison de la bêta est automatiquement inscrite dans la trace du rapport.

9.5. Étape 4 : recette de la version bêta

Après le déploiement de la bêta sur le serveur de test, une recette est réalisée dans l'écran **« Rapports de version »** de la cartographie fonctionnelle.
Cette recette repose sur une liste de contrôle minimale.

Chaque point de cette liste doit être :

- vérifié et déclaré conforme ;
- ou déclaré sans objet, avec une justification.

Une recette déclarée non conforme nécessite obligatoirement un commentaire.

9.6. Étape 5 : livraison de la version client

La livraison de la version client (release) est réalisée par le script `livrer-release`.
Cette étape constitue également une **porte bloquante**.
Les conditions suivantes doivent être réunies :

- la recette de la version bêta doit être conforme ;
- aucun nouveau commit ne doit avoir été ajouté dans l'un ou l'autre des deux dépôts depuis la production de la bêta.

Le rapport qualité publié constitue alors la pièce qualité de la release.

9.7. Gestion des correctifs après livraison d'une bêta

Tout correctif apporté après la production d'une bêta, même mineur, impose une nouvelle qualification et la production d'une nouvelle bêta.
Cette règle garantit que la version livrée a bien été testée et validée sur la base du code exact qui la constitue.

9.8. Gestion des livraisons d'urgence

Une livraison réalisée en dehors de cette chaîne reste possible en cas d'urgence.
Dans ce cas, le rapport de release et l'écran **« À propos »** de LoGeAs Web indiquent explicitement que la version est **non qualifiée**.
Cette mention permet de distinguer une version ayant suivi l'ensemble de la procédure de qualification d'une version livrée exceptionnellement en dehors de celle-ci.

From:
https://wiki-logeas.fr/certif/ - **dokuwiki-certif**

Permanent link:
https://wiki-logeas.fr/certif/doku.php?id=certif:dm:methodologie_des_tests&rev=1790666975

Last update: **2026/09/29 09:29**