

J. Racaud; A. Simon; J. Harrault; J. Blondeel; S. Daguenet; F. Corradin

Music Sheet Writer

Music Sheet Writer

Stratégie de Tests

Objectifs du document

Résumé

Glossaire

***– R –***

Release : Mise à disposition d’une version de l’application. Elle peut être privée, semi publique ou publique. En général une release fait état de la version commerciale (ou plus détaillée pour des releases fermées).

Description du document

|  |  |
| --- | --- |
| Titre | Music Sheet Writer : Stratégie de Tests |
| Date de création | 22/10/2015 |
| Date de publication | 15/11/2015 |
| Auteur | J. Racaud; A. Simon; J. Harrault; J. Blondeel; S. Daguenet; F. Corradin |
| Responsable | Jonathan Racaud |
| E-mail | musicsheetwriter\_2017@labeip.epitech.eu |
| Sujet | Stratégie de Tests |
| Version du modèle | 1.0 |

Tableau des révisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Auteur** | **Section(s)** | **Commentaire** |
| 22/10/2015 | Jeremy HARRAULT | Toutes | Création du document |

Sommaire

[1. Introduction 1](#_Toc433295550)

[1.1. Objectif de l’EIP 1](#_Toc433295551)

[1.2. Principe de base du système 1](#_Toc433295552)

[1.3. Documents de référence 1](#_Toc433295553)

[2. Version/release 2](#_Toc433295554)

[3. Types de tests et périmètre 3](#_Toc433295555)

[4. Livrables 4](#_Toc433295556)

[4.1. Cas de tests 4](#_Toc433295557)

[4.2. Rapport d’exécution de tests 4](#_Toc433295558)

[4.3. Rapport sur le suivi global de la qualité 4](#_Toc433295559)

[5. Types de tests 5](#_Toc433295560)

[5.1. Tests XXX-1 5](#_Toc433295561)

[5.2. Tests XXX-2 5](#_Toc433295562)

[6. Estimation Globales de la charge de travail 6](#_Toc433295563)

[7. Outils 7](#_Toc433295564)

[7.1. Dépôt de tests 7](#_Toc433295565)

[7.2. Gestion de rejets/erreurs 7](#_Toc433295566)

[7.3. Autres outils 7](#_Toc433295567)

# Introduction

## Objectif de l’EIP

## Principe de base du système

## Documents de référence

# Version/release

# Types de tests et périmètre

# Livrables

## Cas de tests

## Rapport d’exécution de tests

## Rapport sur le suivi global de la qualité

# Types de tests

## Logiciel

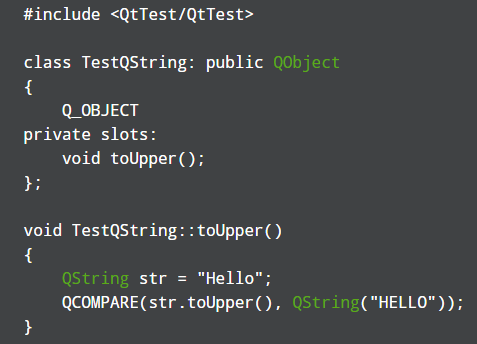
### Tests Unitaire

#### Objectif

Il s'agit pour le programmeur de tester une fonction, indépendamment du reste du programme, ceci afin de s'assurer qu'elle répond aux [spécifications fonctionnelles](https://fr.wikipedia.org/wiki/Sp%C3%A9cification_(informatique)) et qu'elle fonctionne correctement en toutes circonstances.

#### Environnement et conditions de réalisation

Les tests unitaires peuvent être joués à n’importe quel moment de développement et cela sur toute nouvelle fonction implémentée au sein du projet. Ces tests seront réalisés à l’aide de QT (sous version Windows et MAC) qui possède une librairie spécialisée pour les tests, QTestLib.

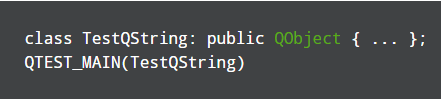


Exemple de la classe de test

Lorsqu’on souhaite réaliser un test, il faut écrire une nouvelle classe, qui hérite de QObject. Chaque private slot représente une fonction qu’on souhaite tester. Il suffit ensuite d’implémenter cette fonction et de réaliser les tests à l’aide des macros de la librairie QTestLib.

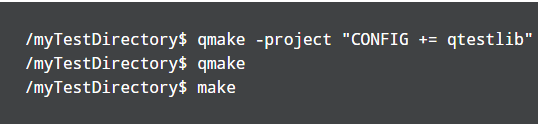
#### Configurations particulières

Comme vu précédemment, lorsqu’on souhaite réaliser un test, il faut écrire une classe qui hérite de QObject. Une fois notre classe créée et les fonctions qu’on souhaite tester sont implémentées, il suffit d’ajouter à la fin du fichier QTEST\_MAIN(NomDeLaClasseTest). Cela va créer un exécutable lors de la compilation, qui une fois lancé va exécuter les tests dans l’ordre dans lequel sont définis les privates slots.



Utilisation de QTEST\_MAIN

Pour la compilation, if faut indiquer à qmake qu’on utilise la librairie QTestLib.



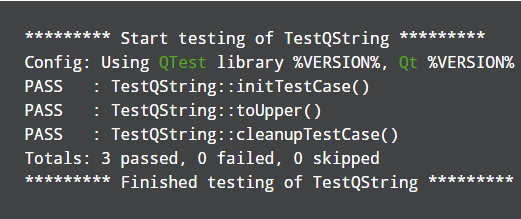
Compilation avec la librairie QTestLib

#### Planning et charge

Ces tests sont pris en compte, sur le Gantt, dans la période de temps consacré au développement d’une fonctionnalité du logiciel. La charge de travail dépend de ce qui est testé.

#### Critères de passage/échec

Les critères de passage/échec seront définis via les tests effectués. Si une erreur est détectée, celle-ci est retournée et le test s’arrête.



Sortie d’exécution du programme de test

initTestCase() et cleanupTestCase() sont automatiquement appelés, le premier permet d’initialiser la fonction qui va être testée et la seconde permet de libérer la place pour la prochaine fonction à test.

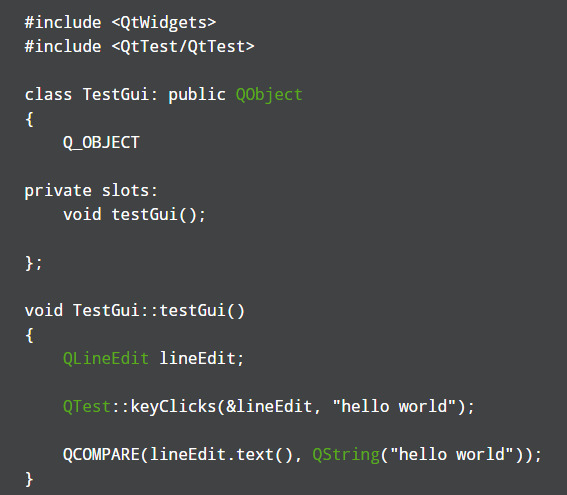
### Tests logiciel UI

#### Objectif

Ces tests ont pour but de vérifier le bon déroulement des interactions entre l’utilisateur et l’interface utilisateur du logiciel.

#### Environnement et conditions de réalisation

Ces tests interviennent dès qu’on ajoute un élément, tel qu’un bouton ou un champ texte, à l’interface utilisateur. Comme pour les tests unitaires, on se sert la librairie QTestLib qui permet notamment de simuler les actions utilisateurs.



Exemple de test sur un champ texte

Dans la fonction testGui, on simule le clic sur un champ texte dans lequel on écrit ‘’hello world’’ et on vérifie ensuite que le contenu de ce champ texte est bien ‘’hello world’’.

#### Configurations particulières

Voir partie 5.1.1.3.

#### Planning et charge

Ces tests sont pris en compte, sur le Gantt, dans la période de temps consacrée au développement d’une fonctionnalité du logiciel. La charge de travail dépend de ce qui est testé.

#### Critère de démarrage des tests

Le logiciel doit être installé.

#### Critères de passage/échec

Voir partie 5.1.1.6.

### Tests Fonctionnels

#### Objectif

C’est un ensemble de tests unitaires ayant pour but de valider le bon déroulement d’une fonctionnalité du logiciel, comme par exemple la création d’un projet ou l’édition d’une partition.

#### Environnement et conditions de réalisation

Un test fonctionnel étant un ensemble de test unitaire, se référer à la partie 5.1.1.2

#### Configurations particulières

Aucune configuration particulière

#### Planning et charge

Ces tests sont pris en compte, sur le Gantt, dans la période de temps consacrée au développement d’une fonctionnalité du logiciel. La charge de travail dépend de ce qui est testé.

#### Critères de passage/échec

Pour que le test soit considéré comme un succès, l’ensemble des tests unitaires qui compose le test de fonctionnalité doivent réussir.

### Tests de non-régression

#### Objectif

Ces tests ont pour objectif de vérifier que les modifications apportées au projet, n’ont pas entraîné d’effets de bord non prévus qui pourraient dégrader le comportement du logiciel antérieurement validé.

#### Environnement et conditions de réalisation

Les tests de non-régression seront appliqué tout au long du développement et plus particulièrement lorsqu’une fonctionnalité, nouvellement implémentée et pas encore testé, interagie avec une partie du logiciel déjà testé.

#### Configurations particulières

Il faut écrire ces tests en réutilisant les tests déjà joués et qui ont réussi afin de s’assurer que le système répond toujours aux exigences spécifiées.

#### Planning et charge

Ces tests sont pris en compte, sur le Gantt, dans la période de temps consacrée au développement d’une fonctionnalité du logiciel. La charge de travail dépend de ce qui est testé.

#### Critères de passage/échec

Pour que le test soit considéré comme un succès, il faut que malgré l’ajout ou la modification de code, ce dernier réponde toujours aux exigences spécifiées.

## Application mobile : Android

### Tests Unitaire

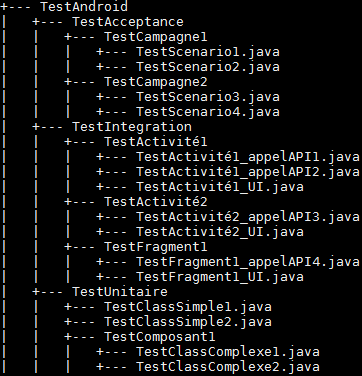
#### Objectif

L’objectif de ces tests est de vérifier le bon fonctionnement de certaines classes ou méthodes particulières appelées au sein de l’application Android. Ces tests concernent exclusivement les fonctions n’utilisant pas de contexte spécifique à l’application Android MusicSheetWriter pour fonctionner. C’est-à-dire qu’ils n’utilisent aucun élément visuel. Ces tests sont la plus basse couche de test à réaliser sur l’application Android et ne nécessitent pas de contexte Android pour fonctionner.

#### Environnement et conditions de réalisation

Ces tests pourront être joués à n’importe quel moment du développement. Le test sera réalisé aussitôt qu’une classe ou une fonctionnalité n’agissant pas sur l’interface graphique est spécifiée. Il sera sous la forme d’une ou plusieurs classes java représentant les suites de tests, chacune comportant des méthodes représentant les cas de tests à exécuter.

Si la suite de tests vérifie une simple classe (Validateur d’email, validateur de mot de passe, téléchargeur d’image) elle pourra être codée en une classe. En revanche, si la suite de tests vérifie fonctionnalité comportant un ensemble de classes (expéditeur de requête HTTP), il est préférable de la diviser en plusieurs suites de test pertinentes pour les regrouper dans un même dossier.

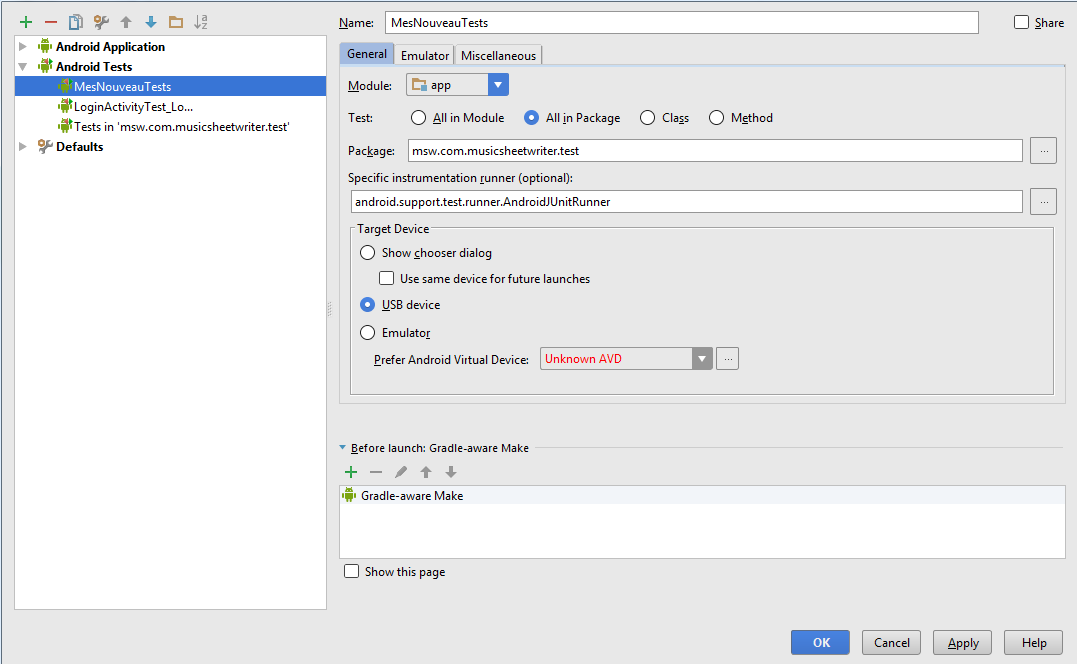


Exemple d'arborescence pour les tests unitaires d’Android

#### Configurations particulières

Seul un périphérique capable de lancer l’application à tester est nécessaire.

En outre, Android Studio facilite grandement la manipulation car il gère lui-même le lancement de l’application et des tests. Il suffit d’ajouter une nouvelle configuration de lancement de test. Afin de créer une nouvelle configuration de lancement de tests, il faut assigner un « Runner », sélectionner les tests à lancer et définir un périphérique cible.



Définition d'une configuration de lancement de tests sur Android Studio

#### Planning et charge

Les tests unitaires peuvent être écrits à la suite de la spécification de la ou des classes qu’ils testent. La charge de travail dépend de ce qui est testé.

#### Critère de démarrage des tests

L’application doit être installée sur le périphérique cible.

#### Critères de passage/échec

Les critères de passage ou d’échec sont déterminés par les assertions dans le code des tests.

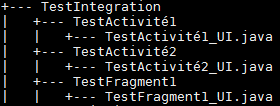
### Test Android UI

#### Objectif

L’objectif de ces tests est de vérifier le bon fonctionnement de la capture des événements graphiques et les conséquences qu’elles engendrent sur l’interface utilisateur. Ces tests concernent les classes chargées de l’affichage graphique, c’est-à-dire les activités et les fragments. De telles classes seront testées de manière unitaire. En effet, chaque activité ou fragment aura une ou plusieurs classes de test associées.

#### Environnement et conditions de réalisation

Ces tests pourront être joués à n’importe quel moment du développement. Le test sera réalisé aussitôt que l’interface et les événements graphiques d’une activité ou d’un fragment sont spécifiés. Il sera sous la forme d’une ou plusieurs classes java représentant les suites de tests, chacune comportant des méthodes représentant les cas de tests à exécuter.



Exemple d'arborescence les tests d’intégration UI d’Android

#### Configurations particulières

Voir la partie 5.2.1.3

#### Planning et charge

Les tests UI peuvent être écrits à la suite de la spécification de la ou des classes qu’ils testent. La charge de travail dépend de ce qui est testé.

#### Critère de démarrage des tests

L’application doit être installée sur le périphérique cible.

#### Critères de passage/échec

Les critères de passage ou d’échec sont déterminés par les assertions dans le code des tests.

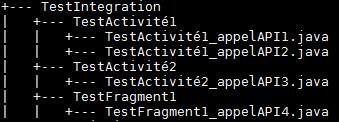
### Test d’appel à l’API

#### Objectif

L’objectif de ces tests est de vérifier l’envoi des différentes requêtes à l’API suite aux actions utilisateur ainsi que l’impact des réponses sur l’interface graphique. Ces tests concernent les classes chargées de l’affichage graphique, c’est-à-dire les activités et les fragments, et pouvant envoyer des requêtes à l’API. De telles classes seront testées de manière unitaire. En effet, chaque activité ou fragment aura une ou plusieurs classes de test associées.

#### Environnement et conditions de réalisation

Ces tests pourront être joués à n’importe quel moment du développement. Le test sera réalisé aussitôt que les requêtes à envoyer et que leur impact sur l’interface graphique sont spécifiés pour une activité ou un fragment. Il sera sous la forme d’une ou plusieurs classes java représentant les suites de tests, chacune comportant des méthodes représentant les cas de tests à exécuter. Pour de tels tests, chaque test suite vérifiera une URL d’API pour une seule activité ou un seul fragment.



Exemple d'arborescence les tests d’intégration Appel API d’Android

Pour chaque requête, les tests devront comprendre les cas où l’API renvoie une réponse valide et les cas où l’API renvoie une erreur.

#### Configurations particulières

Voir la partie

#### Planning et charge

Les tests d’appel à l’API peuvent être écrits à la suite de la spécification de la ou des classes qu’ils testent. La charge de travail dépend de ce qui est testé.

#### Critère de démarrage des tests

L’application doit être installée sur le périphérique cible et l’API doit être fonctionnelle.

#### Critères de passage/échec

Les critères de passage ou d’échec sont déterminés par les assertions dans le code des tests.

## Application mobile : IPhone

### Tests Unitaire

#### Objectif

L’objectif de ce test est très simple, il doit simplement vérifier que chaque méthode de la classe API se lance correctement et s’exécute. Si la méthode retourne une erreur cela n’est pas grave, cela veut dire malgré tout qu’il y a bien une communication avec l’API et que la méthode est donc fonctionnelle. De plus il y a des tests sur la globalité du code, comme un test de connexion réussi, un test de connexion raté, le retour de certaines fonctions, etc.

#### Environnement et conditions de réalisation

Ce test peut être réalisé dans n’importe quel cas ou contexte. Il est bien de le lancer quand l’API est modifiée ou que l’on doute de son bon fonctionnement. Il ne vérifie en rien l’interface de l’application.

#### Configurations particulières

Aucune configuration particulière n’est requise pour ce test.

#### Planning et charge

Les tests unitaires peuvent être écrits à la suite de la spécification de la ou des classes qu’ils testent. La charge de travail dépend de ce qui est testé.

#### Critères de passage/échec

Xcode indiquera le passage ou non du test grâce à une coche verte ou une croix rouge. Il faut passer tous les tests pour valider cette partie. Il faut donc qu’aucune méthode ne retourne *NULL* ou ne vienne à répondre incorrectement.

### Tests IPhone UI

#### Objectif

L’objectif de ce test est de vérifier la partie interface de l’application. Certaines parties n’étant pas détectables via le code, la création de *mocks* fut donc la meilleure solution. Ces tests serviront à mettre en évidence les problèmes lié à l’interface mais par la même occasion de tester à nouveau une partie des fonctionnalités de l’application.

#### Environnement et conditions de réalisation

Les tests seront effectués à des moments clés de chaque étape. Dès qu’une nouvelle fonctionnalité est produite elle se doit d’être testée. Par exemple si le login vient d’être implémentée, il doit être testé grâce à une *Mock* (mais aussi par un test unitaire).

Tous les tests se feront via *Xcode* qui contient une interface claire de test et seront aussi lancés grâce à *xcodebuilder* pour permettre un export des tests.

#### Configurations particulières

Pour réaliser ce test il faudra avoir une interface terminée. De plus Il faudra aucun abonnement à part « bob » pour le compte « bob » et le mot de passe de ce dernier devra être « b ». Si cela venez à changer le test actuel ne pourrait être exécuté et une partie du code sera à changer.

#### Planning et charge

Les tests unitaires peuvent être écrits à la suite de la spécification de la ou des classes qu’ils testent. La charge de travail dépend de ce qui est testé.

#### Critère de démarrage des tests

Pour démarrer ce test l’application doit être fermée et compiler sur le téléphone ou simulateur.

#### Critères de passage/échec

Il y a plusieurs critères de passage pour chaque partie de ce test. IL faut que le temps d’exécution soit inférieur à 10 secondes pour chaque simulation de l’utilisateur et l’exécution des commandes faites dans le code doivent correspondre à ce qui s’affiche.

## Application mobile : Windows Phone

### Tests Unitaire

#### Objectif

Les tests unitaires sont utilisés de manière générale afin de permettre le bon déroulement du développement de l’application. Ils permettront de tester individuellement les composants de l’application. On pourra ainsi valider la qualité du code et les performances d'un module.

#### Environnement et conditions de réalisation

Ces tests pourront être joués à n’importe quel moment du développement. Le test sera réalisé aussitôt qu’une classe ou une fonctionnalité n’agissant pas sur l’interface graphique est spécifiée. Il sera sous la forme d’une ou plusieurs classes C# représentant les suites de tests, chacune comportant des méthodes représentant les cas de tests à exécuter.

Si la suite de tests vérifie une simple classe (Validateur d’email, validateur de mot de passe, téléchargeur d’image) elle pourra être codée en une classe. En revanche, si la suite de test vérifie fonctionnalité comportant un ensemble de classes (expéditeur de requête HTTP), il est préférable de la diviser en plusieurs suites de test pertinentes pour les regrouper dans un même dossier.

#### Configurations particulières

Seul un périphérique capable de lancer l’application à tester est nécessaire.

#### Planning et charge

Les tests seront effectués régulièrement afin de valider l’avancée du projet et le code préalablement effectué. Cela permettra également de valider la qualité du code et les performances d'un module.

#### Critère de démarrage des tests

L’application doit être installée sur le périphérique cible.

#### Critères de passage/échec

Les critères de passage/échec seront définis via les tests effectués. En effet, si la réponse retour est la même que la réponse attendu le test sera passé sinon ce sera un échec.

### Tests Windows Phone UI

#### Objectif

L’objectif de ces tests est de vérifier le bon fonctionnement de la capture des événements graphiques et les conséquences qu’elles engendrent sur l’interface utilisateur. Ces tests concernent les classes chargées de l’affichage graphique, c’est-à-dire les activités et les fragments. De telles classes seront testées de manière unitaire. En effet, chaque activité ou fragment aura une ou plusieurs classes de test associées.

#### Environnement et conditions de réalisation

Ces tests pourront être joués à n’importe quel moment du développement. Le test sera réalisé aussitôt que l’interface et les événements graphiques d’une activité. Il sera sous la forme d’une ou plusieurs classes C# représentant les suites de tests, chacune comportant des méthodes représentant les cas de tests à exécuter.

#### Configurations particulières

Seul un périphérique capable de lancer l’application à tester est nécessaire.

#### Planning et charge

Les tests seront effectués régulièrement afin de valider l’avancée du projet et le code préalablement effectué. Cela permettra également de valider la qualité du code et les performances d'un module.

#### Critère de démarrage des tests

L’application doit être installée sur le périphérique cible.

#### Critères de passage/échec

Les critères de passage/échec seront définis via les tests effectués. En effet, si la réponse retour est la même que la réponse attendu le test sera passé sinon ce sera un échec.

### Tests fonctionnels

#### Objectif

Les tests fonctionnels sont utilisés de manière générale afin de permettre le bon déroulement du développement de l’application. Ces tests ont pour but de vérifier la conformité de l'application développée avec le cahier des charges initial. Ils sont donc basés sur les spécifications fonctionnelles et techniques. Les tests fonctionnels auront une couverture complète de l’application.

#### Environnement et conditions de réalisation

Les tests fonctionnels seront utilisés tout au long du projet et sur toutes les releases. Les tests seront effectués sous Microsoft Visual Studio via la création d’un projet de test unitaire.

#### Configurations particulières

Seul un périphérique capable de lancer l’application à tester est nécessaire.

#### Planning et charge

Les tests seront effectués régulièrement afin de valider l’avancée du projet et le code préalablement effectué. Cela permettra également de vérifier la conformité de l'application développée avec le cahier des charges initial. Ils sont donc basés sur les spécifications fonctionnelles et techniques.

#### Critère de démarrage des tests

L’application doit être installée sur le périphérique cible.

#### Critères de passage/échec

Les critères de passage/échec seront définis via les tests effectués. En effet, si la réponse retour est la même que la réponse attendu le test sera passé sinon ce sera un échec.

### Tests fonctionnels

#### Objectif

Les tests de non-régression sont utilisés de manière générale afin de permettre le bon déroulement du développement de l’application. Les tests de non-régression permettent de vérifier que des modifications n'ont pas altéré le fonctionnent de l'application. Les tests de non-régression auront une couverture complète de l’application.

#### Environnement et conditions de réalisation

Les tests de non-régression seront utilisés tout au long du projet et sur toutes les releases. Les tests seront effectués sous Microsoft Visual Studio via la création d’un projet de test unitaire.

#### Configurations particulières

Seul un périphérique capable de lancer l’application à tester est nécessaire.

#### Planning et charge

Les tests seront effectués régulièrement afin de valider l’avancée du projet et le code préalablement effectué. Cela permettra également de vérifier que des modifications n'ont pas altéré le fonctionnent de l'application.

#### Critère de démarrage des tests

L’application doit être installée sur le périphérique cible.

#### Critères de passage/échec

Les critères de passage/échec seront définis via les tests effectués. En effet, si la réponse retour est la même que la réponse attendu le test sera passé sinon ce sera un échec.

## Site internet

### Tests Unitaires

#### Objectif

Le but de ces tests va être de comparer la valeur de retour d’une fonction avec la valeur attendue. Cela permet de retrouver facilement le moment ou le programme ne fonctionne pas comme prévu et de trouver le problème rapidement.

#### Environnement et conditions de réalisation

Les tests unitaires pour le site web seront exécutés sur la partie interface utilisateur et sur l’API. Avant la mise en production les tests seront effectués premièrement un environnement de développement puis une fois déposé sur le SVN ils seront testés en environnement de production pour vérifier que le comportement soit bien le même.

En environnement de développement les serveurs de tests ne seront pas configurés dans des conditions optimales les serveurs n’ont pas besoin de configuration de sécurité ou de performance particulières la machine hébergeant l’application est moins performante. Contrairement à l’environnement de production.

#### Planning et charge

Les tests unitaires d’une partie du site devront être effectués avant les dates limites de la réalisation de la partie prévue dans le gant.

#### Critère de démarrage des tests

Pour démarrer les tests le serveur web, de base de données et de messagerie doivent être fonctionnels et bien configuré.

#### Critères de passage/échec

Un test unitaire est considéré comme un échec du moment où les deux valeurs de comparaison sont différentes.

### Tests Fonctionnels

#### Objectif

Le but des tests fonctionnels est de vérifier qu’une certaine fonctionnalité de l’application fonctionne parfaitement. Pour cela un test fonctionnel est composé de plusieurs tests unitaires qui vont vérifier l’ensemble des critères nécessaires pour juger qu’une fonctionnalité de l’application marche correctement.

#### Environnement et conditions de réalisation

Avant la mise en production les tests seront effectués premièrement un environnement de développement puis une fois déposé sur le SVN ils seront testés en environnement de production pour vérifier que le comportement soit bien le même.

#### Planning et charge

Les tests fonctionnels d’une partie du site devront être effectués avant les dates limites de la réalisation de la partie prévue dans le gant.

#### Critère de démarrage des tests

Pour démarrer les tests le serveur web, de base de données et de messagerie doivent être fonctionnels et bien configuré.

#### Critères de passage/échec

Un test fonctionnel est considéré comme un échec lorsqu’au moins un des tests unitaires inclus dans le composant a échoué.

# Estimation Globales de la charge de travail

# Outils

## Dépôt de tests

## Gestion de rejets/erreurs

## Autres outils

### Outils d’automatisation des tests

### Outils pour les tests de performance

### Outils pour/de XXX

### Outils pour/de XXX