PLAN DE TEST

David EVAN

10/05/2022

Version 1.0

Projet de streaming vidéo interactif - GIBBERISH.NET

Historique des révisions

Numéro de version	Auteur	Description	Date de modification
1.0	EVAN David *(Architecte logiciel)*	Livraison initiale	10/05/2022

Tableau 1 : Historique des révisions

Objectif du document

L'objectif de ce plan de test est défini le processus de validation permettant de s'assurer de la conformité de l'architecture cible avec les besoins exprimés dans le cahier des charges d'architecture.

Les objectifs à atteindre, les fonctionnalités attendues pour le projet et la conformité de l'architecture cible sont rappelé.

La méthodologie de test retenue, incluant l'approche, les niveaux de tests à réaliser et l'outillage, sera précisée.

TABLE DES MATIERES

INTRODUCTION

Rappel du contexte

Dans le cadre de l'extension de ses activités, l'entreprise Gibberish.net souhaite développer une nouvelle plateforme de streaming un service interactives.

Afin de cadrer le projet, plusieurs livrables ont déjà été produit :

- **Le cahier des charges d'architecture** présente les principaux attendus fonctionnels et non-fonctionnels à atteindre.
- **Le document de définition d'architecture** définit la conception retenue pour la nouvelle plateforme vidéo.

Le présent document propose une méthodologie afin de s'assurer que l'architecture produite sera conforme à la conception détaillée dans ces livrables.

Objectifs des tests

L'objectif des tests est d'assurer la conformité de l'architecture par rapport aux normes, principes et exigences définies. Il convient de s'assurer que :

- Les anomalies (bug) et écarts fonctionnels sont identifiés et corrigés avant la mise en ligne.
- Les composants testés sont conformes aux exigences fonctionnelles et non fonctionnelles (telles que définies dans le cahier des charges d'architecture).
- S'assurer que la nouvelle architecture répond aux spécifications de qualité définies par la société et aux normes en vigueur dans l'industrie.
- S'assurer que l'architecture est en conformité avec la réglementation, notamment en matière de protection des données personnelles.

Rôles et responsabilités des parties prenantes

Dans le cadre du plan de test, la matrice ci-après défini les rôles et responsabilités des différentes parties prenantes intervenant dans sa mise en œuvre.

Partie prenante	Rôle - Responsabilités
Architecte logiciel	- Établit la stratégie des tests - Définit les objectifs à atteindre - Évalue la conformité de l'architecture et identifie les écarts - Définition des plans de correction pour les écarts d'architecture
Responsable d'exploitation	- Construit et s'assure que l'environnement de test et les actifs sont gérés et maintenus (machines, environnements, liens réseaux, compte de service …) - Fournit le support pour l'utilisation et l'exploitation de l'environnement de test.
Responsable des tests	- Pilote la stratégie de test : Gestion du planning - budget, suivi de l'avancement, distribution des tâches, coordination des équipes. - Suivi et validation de la conformité des tests. - Collecte des résultats des tests et consolide les rapports. - Reporting sur les écarts constatés
Testeurs fonctionnels	- Participe à l'écriture des cas de test, - Exécute les cas de tests avec les paramètres définis - Consigne les résultats et rapporte les anomalies
Développeurs	- Programme les tests unitaires / fonctionnels automatisés - Corrige les anomalies et écarts identifiés.

Tableau 2 : Définition des rôles et responsabilités des parties prenantes

FONCTIONNALITÉS À TESTER

Architecture retenue

L'architecture décrite dans le Document de Définition d'Architecture permet de satisfaire aux exigences fonctionnelles telles que décrites dans le cahier des charges d'architecture.

Le tableau ci-dessous met en relation les briques de solution d'architecture avec les besoins fonctionnels. Notons que les composants non cités permettent principalement de satisfaire aux exigences non fonctionnelles (performance, capacité d'accueil …) de la plateforme vidéo.

Id. Fonc.	Fonctionnalité	Id Comp.	Composant
BR-1	Téléchargements en utilisant les liens à l'intérieur de la vidéo.	**SSB-6**	Plateforme vidéo
BR-2	Modification de l'histoire en fonction des choix de l'utilisateur.	**SSB-1** **SSB-6**	- Production de média (Unity) - Plateforme vidéo (+ NexPlayer)
BR-3	Rotation interactive tridimensionnelle (vidéo 360).	**SSB-1** **SSB-6**	- Production de média (Unity) - Plateforme vidéo (+ NexPlayer)
BR-4	Un menu permet de passer directement à un segment spécifique de la vidéo.	**SSB-6**	Plateforme vidéo (+ NexPlayer)
BR-5	Saisie des données et des commentaires des utilisateurs dans la vidéo.	**SSB-6**	Plateforme vidéo
BR-6	Génération dynamique de vidéos personnalisées.	**SSB-6**	Plateforme vidéo
BR-7	Sécurisation de parties de la vidéo avec un mot de passe.	**SSB-5** **SSB-6**	- Authentification / Autorisation (IAM) - Plateforme vidéo (+ NexPlayer)
BR-8	Capacité à activer des zones cliquables pour avoir du contenu additionnel.	**SSB-1** **SSB-6**	- Production de média (Unity) - Plateforme vidéo (+ NexPlayer)
BR-9	Répondre à un quizz de type questions à choix multiples ou vrai/faux.	**SSB-1** **SSB-6**	- Production de média (Unity) - Plateforme vidéo (+ NexPlayer)
BR-10	Diffusion de la vidéo personnalisée en générant un lien de partage ou en intégrant la vidéo directement dans son site internet.	**SSB-4**	Diffusion de média sur le web
BR-11	Intégration de segments de publicité « shoppable » dans la vidéo.	**SSB-1** **SSB-6**	- Production de média (Unity) - Plateforme vidéo (+ NexPlayer)
BR-12	Mise à disposition de vidéo au format « vertical » ou « horizontal » adapté aux mobiles.	**SSB-1** **SSB-6**	- Production de média (Unity) - Plateforme vidéo (+ NexPlayer)

Tableau 3 : Matrice de mise en relation des fonctionnalités et des composants d'architecture

Conformité de l'architecture

Le Document de Définition d'Architecture propose conception architecturale conforme aux besoins exprimés dans le ***Cahier des Charges d'Architecture*** est en adéquation avec les principes définis dans le référentiel d'architecture de l'entreprise.

La lecture de ces documents permet de valider la conformité.

PLAN DE TEST

Périmètre

Les tests vont porter sur les nouveaux composants ou les composants ou les composants existants modifiés qui seront impactés par le nouveau service de streaming de vidéo interactives, et notamment :

- Outils de conception de vidéo interactives
- Accès et diffusion des médias interactifs
- Nouvelle plateforme de diffusion de vidéo

Les composants existants et non modifiés (lié à la production / diffusion de média non interactifs par exemple) ne feront pas partie du périmètre de ce plan de test.

Méthodologie

Approche « Business Driven Development »

Le plan de test retenu pour ce projet est basé sur la méthodologie dite **de Business-Driven Development** (BDD). Dans cette approche BDD, les règles métiers permettent de structurer les tests.

***Disclamer** : Cette section s'appuie en partie sur la documentation disponible sur les guides du projet « Cucumber.io ». https://cucumber.io/docs/cucumber/.*

Caractéristiques de l'approche « Busines Driven Development »

L'approche BDD se caractérise par :

- Les fonctionnalités sont retenues en fonction de leur valeur ajoutée commerciale (« business »).
- Mise en relation et collaboration des parties prenantes du projet (Analystes fonctionnels, testeurs, responsables qualité, architectes, développeurs …).
- Utilisation d'un langage universel facile à décrire (ex : syntaxe Gherkin) adapté à toutes les parties prenantes (À la différence de l'approche TDD seule qui est très orienté « développeur » et basée uniquement sur des tests implémentés dans le code avec peu de visibilité « métier »).
- Travaille en petites itérations rapides pour augmenter le retour et le flux de valeur. (Approche « agile »).

Les méthodes « Agile » et d'itération sont au cœur de l'approche « BDD ». L'approche BDD encourage le travail par itérations, en décomposant continuellement les problèmes des utilisateurs en petites sections qui peuvent facilement être intégrées au processus de développement et implémentées aussi rapidement que possible.

Cycle itératif de développement du produit

L'activité BDD est un processus itératif en trois étapes :

1. Réflexion sur les changements ou ajouts à apporter au système et la valeur ajoutée pour l'utilisateur. Ces changements sont décrits sous la forme de cas d'utilisation pour l'utilisateur (User-Story).
2. Documentation des exemples d'une manière qui peut permettre l'automatisation et la vérification des résultats avec les critères d'acceptation.
3. Implémentation du comportement décrit de chaque US documenté, en commençant par un test automatisé dans le code (Approche TDD)

Cette pratique est nommée « **Découverte, Formulation et Automatisation** » (*Discovery, Formulation, and Automation*)

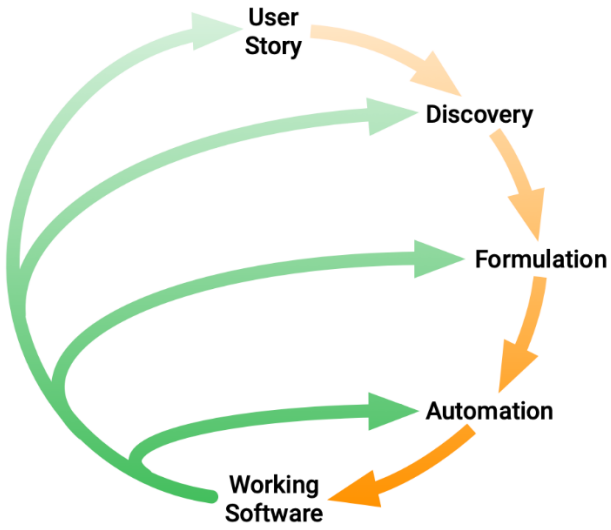

Figure 1 : Discovery, Formulation, and Automation dans l'activité de BDD. (Source: cucumber.oi)

Discovery : « What it could »

L'étape de découverte permet de répondre à la question « Ce que ça pourrait faire ». BDD aide les équipes à communiquer de manière adéquate, BDD permet aux parties impliquées de minimiser le temps qu'elles passent en réunion et de maximiser le degré de valeur contenu dans le code final.

Les conversations structurées, également connues sous le nom d'ateliers de découverte qui entourent des exemples réels, aident les parties à se rapporter au système du point de vue de l'utilisateur.

Cette approche permet également de combler les écarts dans la compréhension des membres de l'équipe et d'apporter plus de clarifications pour éviter un goulot d'étranglement dans le projet.

Formulation : « What it should do »

La formulation se préoccupe de « Ce que ça devrait faire ». Après avoir identifié un exemple pratique à partir de la session de découverte, chacun peut maintenant être formulé sous forme de documentation structurée.

Cela fournit un moyen rapide de prouver que toute l'équipe a une compréhension commune de ce qu'il faut construire. Chaque membre de l'équipe peut donner son avis sur l'objectif commun du projet de développement. Aussi, cette approche permet à l'équipe d'automatiser les exemples et de s'en servir comme guide pour développer une solution logicielle complète pour les objectifs requis.

Pour rédiger les cas de test BDD pour un Use Case on utilisera la syntaxe du modèle Gherkin avec la formule **Given-When-Then** (formulation déjà utilisée dans le cahier des charges d'architecture).

Automation : « What it really does »

"Ce qu'il fait vraiment". En choisissant les exemples au fur et à mesure, la mise en œuvre est automatisée lorsqu'elle est connectée en tant que test au système. L'automatisation des tests en BDD est une approche des tests dans laquelle les pratiques d'assurance qualité (QA) sont mises en œuvre et mesurées par rapport aux objectifs commerciaux avec un objectif de test défini. L'ensemble du processus de test est piloté par des métriques commerciales dérivées d'objectifs commerciaux spécifiques.

Dans cette approche de test, l'équipe d'assurance qualité et les analystes fonctionnels s'impliquent dès le démarrage du projet pour définir les règles métier ainsi que des scénarios de test et générer les scripts automatisés qui sont ensuite exécutés dans l'application.

Cette approche confère une grande flexibilité à l'équipe d'assurance qualité car les scripts de test peuvent être modifiés pour s'adapter à l'évolution des besoins d'automatisation de l'entreprise.

Si le test échoue, cela implique simplement que la fonctionnalité n'a pas été implémentée.

Un code d'implémentation est ensuite développé et des exemples de niveau inférieur représentant le comportement des composants internes du système sont utilisés pour une orientation appropriée.

Ces exemples automatisés aident l'équipe à poursuivre le processus de développement.

Les exemples automatisés aident par ailleurs l'équipe de développement à comprendre ce que fait le système lorsqu'il est nécessaire de maintenir le système plus tard. Ils permettent également d'apporter des modifications sans détruire la libre circulation du système.

Implémentation et enchaînement du cycle de développement

Le schéma ci-après présente une vue simplifiée de l'implémentation concrète du cycle de développement itératif tel que décrit dans la section précédente.

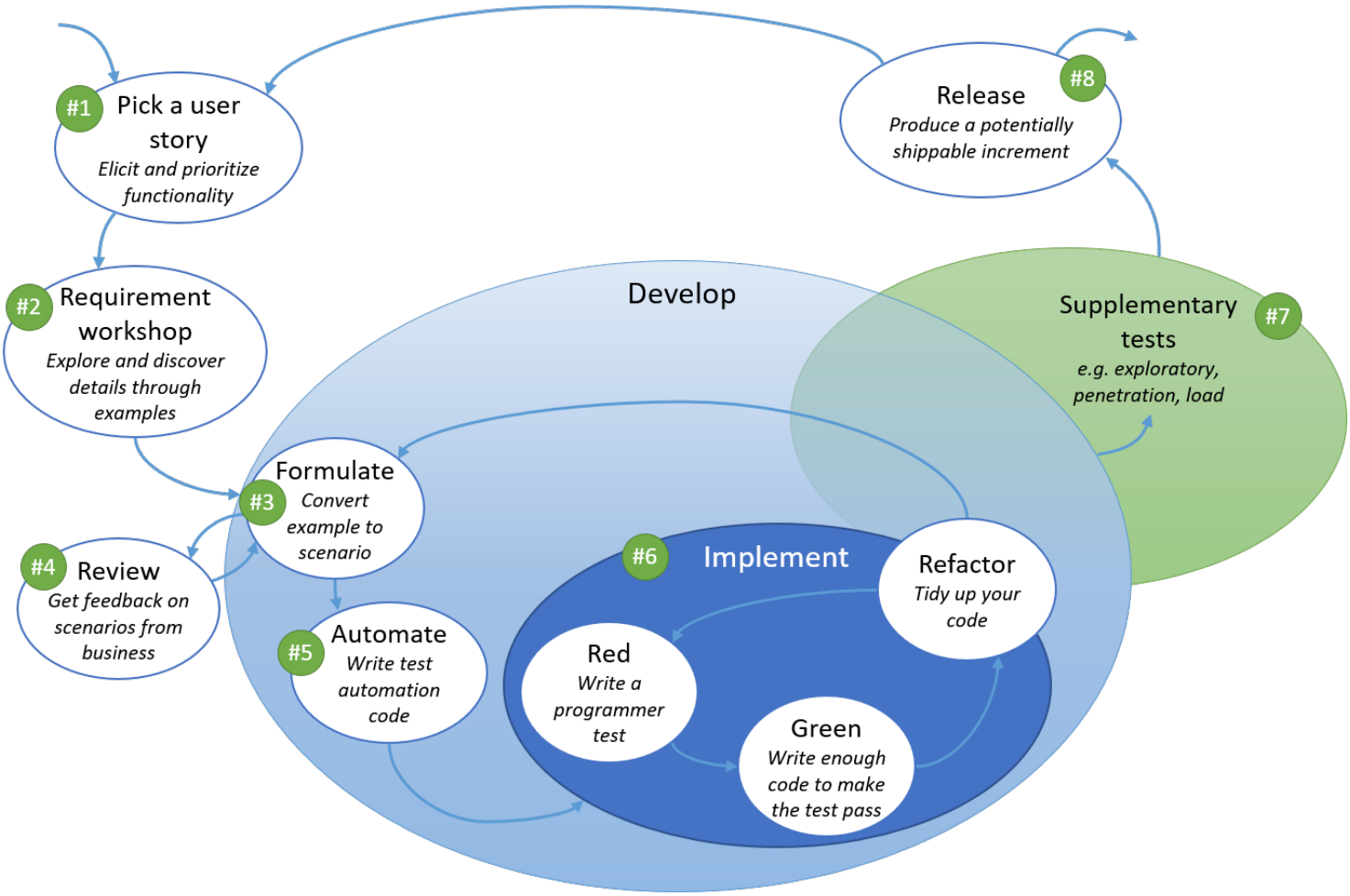

Figure 2 : Implémentation du cycle de développement itératif (source : Medium.com/@hgsgtk)

TDD - Londre VS Chicago

L'implémentation du BDD suggère l'utilisation du avec une approche « test-first » lors de l'écriture du code. Deux écoles s'affrontent [1]pour la mise en place de ce TDD : Une approche « business-first » (dite « *London* », qui vise à se concentrer en premier sur l'extérieur de l'application (API – Controller) pour ensuite descendre sur la couche model, et une approche inverse dite « *Chicago* ».

Dans le cadre de ce projet, étant donné que l'aspect « business » et « fonctionnel » est préférée par rapport à l'ingénierie logicielle, l'approche « London » sera recommandé.

[1] Pour aller plus loin: https://devlead.io/DevTips/LondonVsChicago

Niveaux des tests

De manière générale, un projet de développement logiciel contient des tests de différentes natures qui, pour répondre aux exigences d'une conception de qualité, doivent être présents et exécutés à chaque itération du produit.

La quantité de tests généralement attendus en fonction de leur nature peut être représentée sous la forme d'une pyramide.

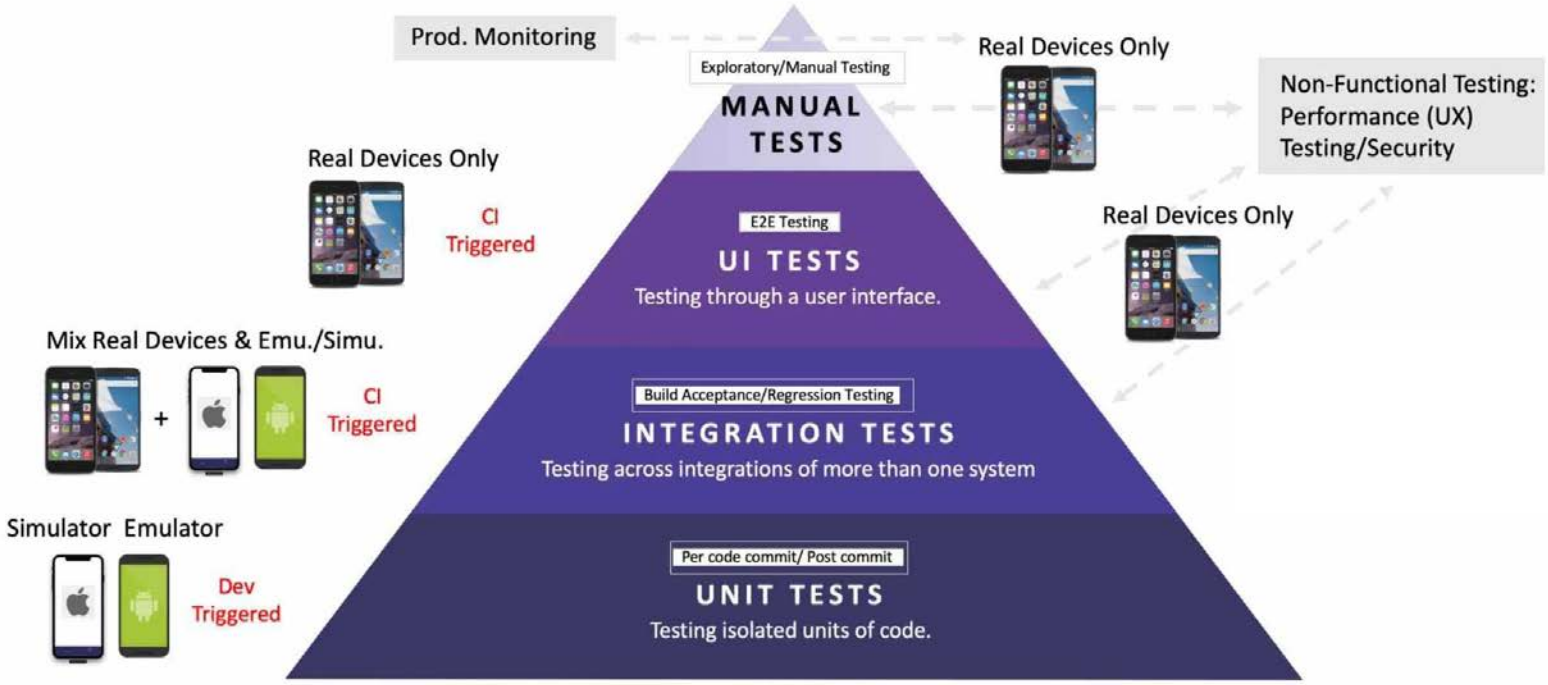

Figure 3 : ISTQB - Testing pyramide (Source : perfecto.io)

Le plan de test du projet de streaming vidéo vise à couvrir de **manière automatique** les trois premiers échelons de la pyramide.

Afin de maximiser le ROI et la production de valeur, ces tests seront complétés par des tests fonctionnels lancés manuellement par les équipes de testeur pour s'assurer de la conformité du produit aux exigences fonctionnelles.

Tests de composants

Les tests de composants ont pour but de tester les différents composants du logiciel séparément afin de s'assurer que chaque élément fonctionne comme spécifié. Ces tests sont aussi appelés tests unitaires et sont généralement écrits et exécutés par le développeur qui a écrit le code du composant.

Chaque composant impacté par le nouveau service sera testé lors des tests fonctionnels dans les différentes phases de test et en particulier lors des tests de composants.

Sa maintenance sera assurée par l'automatisation des tests lors des tests de non-régression.

Pour qu'un composant soit validé, il faut que tous les tests fonctionnels et non fonctionnels (contraintes techniques) qui lui sont associées soient validés.

Tests d'intégration

Les tests d'intégrations sont des tests effectués entre les composants afin de s'assurer du bon fonctionnement des interactions et de l'interface entre les différents composants (ex : API). Ces tests sont également gérés par des développeurs.

Ces tests permettent de s'assurer du bon fonctionnement des différents systèmes les uns par rapport aux autres et de la conformité de l'architecture logicielle.

Tests d'interface utilisateur (UI)

Les tests d'interface utilisateur vise à s'assurer de la conformité de l'interface UI avec les exigences spécifiées. Ces tests peuvent par ailleurs permette d'automatiser un certain nombre de tests fonctionnels en vérifiant que les résultats produits par une action sur l'UI sont conformes à l'attendu.

Le test d'un formulaire d'authentification peut être par exemple facilement automatisé : Les champs login / password sont-ils bien présents ? Lors du remplissage des champs avec un login / password invalide, l'utilisateur dispose-t'il bien d'un message d'erreur ? Lors de la connexion en succès, l'utilisateur est-t-il bien renvoyé vers son profil ?

Les scénarios utilisateur décrits en GWT (*Given - When - Then*) permettent de facilement automatiser ces points en définissant les critères d'acceptation et les résultats attendus après chaque action.

Tests d'acceptation (manuel)

Les tests d'acceptation sont des tests « finaux » effectués par les testeurs ou les utilisateurs finaux. Leur but est de confirmer que le produit final correspond bien **aux besoins** des utilisateurs finaux. Notons que ce n'est pas parce qu'une application répond aux spécifications qu'elle répond aux besoins des utilisateurs.

Avec ces tests on vérifie qu'en plus de répondre aux exigences, celles-ci correspondent bien à ce à quoi le métier ou les clients finaux s'attendent (Validation de la valeur métier).

Tests de charge

Il s'agit d'un test au cours duquel on va simuler un nombre d'utilisateurs virtuels prédéfinis, afin de valider l'application pour une charge attendue d'utilisateurs. Ce type de test permet de mettre en évidence les points sensibles et critiques de l'architecture technique. Il permet en outre de mesurer le dimensionnement des serveurs, de la bande passante nécessaire sur le réseau, etc.

Ces tests devraient être automatisés et exécuté tout au long du projet de développement

Tests de sécurité / test de surface

Les tests de sécurité visent à garantir la sécurité de l'application et du réseau informatique en automatisant la détection des vulnérabilités (ports réseaux, accessibilité en https seulement) et des règles métiers de sécurité (force des mots de passe).

Ils devraient être automatisés à l'aide de scanner et complétés par des campagnes de pen-testing réalisées par des experts en sécurité.

Tests de pénétration système (pen-testing)

S'intégrant dans les tests de sécurité, le pen-testing vise à tester le système informatique ou une application spécifique pour trouver les vulnérabilités de sécurité qu'un attaquant pourrait exploiter. Les tests de pénétration peuvent être automatisés ou effectués manuellement.

L'objectif principal des tests de pénétration est d'identifier les faiblesses en matière de sécurité. Le test de pénétration peut également être utilisé pour tester la politique de sécurité de l'entreprise, son respect des exigences de conformité, la sensibilisation de ses employés à la sécurité et la capacité de la société à identifier et à répondre aux incidents de sécurité.

Tests d'intégrité (tests sanitaires)

Les tests sanitaires visent à garantir qu'une modification mineures (correction de bug ou évolution fonctionnelle) n'ont pas introduit de régression dans le logiciel. Ils sont relativement limités dans leur périmètre et visent principalement à valider rapidement la réception d'un changement avant de se lancer dans une batterie de test plus complète.

Automatisation des campagnes de test

L'automatisation des campagnes de test vise à construire un environnement permettant d'exécuter les campagnes de tests programmables selon une périodicité (tous les soirs, semaines …) ou des déclencheurs définis (ex : Fin de sprint, commit GIT, fusion de branche …)

Cette automatisation nécessite l'utilisation d'outils adaptés et d'une culture assurément orienté « DevOps ». Ces outils sont décrits dans la section « environnement et outillage ».

Jeux de données de tests

Des jeux de données à multiple valeur seront définis pour chaque scénario de test. Ces jeux de données doivent permettre à la fois de valider le bon fonctionnement des règles métiers, ainsi que de prouver la validation des rejets attendus (ex : Un mot de passe « fort » est accepté, et un mot de passe « faible » est rejeté par le système).

Collecte des résultats

La collecte des résultats sera réalisée à chaque fin de sprint (durée conseillée de 2 semaines) par le responsable des tests. Ces résultats seront issus :

- Des rapports générés dans le cadre des campagnes de tests automatisés.
- Des rapports fournis par l'outil de suivi des campagnes de test manuels.
- Des rapports automatiques générés à l'aide des tableaux de suivi des bugs.

Les données ainsi collectées seront consolidées dans un rapport unique et transmis pour analyse des écarts avec l'architecture et les exigences définis (fonctionnelles et non fonctionnelles). (Voir : RACI).

Environnements et outillage

Outillage

Catégorie	Outils	Commentaire
Spécification des tests	Cucumber	Framework pour l'implémentation des scénarios de Type BDD, écrits en syntaxe Gherkin. Compatible avec la plupart des langages de développement.
Tests unitaires / Test d'intégration	Framework de test unitaire (JUnit / PHPUnit / MSTest …) + Implémentation Cucumber (via annotations)	Framework de test unitaire permettant aux développeurs d'écrire des tests programmables et utilisant plusieurs jeux de données. Peut-être utilisé avec l'implémentation Cucumber du langage retenu.
Automatisation : Tests unitaires / Tests d'intégration	GitLab (Pipelines CI)	Exécution automatisée des tests unitaires, fonctionnels et des tests d'intégrations. Création de pipeline à l'aide de Runner. Facilement customisable à l'aide des images dockers lancées. Peut être lancé à chaque « commit » ou fusion de branche pour déclencher automatiquement les campagnes de test et / ou l'analyse statique (voir ci-après).
Tests UI	Selenium Webdriver (Web) Alt Unity Tester (Unity)	Création / exécution de tests d'interface utilisateurs automatisés sur les plateforme web et Unity (voir : documentation des plug-ins).
Automatisation : Tests UI	JMeter	Exécution automatique et collecte des résultats des campagnes de tests d'UI.
Tests de sécurité	Qualys	Scanner de failles de sécurité automatisé agissant multi-surface.
Tests de charge	JMeter	Exécution de tests de charge automatisés visant à s'assurer de la capacité des logiciels testés à répondre aux pics de charge.
Test de pénétration	Burp Suite	Création / Exécution de campagne de pen-testing afin de garantir la sécurité de la plateforme.
Reporting / Analyse qualité	Sonar Qube (Intégré aux Pipeline CI GitLab)	Outil d'analyse statique fournissant des métriques sur la qualité du code, la couverture de code par les tests, les bugs – failles potentielles. Facilement customisable et pouvant être intégré aux gitlab runner pour automatiser la mise à jour des rapports à chaque « pull request ».
Suivi des bugs et des incidents	Jira Software	Outil de ticketing et de suivi de la résolution des incidents permettant de faciliter la communication inter-équipe et l'analyse des bugs.
Campagne de test manuelles	Squash TM	Création et suivi des campagnes de tests fonctionnels « manuels ». L'outil offre un excellent moyen de s'assurer du respect des exigences en les mettant en relation avec les scénarios de test.

Tableau 4 : Catalogue des outils utilisés dans le cadre du plan de test

Environnements

Dans le cadre **du plan de test**[2], plusieurs environnements seront définis pour l'exécution des campagnes de tests afin de s'assurer que les tests et les développements en cours n'auront pas d'impact sur les résultats des autres campagnes.

Le tableau ci-après présente ces environnements :

Nom de l'environnement	Type de campagne(s) exécutée(s)	Utilisateurs de référence	Fréquence mise à jour (codebase et données)	Disponibilité
DEVELOPPEMENT	- Tests unitaires (composants)	Développeurs	Chaque commit	Permanente
RECETTE	- Tests d'intégration (fonctionnels) - Tests d'UI (automatisés)	Développeurs	Chaque merge de feature (GitFlow)	Permanente
QUALIFICATION	- Tests fonctionnels manuels - Tests d'exploration - Tests d'acceptation utilisateurs - Tests d'UI (manuels)	Testeurs	Chaque release (= Fin sprint)	Permanente
PREPRODUCTION	- Tests de charge - Pen testing	Responsable d'exploitation	Variables en fonction des besoins	Variables en fonction des besoins

Tableau 5 : Catalogue des environnements disponible pour la campagne de test

Les environnements devront disposer de mécanismes permettant de réaliser facilement des duplications / rollback des jeux de données.

Le dimensionnement des environnements doit être adapté à l'objectif recherché afin d'éviter tout gaspillage de ressources matérielles ou financières tout en s'assurant que les conditions de services permettent de tester les processus de manière fiable. L'environnement de PREPRODUCTION, notamment dans le cadre des campagnes de tests de charge / tests de performance devrait être au plus proche de l'environnement de production.

Les jeux de données peuvent être répliquées d'un environnement à l'autre. Toutes données issues d'un environnement de production et contenant des DCPs devra cependant passer **obligatoirement** par un processus d'anonymisation **à l'aide d'un algorithme non prédictif**.

[2] D'autres environnements sont susceptibles d'être définis pour les besoins du projet, indépendamment des tests.

Livrables

Il existe différents livrables concernant les tests et qui seront fournis pour chaque phase du cycle de vie du développement logiciel.

Les livrables de test fournis avant la phase de test.

- Stratégie de test
- Plan de test
- Cas de test

Les livrables de test fournis pendant les tests

- Liste des scénarios éligibles à l'automatisation
- Scripts de test
- Données de test utilisées en entrée
- Matrice de traçabilité des exigences
- Journaux d'erreurs et journaux d'exécution.

Les livrables de test sont fournis une fois les cycles de test terminés.

- Rapports d'incidents des tests
- Rapport de synthèse des tests
- Directives relatives aux procédures d'installation/de test
- Matrice de traçabilité des exigences
- Métriques et mesure de test

Tests des exigences fonctionnelles

Fonctionnalité	**BR1 : Téléchargements en utilisant les liens à l'intérieur de la vidéo**
User Story	En tant qu'utilisateur, je souhaite pouvoir télécharger un fichier (PDF, vidéo) afin d'obtenir des informations complémentaires ou de récupérer une vidéo (exemple ma vidéo interactive personnalisée).
Scénario S1-1	Un utilisateur clique sur un lien de téléchargement
GWT	***GIVEN*** : Je visualise une vidéo qui contient un lien de téléchargement ***WHEN*** : j'active le lien de téléchargement ***AND*** : je choisis la destination (locale ou Cloud) ***THEN*** : je récupère un fichier (PDF, Vidéo) ***AND*** : le contenu de ce fichier est conforme et intègre.

Tableau 6 : BR1 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	**BR2 : Modification de l'histoire en fonction des choix de l'utilisateur**
User Story	En tant qu'utilisateur, je peux décider du scénario de l'histoire que je regarde parmi un choix proposé par le système.
Scénario S2-1	L'utilisateur décide du scénario
GWT	***GIVEN :*** je visualise une vidéo qui s'interrompt pour me proposer plusieurs scénarios pour la suite de ma vidéo ***WHEN*** : je sélectionne un des scénarios proposés ***THEN :*** une nouvelle vidéo se charge correspondant au scénario choisi
Scénario S2-2	Le système décide du scénario
GWT	***GIVEN*** : je visualise une vidéo qui s'interrompt pour me proposer plusieurs scénarios pour la suite de ma vidéo ***AND :*** 20 secondes s'écoulent sans que j'aie sélectionné un scénario ***WHEN :*** le système choisi à ma place un des scénarios ***THEN :*** une nouvelle vidéo se charge correspondant au scénario choisi

Tableau 7 : BR2 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	**BR1 : Rotation interactive tridimensionnelle (vidéo ou image 360)**
User Story	En tant qu'utilisateur, je souhaite effectuer une rotation de ma vidéo ou image 360° pour changer mon axe de visualisation.
Scénario S3-1	Un utilisateur se déplace dans une vidéo ou image 360
GWT	***GIVEN :*** je visualise une vidéo ou une image tridimensionnelle qui est entièrement couverte de zones actives ***WHEN*** : j'active la rotation avec les options de déplacement dans la vidéo ou image ***THEN :*** l'image 360 effectue une rotation dans le sens et la direction de la zone activée

Tableau 8 : BR3 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	**BR4 : Passer directement à un segment spécifique de la vidéo**
User Story	En tant qu'utilisateur, je souhaite passer directement à un segment spécifique de la vidéo
Scénario S4-1	Un utilisateur accède directement à un segment de la vidéo
GWT	***GIVEN*** : Je visualise une vidéo qui contient un lien de téléchargement ***WHEN*** : j'active le lien de téléchargement ***AND*** : je choisis la destination (locale ou Cloud) ***THEN*** : je récupère un fichier (PDF, Vidéo) ***AND*** : le contenu de ce fichier est conforme et intègre

Tableau 9 : BR 4 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	**BR5 : Discussion instantanée par système de chat intégré**
User Story	En tant qu'utilisateur, je souhaite discuter avec d'autres utilisateurs pour échanger sur la vidéo diffusée en direct
Scénario S5-1	Des utilisateurs échangent des commentaires en direct sur une vidéo
GWT	***GIVEN :*** je visualise une vidéo qui contient un chat intégré ***WHEN :*** *je me suis identifié avec un pseudo* ***AND :*** *je poste des commentaires* ***THEN :*** le fil de discussion en direct est visible (mes commentaires et ceux des autres utilisateurs sont visibles)

Tableau 10 : BR5 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	BR6 : Génération dynamique de vidéos personnalisées
User Story	Le service, récupère automatiquement les données du Client (dans une base de données propriétaire) afin de générer dynamiquement une vidéo personnalisée (nom, prénom, contrat du client)
Scénario S6-1	Le service génère dynamiquement des vidéos personnalisées
GWT	***GIVEN :*** le service accède à une vidéo Template contenant des champs génériques ***AND :*** le service accède aux données du client X concerné ***WHEN*** : le service est déclenché automatiquement ou manuellement, ***THEN :*** la vidéo générée est personnalisée quand elle contient les données du client X

Tableau 11 : BR6 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	BR7 : Sécurisation de parties de la vidéo avec un mot de passe
User Story	En tant qu'utilisateur, je souhaite accéder à une partie de la vidéo protégée par un mot de passe
Scénario S7-1	Un utilisateur accède à un contenu protégé
GWT	***GIVEN :*** je visualise une vidéo protégée qui s'interrompt pour me demander de saisir un mot de passe ***WHEN*** : je saisis un mot de passe valide ***THEN :*** je peux visualiser la partie protégée de la vidéo

Tableau 12 : BR7 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	BR8 : Contenu additionnel par activation de zones cliquables
User Story	En tant qu'utilisateur, je souhaite avoir un accès rapide à du contenu additionnel (image, texte, son, lien hypertexte) à certains endroits ou moments de la vidéo ou du média 360°
Scénario S8-1	Un utilisateur accède à du contenu additionnel
GWT	***GIVEN :*** je visualise une vidéo qui contient des zones cliquables (points actifs) ***WHEN*** **:** j'active une zone cliquable ***THEN*** **:** j'accède à l'information supplémentaire

Tableau 13 : BR7 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	**BR9 : Répondre à un quizz**
User Story	En tant qu'utilisateur, je souhaite répondre à des questions (choix multiples ou vrai/faux) afin par exemple d'évaluer ma bonne compréhension du sujet présenté par la vidéo (e-learning)
Scénario S9-1	Un utilisateur répond à un quizz intégré dans la vidéo
GWT	***GIVEN :*** je visualise une vidéo qui s'interrompt et affiche une question (choix multiple ou vrai/faux) ***WHEN :*** je sélectionne et valide ma/mes réponses ***THEN :*** si mes réponses sont valides j'ai un message indiquant que c'est correct sinon j'ai un message indiquant que c'est incorrect et m'indique la/les bonnes réponses.

Tableau 14 : BR9 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	**BR10 : Partage de vidéos personnalisées**
User Story	En tant qu'utilisateur, je souhaite partager ma vidéo personnalisée en générant un lien de partage ou en intégrant ma vidéo directement dans un site internet
Scénario S10-1	Un utilisateur partage sa vidéo avec un lien
GWT	***GIVEN*** : je visualise ma vidéo personnalisée contenant un lien de partage ***WHEN*** : j'active l'option partager la vidéo avec un lien ***THEN :*** ma vidéo est partagée directement sur mon réseau social si j'ai choisi cette option sinon mon presse papier contient l'URL de partage de ma vidéo
Scénario S10-2	Un utilisateur partage sa vidéo avec son code source
GWT	***GIVEN :*** je visualise ma vidéo personnalisée contenant un lien pour récupérer le code source ***WHEN*** : j'active l'option partager la vidéo avec le code source ***THEN :*** mon presse papier contient le script permettant d'intégrer la vidéo directement dans un site internet

Tableau 15 : BR10 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	BR11 : Publicité et vidéo shoppable
User Story	En tant qu'utilisateur, je souhaite effectuer de façon ludique mes achats d'articles depuis une vidéo commerciale intégrant des publicités me permettant de remplir facilement mon panier
Scénario S11-1	Un utilisateur effectue des achats via une vidéo
GWT	***GIVEN :*** je visualise une vidéo commerciale qui contient de la publicité sur des articles représentés par des zones cliquables ***WHEN*** : j'active une zone cliquable de l'article ***THEN :*** une fenêtre s'ouvre directement sur le site commercial avec l'article à ajouter dans le panier ***AND :*** je peux finaliser ma commande ou poursuivre la lecture de la vidéo ***AND :*** je reviens automatiquement sur ma vidéo qui reprends sa lecture

Tableau 16 : BR11 - Scénario(s) de test et critères d'acceptation

Fonctionnalité	BR12 : Vidéos au format vertical et horizontal (« multiformat »)
User Story	En tant qu'utilisateur, je souhaite pouvoir visualiser les vidéos avec des formats adaptés à chaque mode d'affichage (portait ou paysage)
Scénario S12-1	Un utilisateur visualise une vidéo compatible « multiformat » en mode paysage
GWT	***GIVEN :*** je visualise une vidéo multiformat en mode paysage. ***WHEN*** : La vidéo se joue ***THEN :*** La vidéo est adaptée au format de l'écran (ex : les miniatures des présentateurs s'affichent sur le côté).
Scénario S12-2	Un utilisateur visualise une vidéo compatible « multiformat » en mode portrait
GWT	***GIVEN :*** je visualise une vidéo multiformat en mode portrait. ***WHEN*** : La vidéo se joue ***THEN :*** La vidéo est adaptée au format de l'écran (ex : les miniatures des présentateurs s'affichent sur le dessus).

Tableau 17 : BR12 – Scénario(s) de test et critères d'acceptation

Tests des exigences non fonctionnelles

Id.	Critère d'acceptation	Mesure de la conformité
TR-1	La plateforme supporte > 500.000 utilisateurs sur lors de la lecture de flux vidéo HD.	Tests de charge automatisés via Apache JMeter.
TR-2	Temps de chargement des vidéos <= 3sec.	Tests de performance automatisés via Apache JMeter.
TR-3	La plateforme doit être capable de supporter des vidéos en flux 4K.	Test unitaires automatisés via JUnit
TR-4	La plateforme peut être en partie interrompue sans perte du service	Tests d'intégration via JUnit
TR-5	Compatibilité du service avec les principales plateformes	Tests d'intégration / UI automatisés via Selenium
TR-6	Compatibilité du service avec les principaux navigateurs web et mobile	Tests d'intégration / UI automatisés via Selenium
TR-7	Les vidéos sont accessibles selon un système de droits	Test unitaires automatisés via JUnit
TR-8	Conformité de la plateforme au RGPD	Test unitaires automatisés via JUnit

Tableau 18 : Procédure d'acceptation et test des exigences non fonctionnelles

TABLES DES RÉFÉRENCES

Figures

Tableaux