**PLAN DE TEST**



**David EVAN**

**30/04/2022**

**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 | 30/04/2022 |
|  |  |  |  |
|  |  |  |  |

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, les fonctionnalités et la conformité de l’architecture cible sont rappelé avant de définir la méthodologie de test retenue.

Pour chaque tests, les indicateurs qualité et les méthodologies de collecte des données seront précisés.

**Table des matières**

[INTRODUCTION 4](#_Toc102137510)

[Rappel du contexte 4](#_Toc102137511)

[Objectifs des tests 4](#_Toc102137512)

[Rôles et responsabilités des parties prenantes 5](#_Toc102137513)

[FONCTIONNALITÉS A TESTER 6](#_Toc102137514)

[Architecture retenue 6](#_Toc102137515)

[Conformité de l’architecture 7](#_Toc102137516)

[PLAN DE TEST 8](#_Toc102137517)

[Périmètre 8](#_Toc102137518)

[Méthodologie 8](#_Toc102137519)

[Approche « Business Driven Development » 8](#_Toc102137520)

[Caractéristiques de l’approche « Busines Driven Development » 8](#_Toc102137521)

[Cycle itératif de développement du produit 9](#_Toc102137522)

[Discovery : « What it could » 9](#_Toc102137523)

[Formulation : « What it should do » 10](#_Toc102137524)

[Automation : « What it really does » 10](#_Toc102137525)

[Implémentation et enchainement du cycle de développement 11](#_Toc102137526)

[Niveaux des tests 12](#_Toc102137527)

[Tests de composants 13](#_Toc102137528)

[Tests d’intégration 13](#_Toc102137529)

[Tests d’interface utilisateur (UI) 13](#_Toc102137530)

[Les tests (manuel) d’acceptation 13](#_Toc102137531)

[Environnement et outillage 14](#_Toc102137532)

[Outillage 14](#_Toc102137533)

[Environnement 14](#_Toc102137534)

[Livrables 15](#_Toc102137535)

[Tests des composants 16](#_Toc102137536)

[TABLES DES RÉFÉRENCES 17](#_Toc102137537)

[Figures 17](#_Toc102137538)

[Tableaux 17](#_Toc102137539)

# 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** défini 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 détaillé dans ces livrables.

## Objectifs des tests

L’objectif des tests est d’assurer la conformité et 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 (tel 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 plan 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, lien réseaux, compte de service …) * Fourni 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. * 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 A TESTER

## Architecture retenue

L’architecture décrite dans le Document de Définition d’Architecture permet de satisfaire aux exigences fonctionnelles telle que décrite 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 fonctionnelle (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 media 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’architecturer et 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 parti du périmètre de ce plan de test.

## Méthodologie

### Approche « Business Driven Development »

Le plan de test retenue 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/*](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 implémentée 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*)

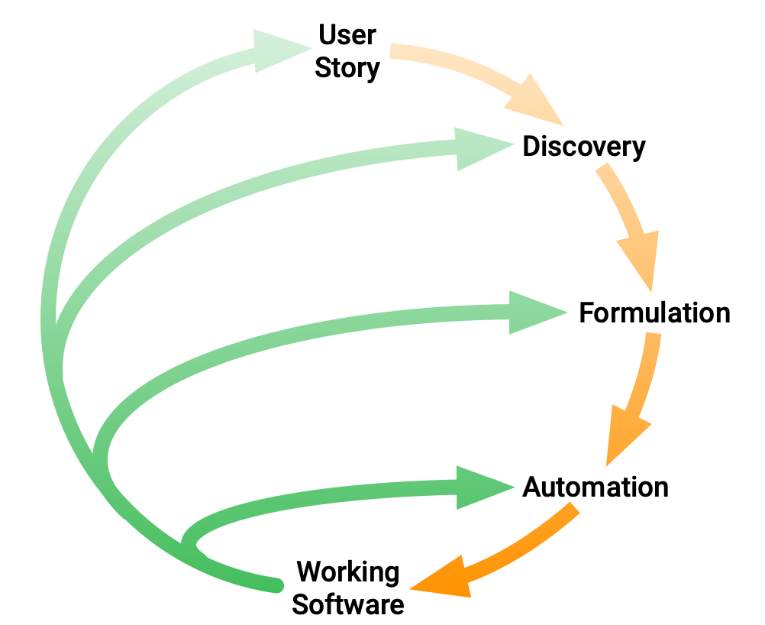


Figure 1 : Discovery, Formulation, and Automation dans l'activité de BDD. (Source: cucumber.oi)

##### 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. Un système lisible à la fois pour les utilisateurs et les système est utilisé. 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 mis en œuvre et mesurés 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 enchainement 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.

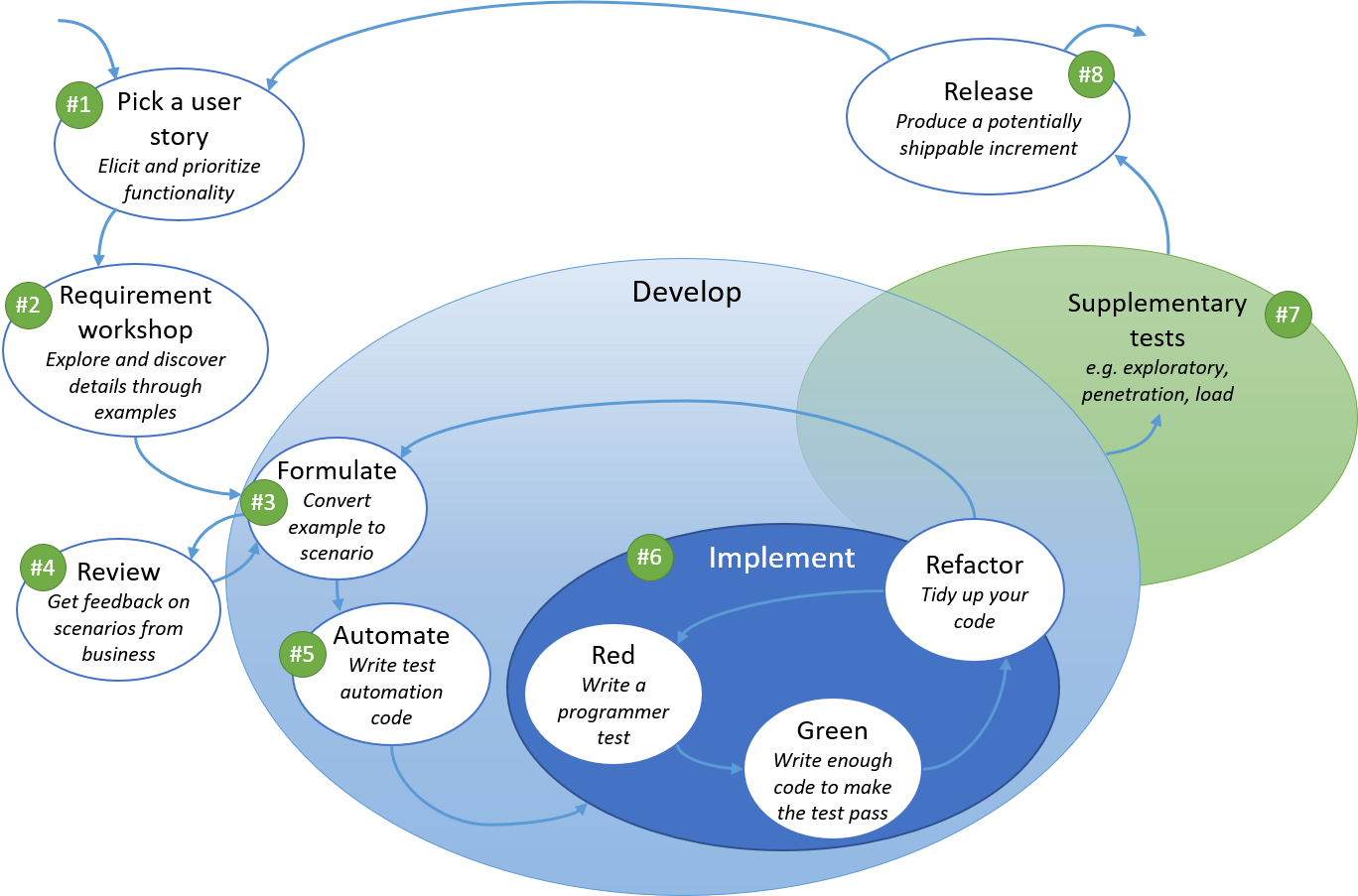


Figure 2 : Implémentation du cycle de développement itératif (source : Medium.com/@hgsgtk)

### 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 présent et exécuté à chaque itération du produit.

La quantité de test généralement attendus en fonction de leur nature peut être représenté sous la forme d’une pyramide.

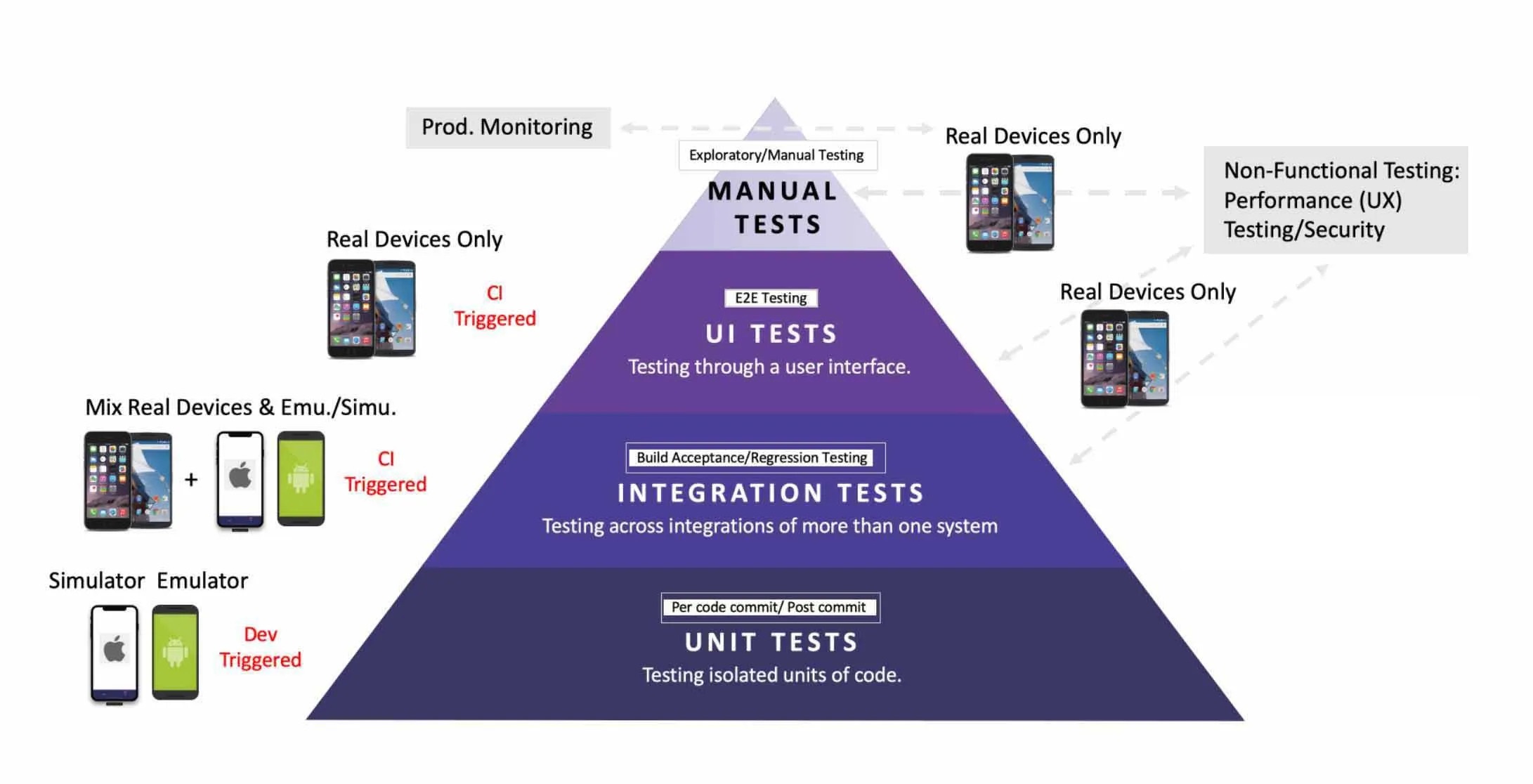


Figure 3 : ISTQB - Testing pyramide (Source : perfecto.io)

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 lancé manuellement par les équipes de testeur pour s’assurer de la conformité du produits 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 système 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 spécifiés. 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-il bien présent ? 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 décrit en GWT (*Given - When - Then*) permettent de facilement automatiser ces points en définissant les critères d’acceptation et les résultats attendus après chaque action.

#### Les tests (manuel) d’acceptation

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).

### Environnement et outillage

#### Outillage

|  |  |  |
| --- | --- | --- |
| **Catégorie** | **Outils** | **Commentaire** |
| **Spécification des tests** | Cucumber |  |
| **Tests unitaires / Test d’intégration** | Framework de test unitaire  (JUnit / PHPUnit / MSTest …)  + Implémentation Cucumber (annotations) |  |
| **Automatisation :**  **Tests unitaires / Tests d’intégration** | GitLab (Pipelines CI) |  |
| **Tests UI** | Selenium Webdriver |  |
| **Automatisation :**  **Tests UI** | JMeter |  |
| **Tests de charge** | JMeter |  |
| **Reporting / Analyse qualité** | Sonar Qube (Intégré aux Pipeline CI GitLab) |  |
| **Suivi des bugs et des incidents** | Jira Software |  |
| **Campagne de test manuelles** | Squash TM |  |

Description des environnements des outils à utiliser Cumcumber, Selenium, JMeter + base de données. Sonar qube + Pipeline CI/CD

#### Environnement

### Livrables

Il existe différents livrables concernant les tests et qui seront fourni 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 composants

Les tests au niveau compo

# TABLES DES RÉFÉRENCES

## Figures

[Figure 1 : Discovery, Formulation, and Automation dans l'activité de BDD. (Source: cucumber.oi) 9](#_Toc102124112)

[Figure 2 : Implémentation du cycle de développement itératif (source : Medium.com/@hgsgtk) 11](#_Toc102124113)

## Tableaux

[Tableau 1 - Historique des révisions 2](#_Toc102124115)

[Tableau 2 : Définition des rôles et responsabilités des parties prenantes 5](#_Toc102124116)

[Tableau 3 : Matrice de mise en relation des fonctionnalités et des composants d'architecture 6](#_Toc102124117)