

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

***– A –***

Activité (Android) : Composant applicatif fournissant un écran avec lequel les utilisateurs peuvent interagir afin d’effectuer une action. Une application est composée d’une ou plusieurs activités.

***– B –***

Bug (ou « bogue ») : Défaut de conception ou d’implémentation d’un programme informatique étant à l’origine d’un disfonctionnement.

***– F –***

Fragment (Android) : Composant d’une activité représentant un comportement ou une portion de l’interface utilisateur. Une activité peut être composée d’une ou plusieurs fragmentS.

***– 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).

***– U –***

UI (User Interface) : Dispositif de de dialogue entre un programme informatique et un utilisateur. Il permet à ce dernier d’effectuer des manipulations au sein du programme informatique et de présenter le résultat de ces manipulations.

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 |
| 04/11/2015 | Jeremy HARRAULT | 2. Version/release  3. Types de tests et périmètre | Ajout de la partie Android. |

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

### Android

#### Plan de release

Voici la liste des releases majeurs de l’application Android. Leurs numéros de version seront notés sous la forme X.0 où X est le numéro de la release majeur. Ce nombre sera incrémenté de 1 à chaque release.

D’autres releases mineures pourront être faites si :

* Une fonctionnalité rapide à implémenter est fortement demandée par la communauté et/ou devient urgent à implémenter pour satisfaire l’expérience MusicSheetWriter.
* Des « bugs » sont trouvés sur l’application, soit par l’équipe de développement, soit pas la communauté. Ces bugs pourront être remontés aux développeurs via la fonctionnalité de report de bug intégré de base au système Android.

Une release mineure sera notée sous la forme X.Y où X est le numéro de la release majeure en cours et Y est le numéro de la release mineure.

#### Plan des release majeures

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro de version | Date de sortie prévue | Description | Fonctionnalités ajoutées |
| 1.0 | 4/08/2016 | Cette version sera la première version de l’application Android. Elle sortira en même temps que la première version du logiciel. Aucune version n’est prévus avant car la gestion des partitions ne sera pas utilisé sans le logiciel et l’aspect communautaire ne sera pas assez poussez pour se suffire à lui seul. | * Création et connexion à un compte utilisateur * Gestion des informations personnelles * Consultation des partitions de l’utilisateur * Gestion des partitions favorites de l’utilisateur * Gestion des abonnements de l’utilisateur * Recherche d’utilisateur * Recherche de partition * Lecture sonore d’une partition |
| 2.0 | 30/03/2017 | Cette version sera tournée vers les fonctionnalités communautaires et commencera son approche vers l’implémentation des fonctionnalités d’édition de partitions. | * Correction de bugs * Commenter les partitions * Envoie de message à d’autre utilisateur * Lecture d’une partition * Téléchargement d’une partition depuis le serveur sous format PDF |
| 3.0 | 30/09/2017 | Cette version rapprochera les fonctionnalités de lecture et d’édition de partition du logiciel et de l’application Android. | * Correction de bugs * Création d’un projet MusicSheetWriter * Edition « note-à-note » d’une partition * Téléchargement d’une partition vers et depuis le serveur sous forme de projet MusicSheetWriter |

# Types de tests et périmètre

## Android

### Test unitaire

Ces tests vérifieront le bon fonctionnement de classes ou méthode particulière. Ils seront effectués hors d’un contexte spécifique à l’application Android MusicSheetWriter. En effet, ils ne nécessitent pas d’élément visuel particulier. Ce type de test sera notamment pour les classes et méthodes utilitaires et susceptible d’être réutilisé dans d’autres projets comme les classes d’appel HTTP, les classes de validation de formulaire ou encore d’authentification.

### Test d’intégration

Ces tests vérifient le bon fonctionnement des classes et méthode couplé avec l’Interface utilisateur de l’application. Les activités et fragments concernés par ces tests seront testés indépendamment les uns des autres. Ces tests seront de deux types différents :

#### Test composant UI

Les tests UI vérifieront le bon fonctionnement des interactions entre l’application et l’utilisateur (l’appui d’un bouton ou le défilement d’une page, etc.) et de leurs conséquences (affichage d’un texte, déclenchement d’une animation, etc.). Ils vérifieront également la gestion des périphériques comme l’appareil photo. Chaque activité et fragment ayant un contenu visuel sera soumis à des tests UI.

#### Test d’appel à API

Les tests d’appel à API vérifieront le bon fonctionnement des actions nécessitant ou incluant l’envoie de requête à l’API MusicSheetWriter suite à une action utilisateur particulière ainsi que les conséquences sur l’interface utilisateurs de chacune des réponses possible renvoyée par le serveur. Chaque activité et fragment contenant un appel à l’API sera soumis à des tests d’appel à l’API.

### Test de non-régression

Ces tests seront établis à partir des tests unitaires et d’intégration et permettront de vérifier l’intégration d’une nouvelle partie du code ou d’une modification du code existant en s’assurant que les fonctionnalités de l’application sont toujours valides.

### Test d’acceptation

Ces tests vérifieront le bon fonctionnement de certains scénario prédéfinit et couvrant l’ensemble des fonctionnalités de l’application. A la différence des tests des tests d’intégrations, ces tests vérifieront l’application dans son ensemble, et pas les activités et fragments de manière unitaire.

# Livrables

## Cas de tests

### Android

#### L’arborescence

Le livrable des cas de tests pour l’application Android suivra la même arborescence que dans leur implémentation.

Les cas de tests seront regroupés par type de test (Acceptation, Intégration, Unitaire, etc.), campagne de test (Campagne, Activité, Fragment, Composant, etc.), et suite de test (Scenario, UI, appelAPI, Class, etc.). Chaque cas de test étant une méthode de la classe de test.

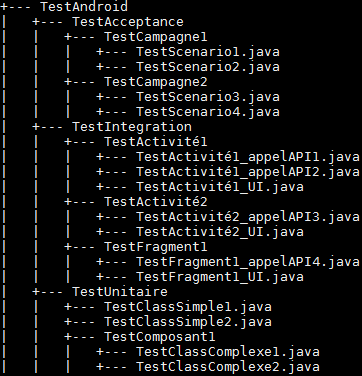


Figure : Exemple d’arborescence des fichiers de tests Android

Cette organisation permettra tout d’abord de comprendre les enjeux et le contexte Android dans lequel sont joués les tests. De plus, il permettra aux développeurs de retrouver facilement le code de chaque test au sein du code source.

#### Tests Unitaire

|  |  |
| --- | --- |
| *Chaque test comportera…* | *Afin de…* |
| Une courte description présentant le test | Connaitre l’élément testé |
| Les données utilisées en temps qu’entré pour le test | Savoir quelle valeur sont testés et par substitution, lesquels ne le sont pas |
| Les conditions à succès | Connaitre quelle et comment est la sortie du test |
| La durée approximative du test | Connaitre le temps nécessaire à son exécution normale. |

#### Test Intégration et d’acceptation

|  |  |
| --- | --- |
| *Chaque test comportera…* | *Afin de…* |
| Une courte description présentant le test | Connaitre la fonctionnalité testé et comprendre le contexte dans lequel se trouve le user s’il devait faire le test manuellement |
| Les données utilisées en temps qu’entré pour le test | Savoir quelles valeurs sont utilisées pour le test |
| Les conditions à succès | Connaitre quelle et comment est la sortie du test |
| Le type de scénario qu’il simule (voir ci-dessous) | Connaitre la nature du test |
| La durée approximative du test | Connaitre le temps nécessaire à son exécution normale. |

Le type de scénario peut être :

* Basic : c’est le flow d’exécution tel qu’il est censé se produire en condition d’utilisation normale
* Alternative : c’est le flow d’exécution tel qu’il est censé se produire en condition d’utilisateur particulière
* Exception : c’est le flow d’exécution tel qu’il est censé se produire en condition d’erreur

Le livrable des cas de tests devra être mis à jour aussitôt qu’une campagne de test est terminé d’être codé, même si tous les tests ne sont pas en succès.

## Rapport d’exécution de tests

Le livrable présentant les cas de tests pour Android sera sous la forme d’un fichier HTML. Il présentera tous les tests joués, regroupé par suite et campagne de tests. Les résultats de ces derniers sera données et, pour les tests ayant échoué, le fil d’exécution du test sera affiché afin de montrer exactement les raisons qui ont amené le test à échouer.



Figure : Résultats des tests Android

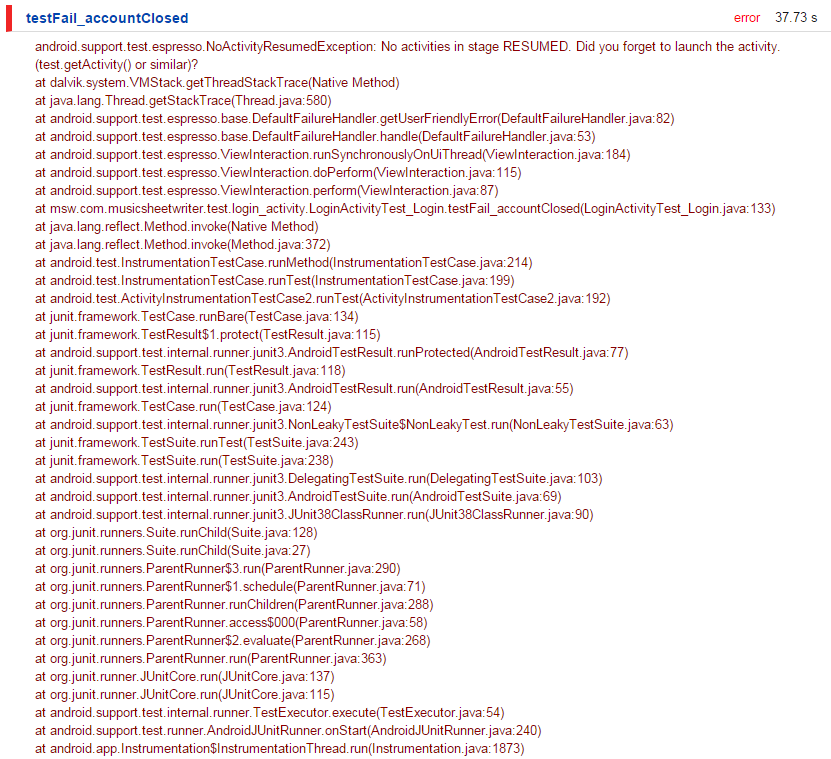


Figure : Fil d'exécution des tests en échec

## Rapport sur le suivi global de la qualité

# Types de tests

## Android

### Test Unitaire

#### Objectif

L’objectif de ces tests et 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 nécessite 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 test, chacune comportant des méthodes représentant les cas de test à exécuter.

Si la suite de test 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.

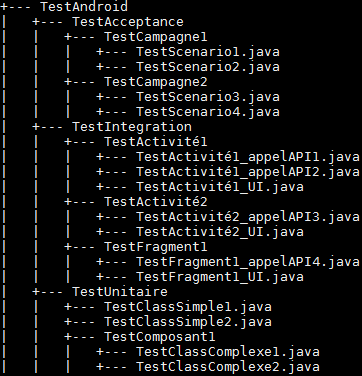


Figure :Exemple d'arborescence pour les tests unitaires d’Android

#### 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.

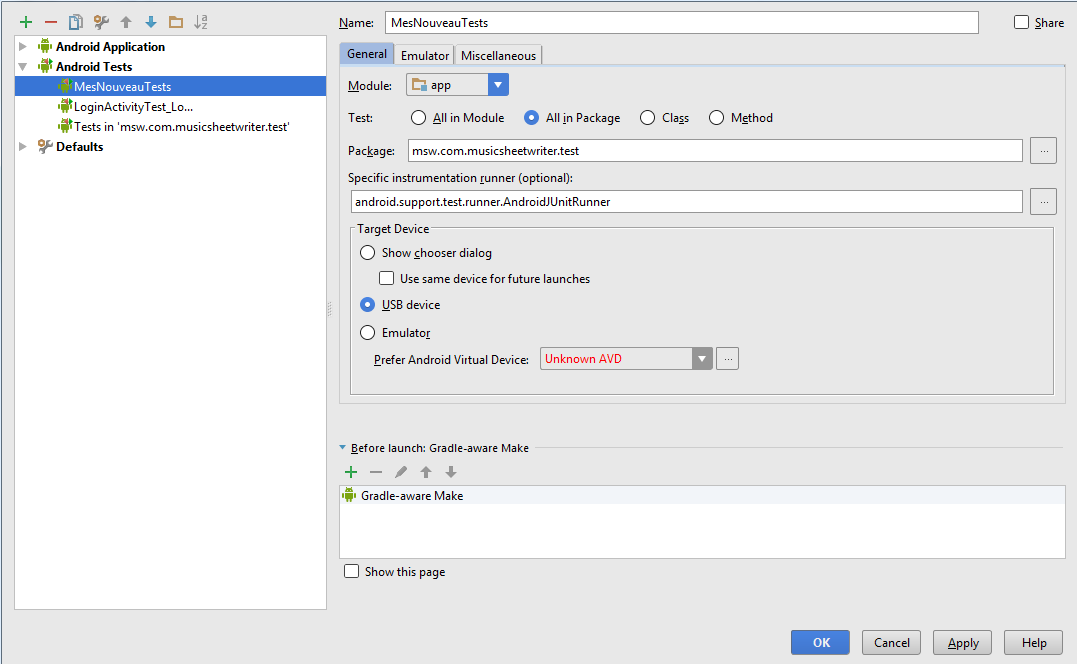


Figure : Définition d'une configuration de lancement de tests sur Android Studio

#### 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 d’Intégration - Test UI

#### Objectif

L’objectif de ces tests et de vérifier le bon fonctionnement de la capture des événements graphiques et les conséquences qu’elles engendrent sur l’interface utilisateurs. 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 test, chacune comportant des méthodes représentant les cas de test à exécuter.

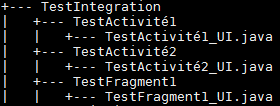


Figure 6 : Exemple d'arborescence les tests d’intégration UI d’Android

#### Configurations particulières

Voir la partie

#### 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’Intégration – Test d’appel à l’API

#### Objectif

L’objectif de ces tests et de vérifier l’envoie 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 requête 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 test, chacune comportant des méthodes représentant les cas de test à exécuter. Pour de tels tests, chaque test suite vérifiera une URL d’API pour une seule activité ou un seul fragment.

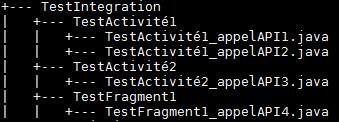


Figure : Exemple d'arborescence les tests d’intégration Appel API d’Android

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.

# Estimation Globales de la charge de travail

# Outils

## Android

### Dépôt de tests

Les tests seront sur un dépôt SVN. Ils seront placés dans les sources du projet Android, dans un dossier « test ».

### Gestion de rejets/erreurs

Aucun outil de gestion des erreurs n’est prévu.

### Autres outils

#### Outils d’automatisation des tests

Pour l’automatisation de test, Android Studio intègre nativement le framework JUnit (<http://junit.org/>). Il gère le lancement automatique des tests.

#### Outils pour les tests d’UI

Le framework Espresso (<https://google.github.io/android-testing-support-library/docs/espresso/index.html>) est utilisé. Il fournit une bibliothèque de fonctions permettant de simuler les actions utilisateur mais également de vérifier l’état des éléments graphiques.