POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

Travail pratique 1

Plan et test, Maintenabilité et Fiabilité

Présenté à :

Alexandre Blasakis

Dans le cadre du cours :

LOG8371 – Ingénierie de la qualité en logiciel

Groupe : 01

Soumis par :

Minh-Tri Do – 2030231
Jeremy Hage – 2014233
Jonathan Siclait – 2026416

Le 14 février 2022

Table des matières

Question 1 : plan d'assurance qualité 3
Description du logiciel 3
Importance de la qualité 3
Parties prenantes 3
Stratégie de validation 6
Question 2 : Stratégie de testing 7
Plan des tests 7
Mesures et méthodes pour valider les objectifs 7
Fonctionalité : 8
Fiabilité : 8
Maintenabilité : 8
Question 3 : Intégration continue 11
Plan d'intégration continue 11
Ajout d'un nouveau module 13
Démonstration de l'intégration continue du système 13
Annexe 14
Tests *App* sur pixel 2 14
Test *App* sur tablette personalisée 20
Tests *Core* 27
Explication des tests 37

Question 1 : plan d'assurance qualité

Description du logiciel

AntennaPod est une application de gestion de podcast fondée en 2011 par Daniel Oeh. Cette application Android, codée en Java, est ouverte au public et est complètement libre de droits. Ce logiciel a été développée dans le but de permettre à n'importe qui de l'utiliser afin d'écouter une multitude de podcasts. De plus, puisque les développeurs ont développé cette application sans but lucratif, il n'y a aucune publicité intégrée dans le système. Elle a présentement été installée sur plus de 180 000 appareils à ce jour. C'est donc sur cette application que nous avons décidé d'effectuer cette analyse. Dans le cadre de ce rapport, nous feront une analyse du plan d'assurance qualité du système qui se penchera sur les modules *app* et *core*. [1]

Importance de la qualité

La qualité est particulièrement importante dans le cas de AntennaPod car ce logiciel est ouvert au public (open source). En effet, le fait que n'importe qui dans le monde peut ajouter des composantes au projet sans qu'il n'y aie une supervision associée rend la présence de standards de codage, de vérification de code et de tests multiples essentiels pour assurer une bonne maintenabilité du code ainsi qu'une application sécuritaire pour les utilisateurs.

Parties prenantes

En analysant le projet AntennaPod, il est facile de remarquer qu'il existe 3 principales parties prenantes. Il existe tout d'abord l'audience. C'est évidemment dans l'intérêt des utilisateurs de valider que l'application fonctionne. Ensuite, il y a les créateurs de podcasts qui placent leurs podcasts dans l'application dans le but de maximiser le nombre de gens qui peuvent l'écouter. Enfin, il existe aussi les développeurs d'AntennaPod qui tiennent à ce que l'application fonctionne car ils ont travaillé sur ce projet à but non lucratif afin de donner la meilleure expérience utilisateur possible.

[1] https://antennapod.org

Couverture de qualité

Critère	Sous-critère	Objectif	Mesure de validation
Fonctionnalité	Complétude : Critère qui décrit le fait que chaque donnée liée à un utilisateur est bien remplie	Amener le pourcentage P de qualité de données en lien avec la complétude des données à 100%.	Formule : P = A/B où A est le nombre de données présentes et B est le nombre de données requises
	Conformité : Critère qui décrit le degré auquel les données respectent les standards, règles et conventions en vigueur en lien avec la qualité de données.	Amener le pourcentage P de conformité des données à 100%.	Formule : P = A/B où A est le nombre de données conformes aux réglementations et B est le nombre de données totales.
	Précision : Critère qui décrit le degré auquel les données des utilisateurs contiennent des attributs qui discriminent les bonnes données dans un contexte spécifique.	Amener le pourcentage P de précision des attributs dans la base donnée à 100%.	Formule : P = A/B où A est le nombre de valeurs ayant la précision requise
Fiabilité	Disponibilité : Critère qui décrit le temps entre les pannes de l'application	Amener le pourcentage P de temps entre les pannes à 99.5% (30 minutes par semaine).	Formule : P = A/B où A est le temps entre les pannes et B est le temps total
	Tolérance aux pannes : Critère qui décrit le pourcentage du système qui fonctionne lors d'une panne	S'assurer que le pourcentage du système qui reste fonctionnel en cas de panne est de 80%.	Formule : P = A/B où A est le pourcentage de fonctionnalités fonctionnelles en cas de panne et B est le nombre total de fonctionnalités.

	Récupérabilité : Critère qui décrit le degré auquel le système peut fonctionner dans le cas d'erreur ou de *crash*.	Amener le pourcentage P de données perdues lors d'une erreur à moins de 1%.	Formule : P = A/B où A est le nombre de données perdues lors d'un *crash* et B le nombre de données total.
Maintenabilité	Modifiabilité : Critère qui décrit le degré auquel le système peut être facilement modifié	Réduire le pourcentage P de fichiers nécessaires à modifier lors d'une seule modification à 2 ou moins.	Formule : P = A/B où A est le nombre de fichiers modifiés par modification et B est le nombre de modifications.
	Testabilité : Critère qui décrit le degré auquel les composants de l'application peuvent être testés individuellement avec facilité	Amener le pourcentage P de focntions autonomes à 70%.	Formule : P = A/B où A est le nombre de fonctions autonomes et B est le nombre de fonctions.
	Réutilisabilité : Critère qui décrit le nombre de réutilisation des fonctions de l'application	Avoir un nombre moyen P de fonctions de réutilisés plus grand que 2.	Formule : P = A/B où A est le nombre de fois que chaque fonction est utilisée et B est le nombre de fonctions.

Stratégie de validation

Il y a plusieurs stratégies de validation qui peuvent et doivent être utilisées dans un projet de cette envergure. Une première stratégie, la moins formelle est la revue personnelle. Elle consiste tout simplement en un processus effectué tout au long du développement où le développeur prend des pauses de développement afin d'effectuer des *checklists* de points à réviser dans son code afin de s'assurer du bon fonctionnement de celui-ci. Il utilise ensuite ces listes afin de faire une révision pour minimiser le nombre de bugs et problèmes qui pourraient survenir.

Une autre stratégie qui peut être utile pour AntennaPod serait les revues par les pairs. Plus précisément, un walk-through serait la stratégie optimale de validation car, dans un contexte ouvert au public comme celui-ci, les pairs vont remarquer des problèmes et laisser des commentaires pour le développeur qui a codé la fonctionnalité en question. Les développeurs qui effectueront les revues devront donc présenter à l'auteur de la fonctionnalité une présentation détaillée de leur revue. Ce type de revue pourra aider à l'analyse de la maintenabilité du projet puisque les pairs analyseront, en partie, la maintenabilité du code.

Question 2 : Stratégie de testing

Plan des tests

Pour tester AntennaPod, nous pourrions choisir la méthode du *Big Bang* ou la méthode incrémentale. Le *Big Bang* consiste à tester le système en entier une fois qu'il est complet. La méthode incrémentale consiste à réaliser des tests unitaires sur chaque composante. Ensuite, lors de l'intégration, il faudrait faire des tests d'intégration et, finalement, il faudrait faire des tests de sytème. Un avantage du *Big Bang* est que cette méthode est plus rapide à exécuter. En effet, il n'y a qu'une seule itération. La méthode incrémentale, quant à elle, contient plusieurs cycles. Elle a comme avantage de mieux trouver des bugs potentiels et de retrouver les problèmes plus facilement s'il y en a. Nous avons opté pour la méthode incrémentale, car nous avions du temps et nous consacrons beaucoup d'importance aux tests. Les tests unitaires ainsi que les tests d'intégration sont principalement dans le module *core*. Finalement les tests de système sont principalement dans le module *app*.

Mesures et méthodes pour valider les objectifs

Objectif	Algorithme(s) concerné(s)	Test ou diagnostic	Description	Logiciel	Objectif de couverture
Complétude	Les algorithmes concernés sont décrits avec les descriptions.	Test unitaire d'égalité	Les descriptions de tous les objectifs sont écrites en dessous	JUnit	100%
Conformité		Test unitaire d'égalité		JUnit	100%
Précision		Test unitaire d'égalité		JUnit	100%
Disponibilité		Audit disponibilité		NA	90%
Tolérance aux pannes		Test unitaire d'égalité et Audit tolérance aux pannes		JUnit	99,5%
Récupérabilité		Vérification manuelle		NA	80%
Modifiabilité		Audit modifiabilité		NA	90%
Testabilité		Audit testabilité		NA	95%
Réutilisabilité		Audit réutilisabilité		NA	90%

Fonctionalité :

Complétude : Nous pouvons voir dans les méthodes de la classe *MainActivityTest* que les entrées sont remises à celles par défaut lorsque l'utilisateur modifie un champ et appuie ensuite sur *back*. Nous pouvons rajouter un test unitaire très simple cette fonctionnalité. Nous voyons dans le fichier *renameItemDialog* que, si l'entrée de l'utilisateur est vide, nous devons la remettre à celle par défaut. Il faudrait tout simplement simuler une entrée vide et s'attendre à recevoir *R.string.rename_tag_label*.

```
.setTitle(feed != null ? R.string.rename_feed_label : R.string.rename_tag_label)
```

Conformité : Nous pouvons voir que les méthodes de la classe *URLCheckerTest* s'assurent toutes que les url entrés sont justes. Ainsi, lorsque un URL est entré, on peut voir qu'il respecte les exigences.

Précision : À l'aide des tests *testComboFilter* et *testMinimalDurationFilter*, nous pouvons nous assurer que les champs remplis par l'utilisateur sont acceptés. En effet, avec le premier test, il ne peut pas entrer un mot qui est refusé et il doit entrer au moins un mot accepté. Ensuite, avec le second, on confirme que la durée du filtre est plus grande que la durée minimale.

Fiabilité :

Disponibilité : Il y a un temps infini entre les pannes. En effet, AntennaPod n'utilise pas de serveur central. Toutes les informations qu'il utilise sont des données publiques sur internet. Par conséquent, même si les sites des créateurs de podcast ne fonctionnent plus, AntennaPod continuera à fonctionner.

Tolérance aux pannes : Nous pouvons voir le test *test404* qui s'assure que, lors d'une panne de connexion, les fonctionalités du système qui ne nécessitent pas de connexion fonctionnent encore. Par exemple, les éléments téléchargés fonctionnent encore. De plus, sur le site d'AntennaPod, il est dit que l'application va chercher les informations des podcasts sur les sites sur lesquels elles ont publiées.

Récupérabilité : Une multitude de tests sont réalisés sur la base de données que ce soit pour envoyer ou recevoir des données. Ces tests sont situés dans *DbReaderTest* et *DbWriterTest*. Ainsi, si l'application crash, les données seront tout-de-même récupérables vu qu'elles se retrouvent sur la base de données.

Maintenabilité :

Modifiabilité : On peut voir que le système est facilement modifiable. Par exemple, le test *testUploadSubscription* et *testUploadSubscription2* sont des tests très similaires. Nous n'avons ajouté qu'un élément de plus. Nous pouvons donc extrapoler que nous pouvons faire de même pour un nombre infini d'éléments.

Testabilité : Nous pouvons voir que tous les tests sont exécutés dans des fragments individuels. Cela confirme donc que toutes les composantes de l'applications peuvent être testées individuellement.

Réutilisabilité : Les tests *testShareDiaologDsiplayed* et *testShareDialogCancelButton* sont des tests qu'on réutilise un peu partout dans l'application. En effet ces tests s'assurent que les boites de dialogues fonctionnent bien. Les boites de dialogues sont, pour la majorité, des vues de l'application et on réutilise toujours la même composante.

Nous pouvons retrouver la majeure partie des tests d'AntennaPod dans les dossiers ci-dessous.

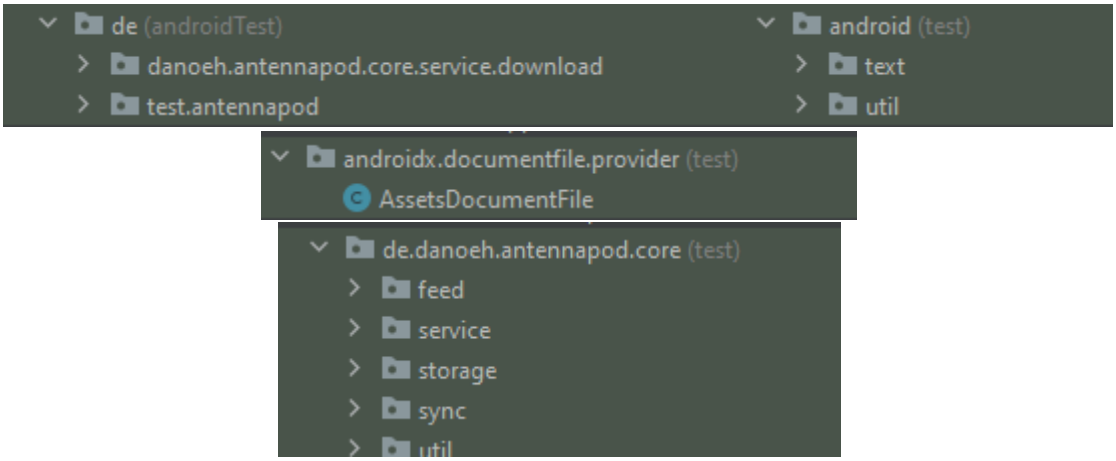

Figure 1 : tests unitaires d'AntennaPod

Ici, on peut voir que tous les tests unitaires passent et que les tests systèmes et d'intégration fonctionnent aussi, car les tests faits par l'émulateur fonctionnent en plus d'avoir testé à la main que le système fonctionnait.

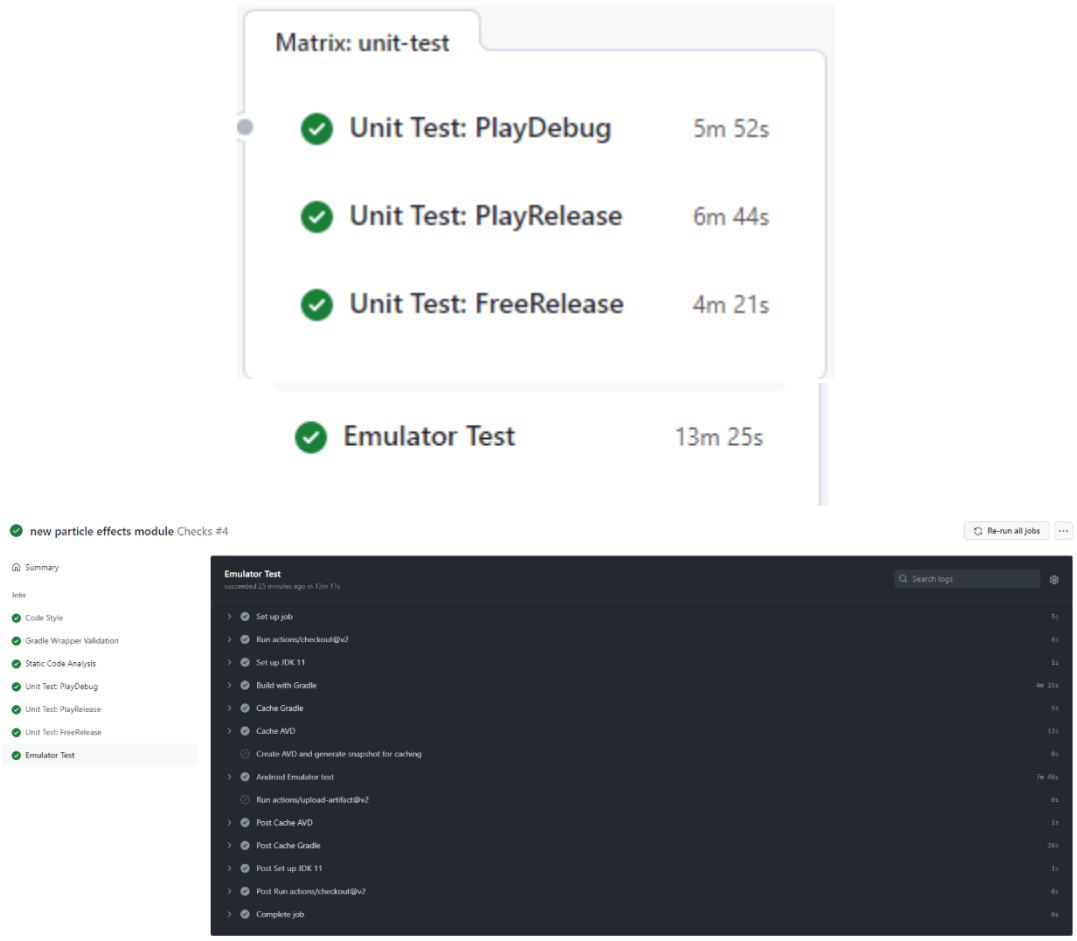

Figure 2 : Résutat des tests d'AntennaPod sur GitHub Actions

Les tests unitaires passent localement comme on peut voir ici en lançant la commande *gradlew test*. `BUILD SUCCESSFUL in 8m 48s`

Cependant, pour le module *App*, certains des tests de l'émulateur ne fonctionnent pas localement. On peut voir que c'est à cause de l'émulateur utilisé. En effet, on peut voir que, avec deux émulateurs différents, nous obtenons des résultats de tests largement différents. Nous pouvons cependant voir que les tests d'émulateur passent tous sur la pipeline. Les résultats des tests ont été inclus dans l'annexe.

Question 3 : Intégration continue

Plan d'intégration continue

Pour faire l'intégration continue du projet, nous avons opté pour l'outil GitHub Actions. Le choix de cet outil est basé sur le fait que le projet original contenait déjà un pipeline sur GitHub Actions, et donc par souci de simplicité, nous avons utilisé le même outil pour construire notre propre pipeline. Ce faisant, nous avons pu nous inspirer fortement du pipeline original du projet, qui est très complet et qui couvre très bien les critères de qualités que nous avons posé dans notre plan. Pour ajouter l'outil d'intégration continue au projet, il faut ajouter un fichier de configuration de *workflow*, dans notre cas checks.yml, dans le dossier .github/workflows/ associé au projet. Ensuite, après avoir activé GitHub Actions sur la page du repo GitHub (l'onglet Actions), le pipeline sera activé et opérationnel. Le pipeline sur GitHub Actions permet l'exécution automatique des tests unitaires, des tests d'intégration, des tests de système et de d'autre vérifications complémentaires tels que la vérification du style, du *gradle-wrapper* ainsi que du code statique. Le pipeline a été configuré pour que l'ensemble des étapes soient exécutées lorsque l'on modifie, en faisant un push par exemple, la branche *master* ou *develop* du projet. Lorsque le *pipeline* est lancée, les taches vont être exécutées selon l'ordre séquentiel du pipeline, chacune dans une machine virtuelle ou un conteneur séparé, dépendamment de la tâche et des paramètres du workflow. Ces configurations se retrouvent dans un fichier *Yaml*, qui détermine ce que le pipeline doit faire, comment et dans quel ordre.

https://github.com/AntennaPod/AntennaPod/blob/924543b0202c39013a6cea9b59255176d83c9ec2/.github/workflows/checks.yml

Pour notre *pipeline*, les vérifications sont exécutées comme suit :

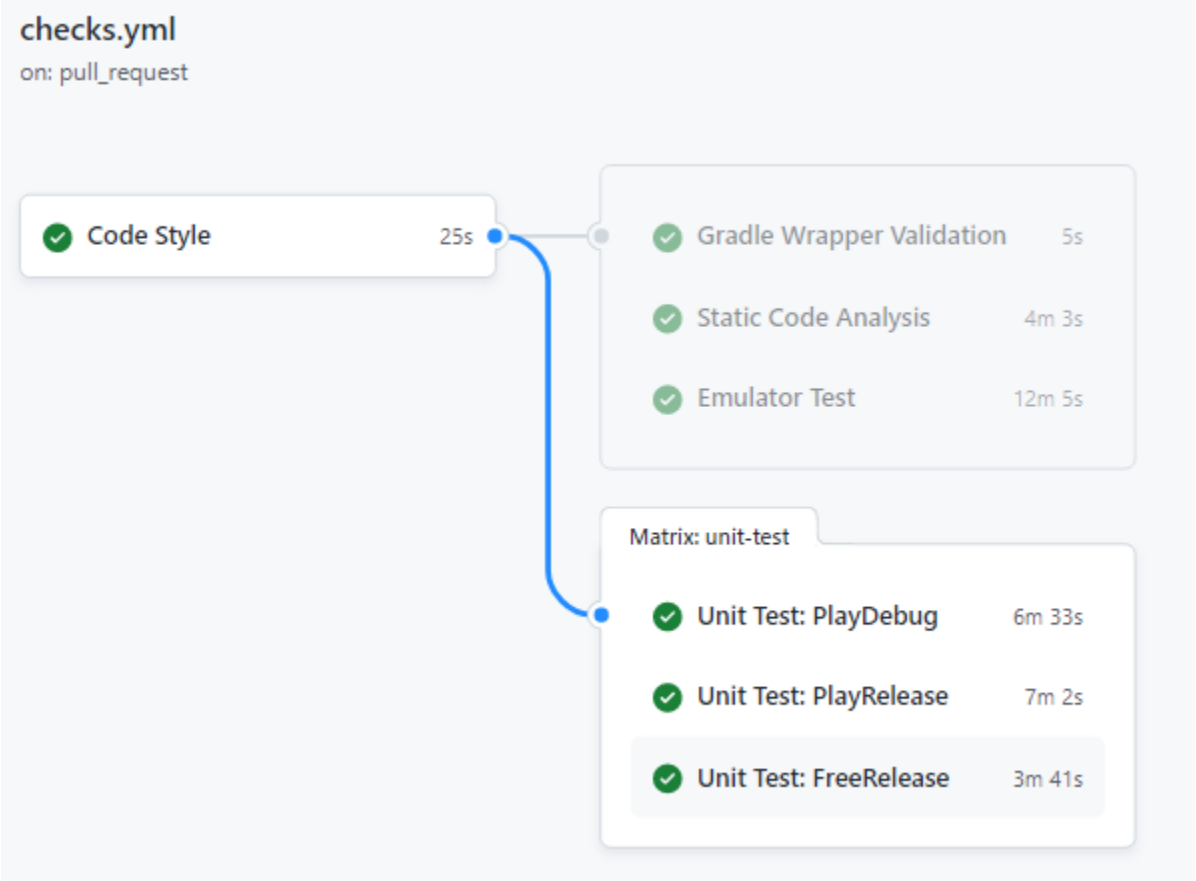

Figure 3 : Résultat de l'exécution du pipeline sur GitHub Actions

En partant de code style, qui teste la validité et l'apparence du code, notamment les fichiers xml et Java, à l'aide des outils diff-checkstyle et android-xml-formatter. Ces vérifications se font seulement sur les ajouts du dernier commit sur la branche, pour éviter de revalider plusieurs fois du code qui n'a pas été changé.

Ensuite, lorsque le *code style* est validé, les autres tâches sont lancées :

- Gradle wrapper validation : utilise l'outil *gradle-wrapper* pour trouver des erreurs dans les fichiers *gradle*.
- Static code analysis : applique un linter sur le projet pour trouver des erreurs communes, détectables par une analyse du code.
- Emulator test : build le projet sur une machine virtuelle et lance les tests d'émulateur Android. Ces tests incluent les tests de systèmes du projet.
- Unit Tests : lance les tests unitaires selon 3 différentes séries de tests, basé sur les types de build : *PlayDebug*, *PlayRelease* et *FreeRelease*. Les tests exécutés diffèrent selon le type de build précisé.

Ajout d'un nouveau module

Supposons que nous devons ajouter un nouveau module, par exemple *playback* pour AntennaPod. Pour garantir la qualité du système, il faut s'assurer que des tests unitaires ont été ajoutés pour tester le comportement du module, mais aussi que des tests d'intégration sont présent pour tester l'intégration du module au reste du projet. De plus, l'ajout de test de système est fortement recommandé pour valider que le comportement lors de l'utilisation est celui attendu. L'ensemble de ces tests seront automatiquement lancés avec le reste des tests unitaire une fois que le module sera ajouté à la branche *develop*. Cela va garantir que le module est testé convenablement et bien intégré dans le reste du projet. De plus, le *pipeline* valide l'ensemble de la syntaxe, le *gradle-wrapper* et le build, ce qui offre une validation supplémentaire contre les erreurs qui ne sont pas détectées par les tests.

Dans cet exemple, il n'est pas nécessaire de mettre à jour le plan de qualité, car le plan est défini pour assurer la qualité globale du logiciel et non pour des fonctionnalités spécifiques. Du moment que le nouveau module n'introduit pas de fonctionnalité majeure qui ne serait pas pris en compte comme le support d'une autre plateforme, par exemple une application de bureau, l'intégration de celui-ci sera automatiquement pris en compte par l'outil de CI. C'est d'ailleurs un des avantages de faire de l'intégration continue. Dans notre exemple, l'ajout du module *playback* est très bien pris en compte par le plan de test, qui valide l'ensemble du module et de son intégration, sans avoir besoin d'être modifié. Cela vient également avec la condition que le module est adéquatement testé au préalable et que des tests ont été rédigés. Si c'est le cas, alors le plan de test va exécuter ceux-ci avec le reste des tests et valider si l'ajout du nouveau module assure que le projet respecte toujours les objectifs de qualité.

Démonstration de l'intégration continue du système

Pour démontrer l'intégration d'un module à l'aide de l'outil de CI, nous avons joint une vidéo qui montre l'ajout d'un module SpecialEffects au projet AntennaPod. Il est à noter que dans la vidéo, l'implémentation du module a été abrégé et que bien que ce n'est pas explicitement montré, le module ajouté contient des tests unitaires et d'intégration.

https://www.youtube.com/watch?v=iF_IuRCMqfk

Annexe

Tests *App* sur pixel 2

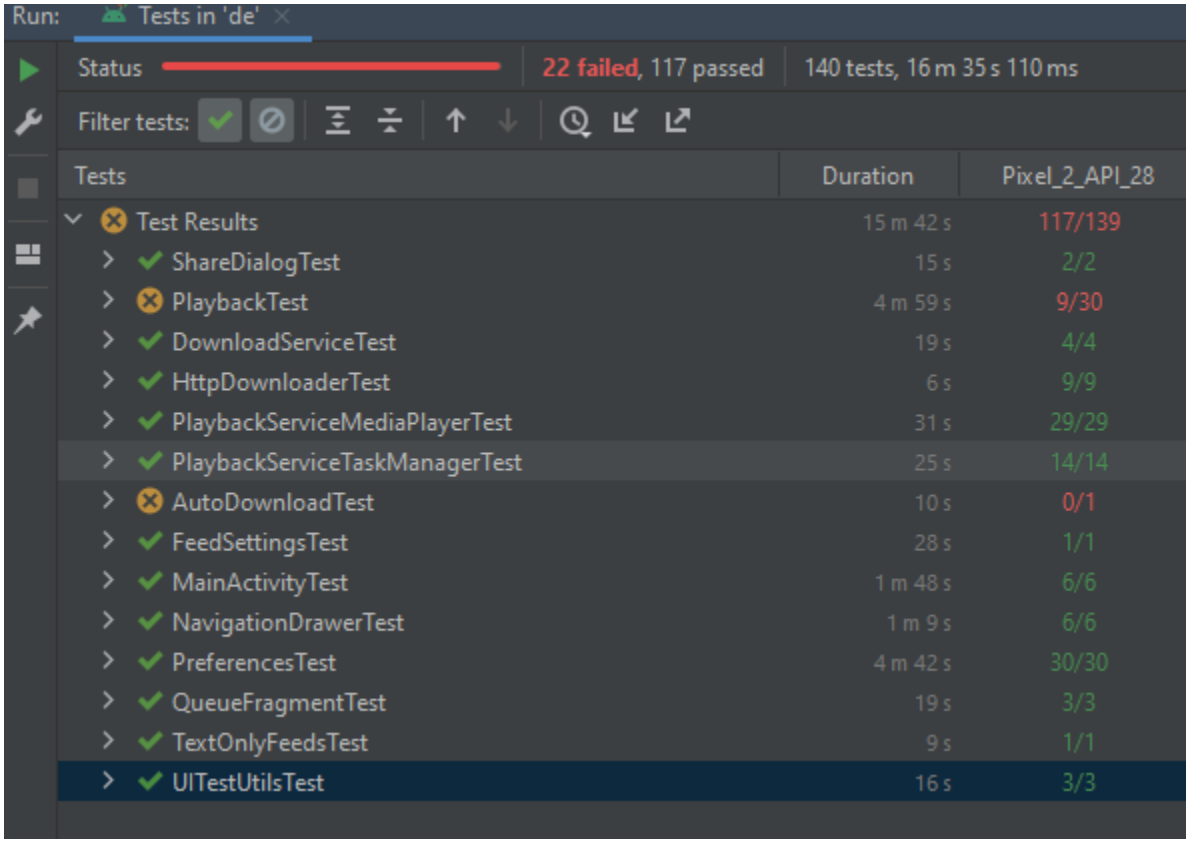

Figure 4 : Résutat des tests d'AntennaPod sur un émulateur du Google Pixel 2

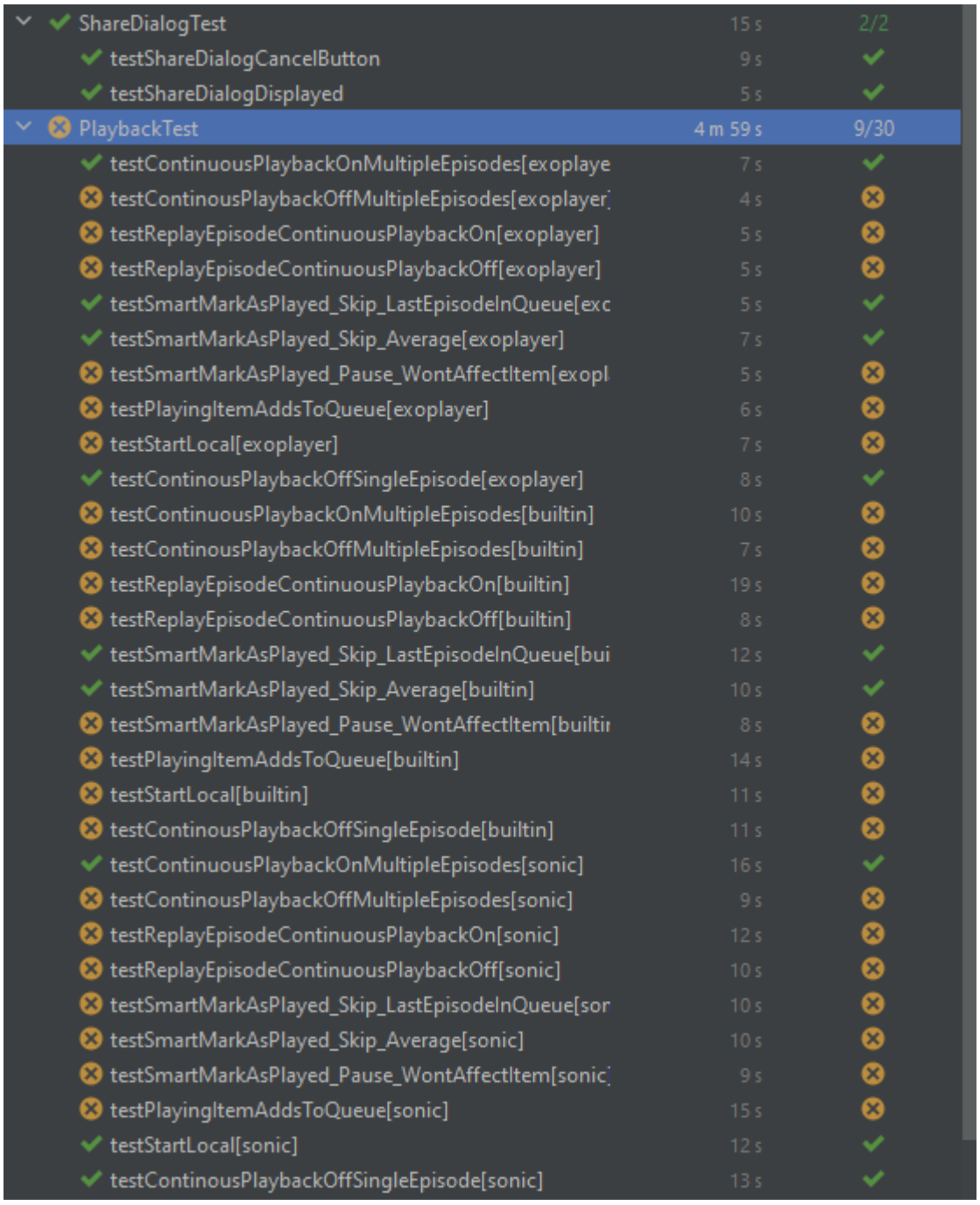

Figure 5 : Résutat des tests d'AntennaPod sur un émulateur du Google Pixel 2 (suite)

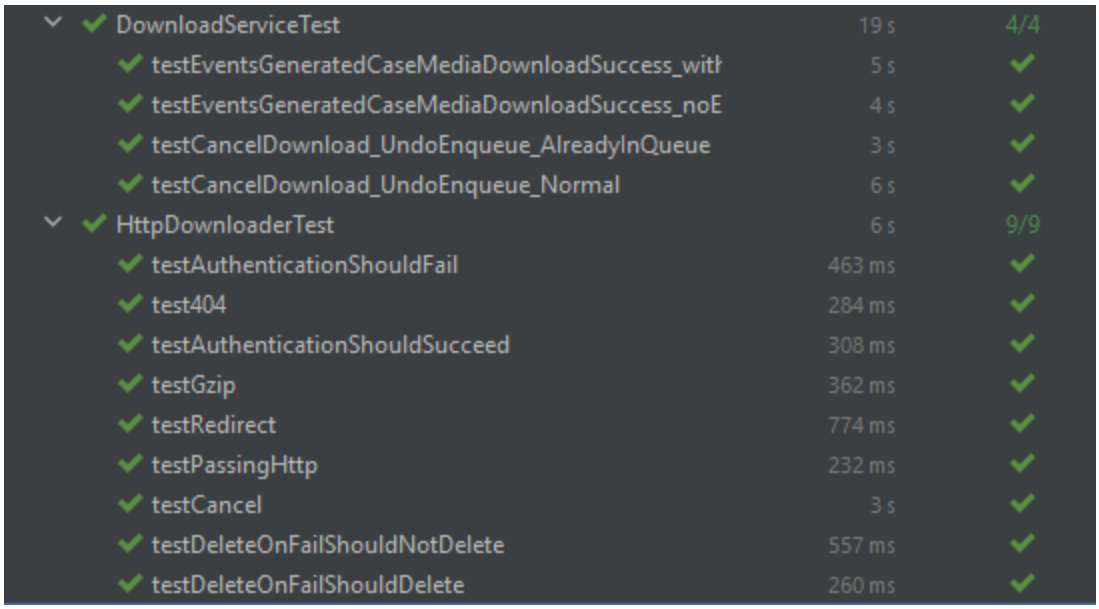

Figure 6 : Résutat des tests d'AntennaPod sur un émulateur du Google Pixel 2 (suite 2)

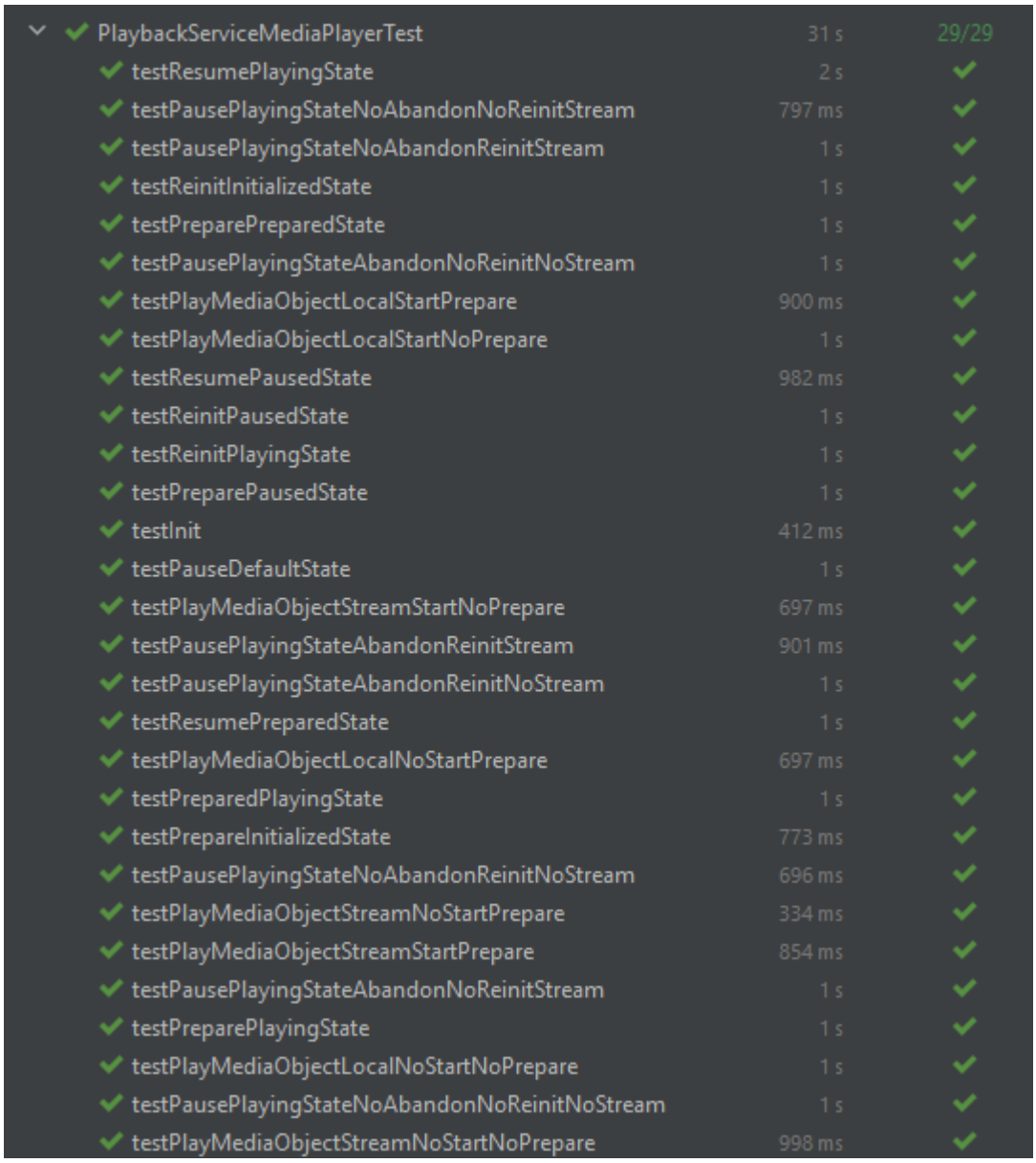

Figure 7 : Résutat des tests d'AntennaPod sur un émulateur du Google Pixel 2 (suite 3)

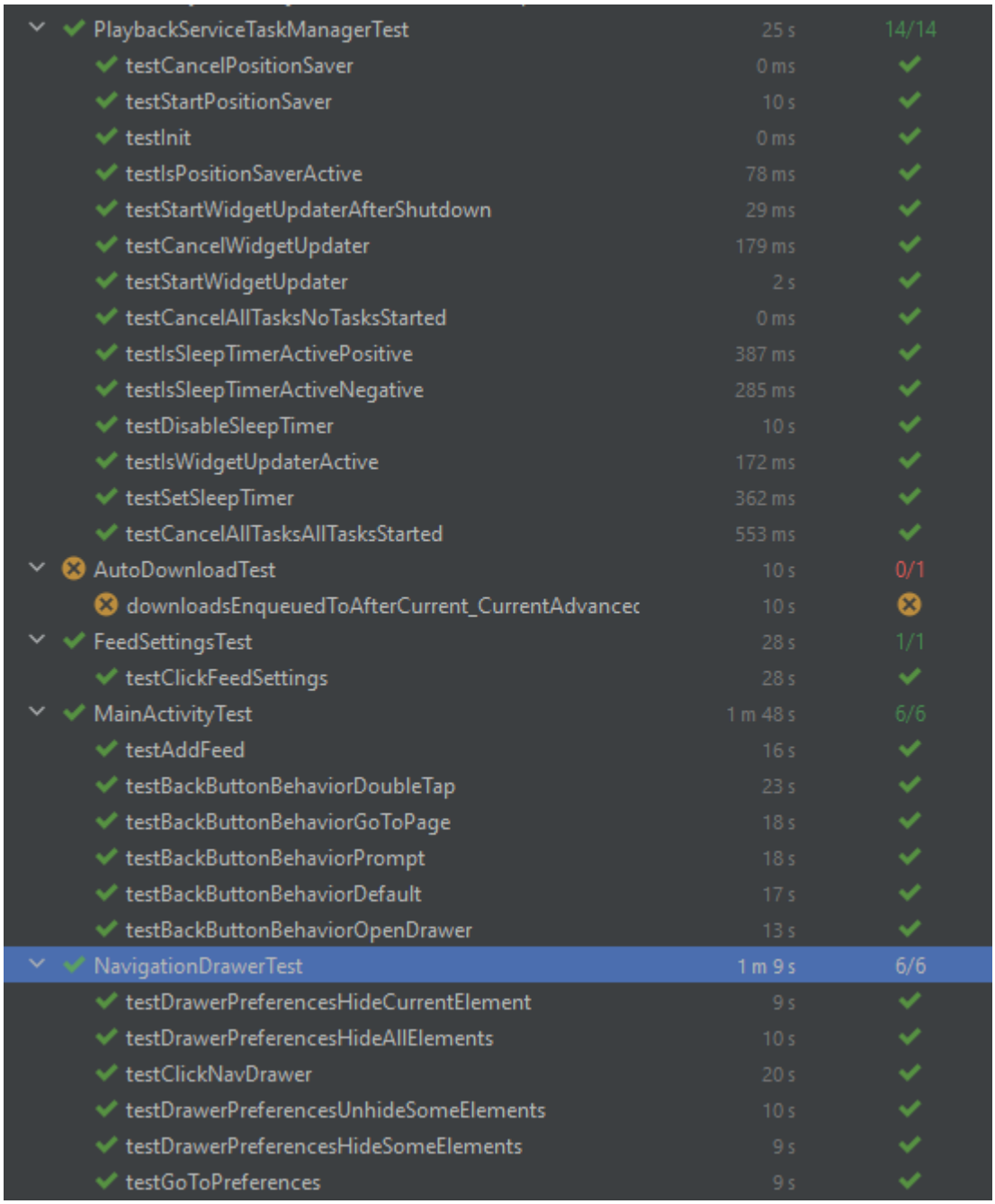

Figure 8 : Résutat des tests d'AntennaPod sur un émulateur du Google Pixel 2 (suite 4)

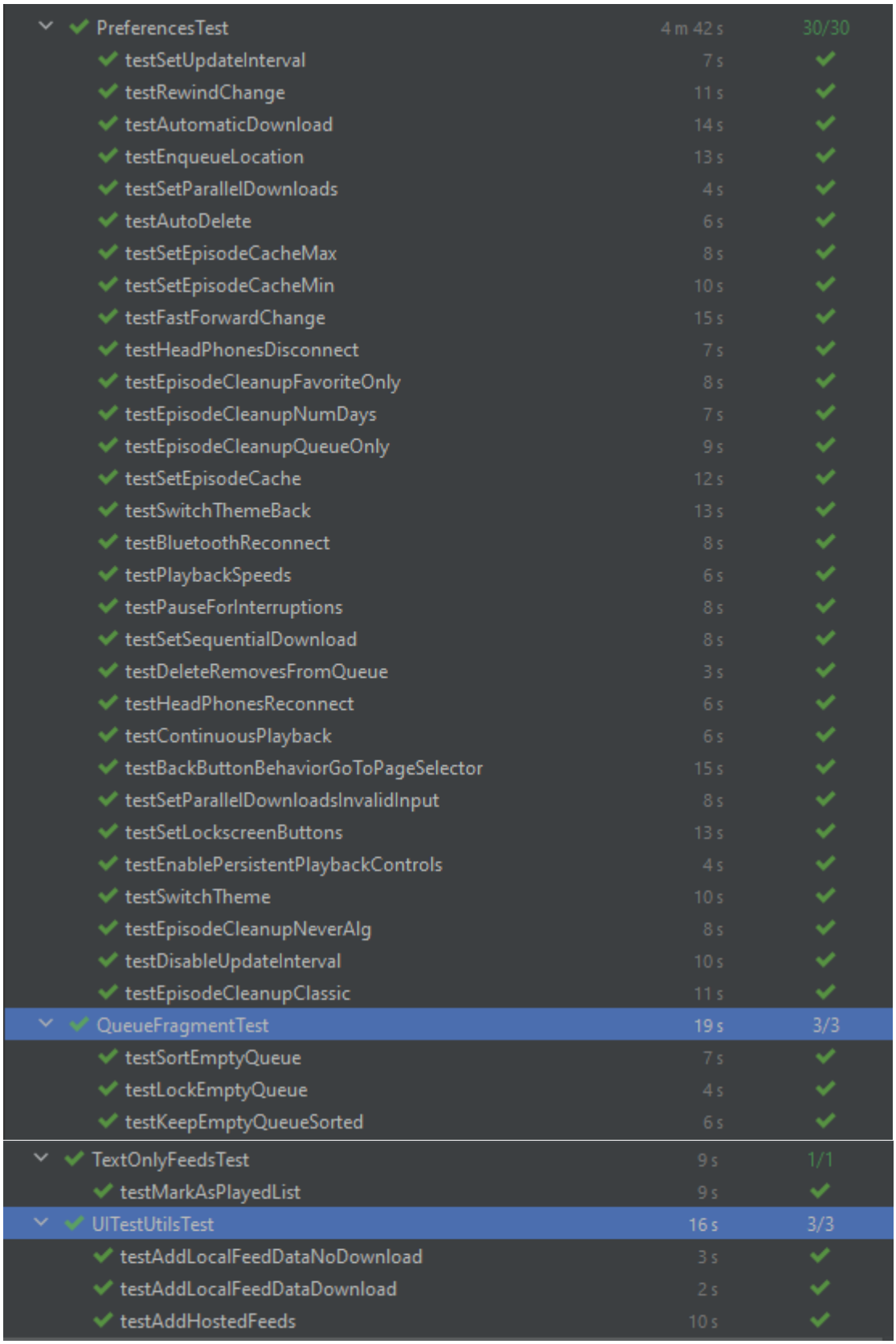

Figure 9 : Résutat des tests d'AntennaPod sur un émulateur du Google Pixel 2 (suite 5)

Test *App* sur tablette personalisée

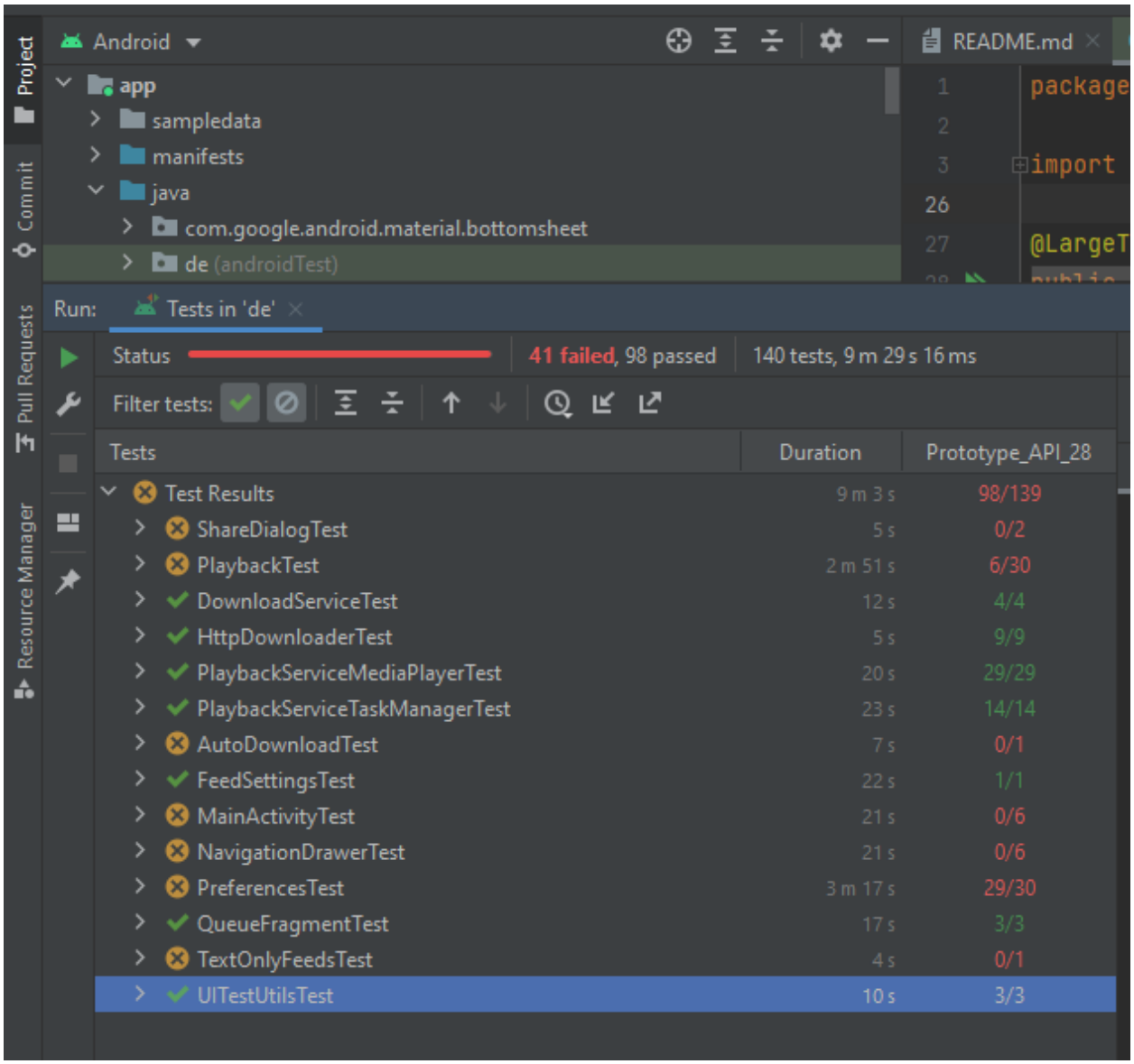

Figure 10 : Résutat des tests d'AntennaPod sur un émulateur d'une tablette Samsung A2019 WUXGA

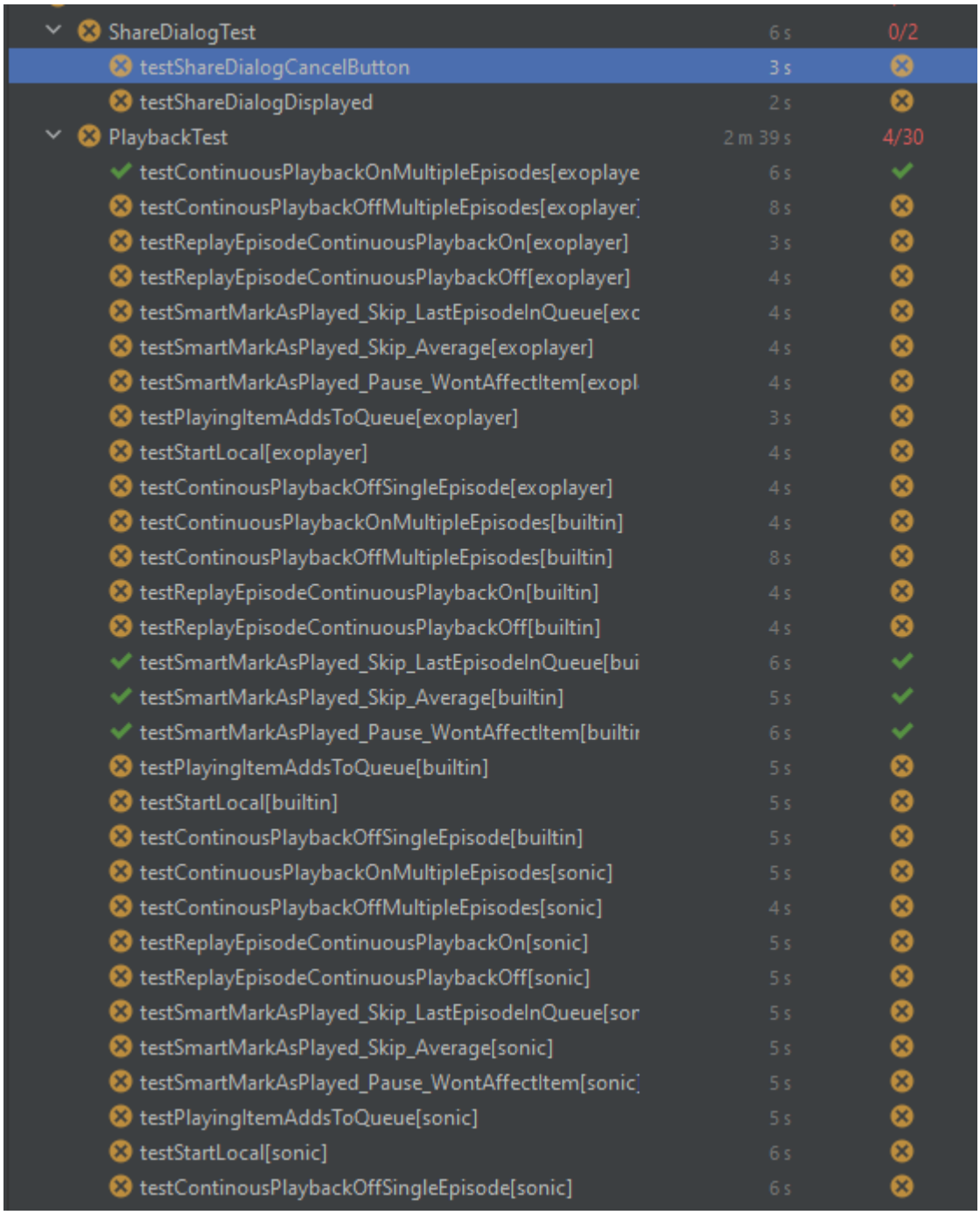

Figure 11 : Résutat des tests d'AntennaPod sur un émulateur d'une tablette Samsung A2019 WUXGA (suite)

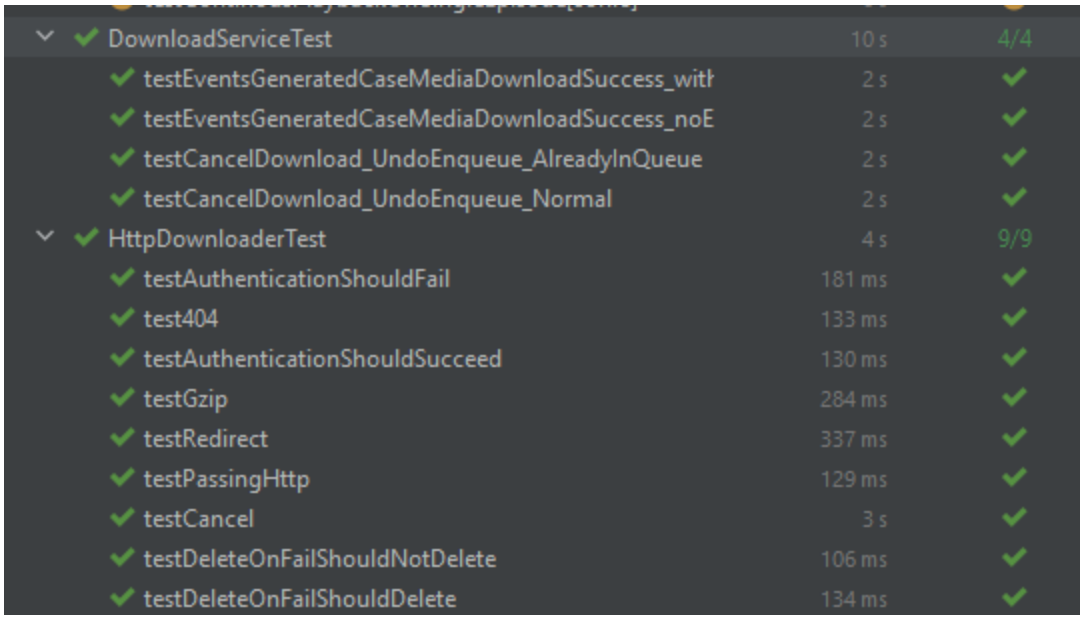

Figure 12 : Résutat des tests d'AntennaPod sur un émulateur d'une tablette Samsung A2019 WUXGA (suite 2)

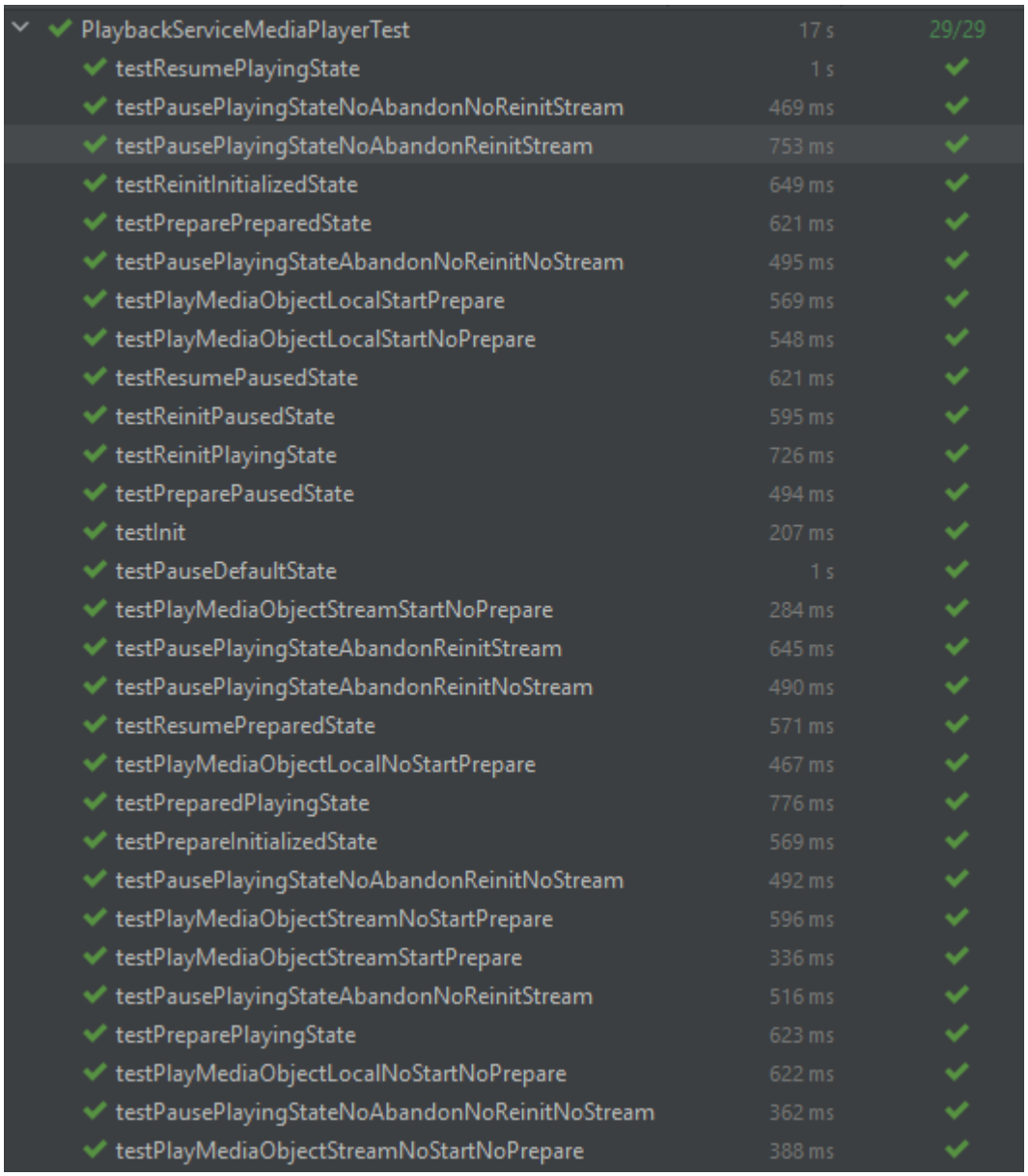

Figure 13 : Résutat des tests d'AntennaPod sur un émulateur d'une tablette Samsung A2019 WUXGA (suite 3)

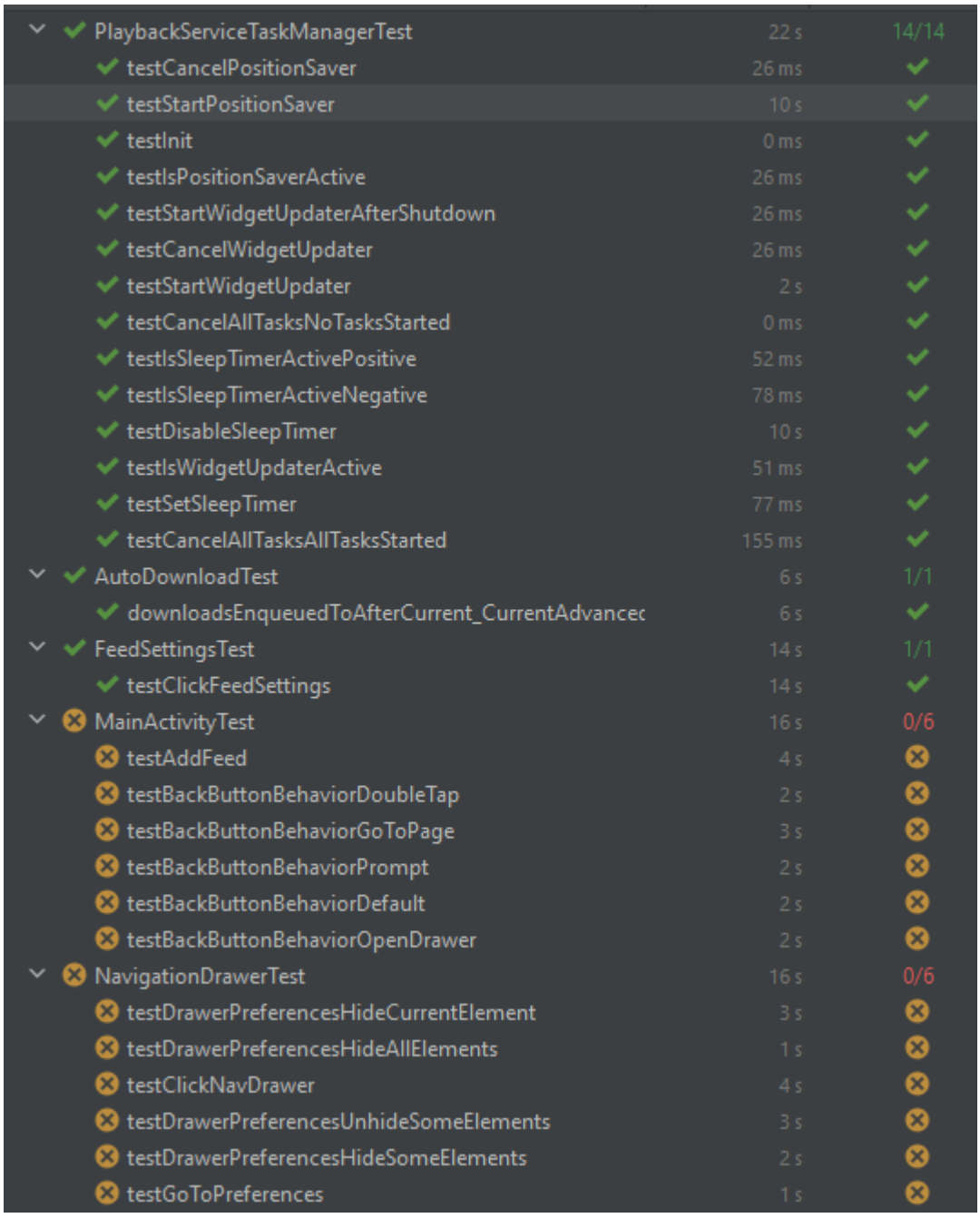

Figure 14 : Résutat des tests d'AntennaPod sur un émulateur d'une tablette Samsung A2019 WUXGA (suite 4)

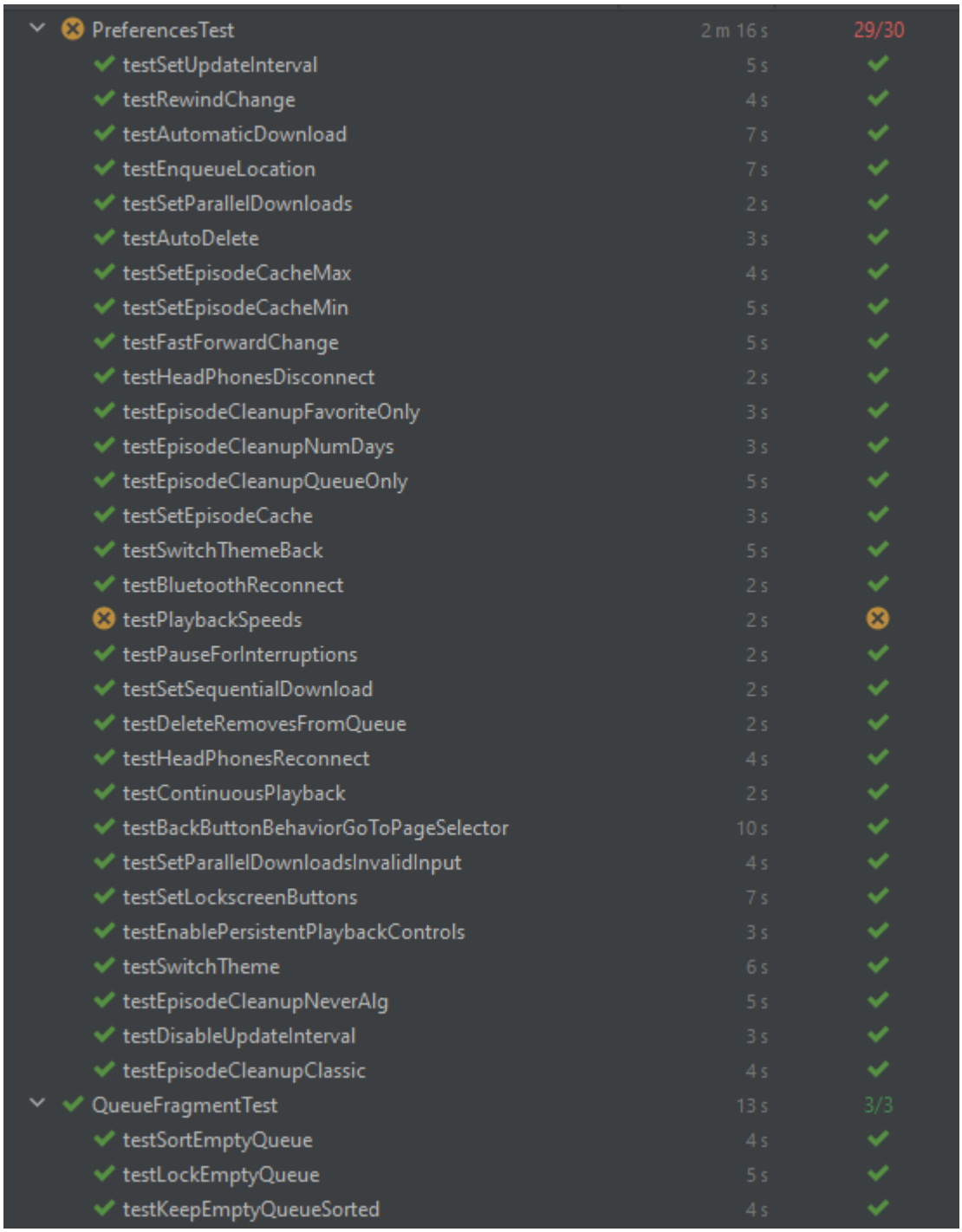

Figure 15 : Résutat des tests d'AntennaPod sur un émulateur d'une tablette Samsung A2019 WUXGA (suite 5)

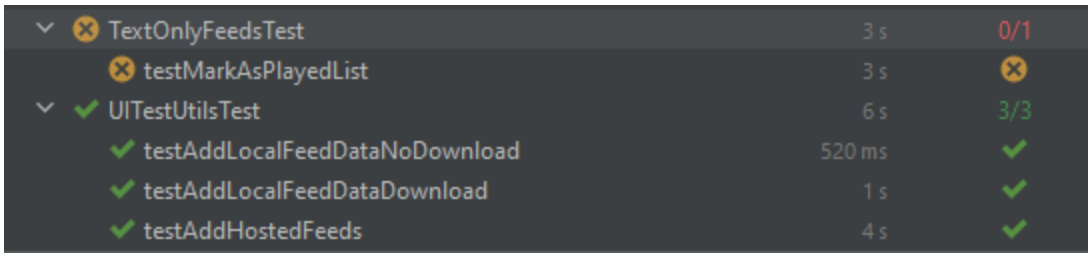

Figure 16 : Résutat des tests d'AntennaPod sur un émulateur d'une tablette Samsung A2019 WUXGA (suite 6)

Tests *Core*

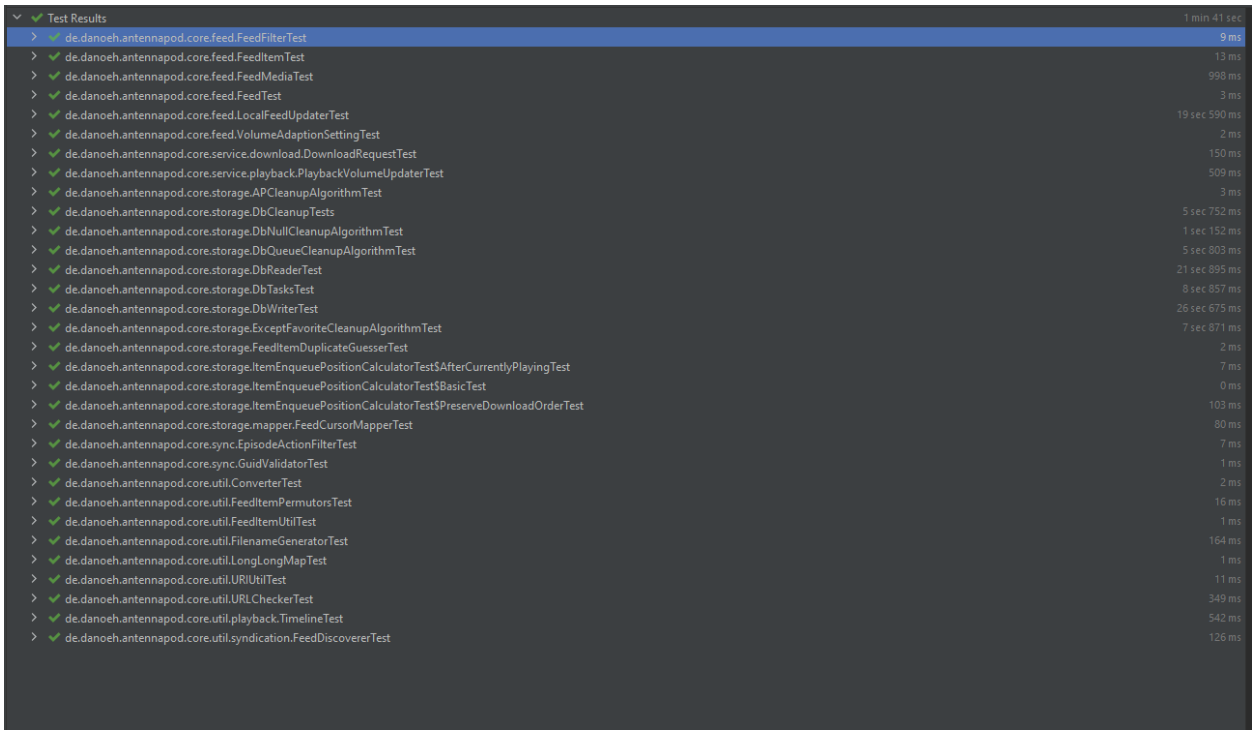

Figure 17 : Résutat des tests du module core d'AntennaPod

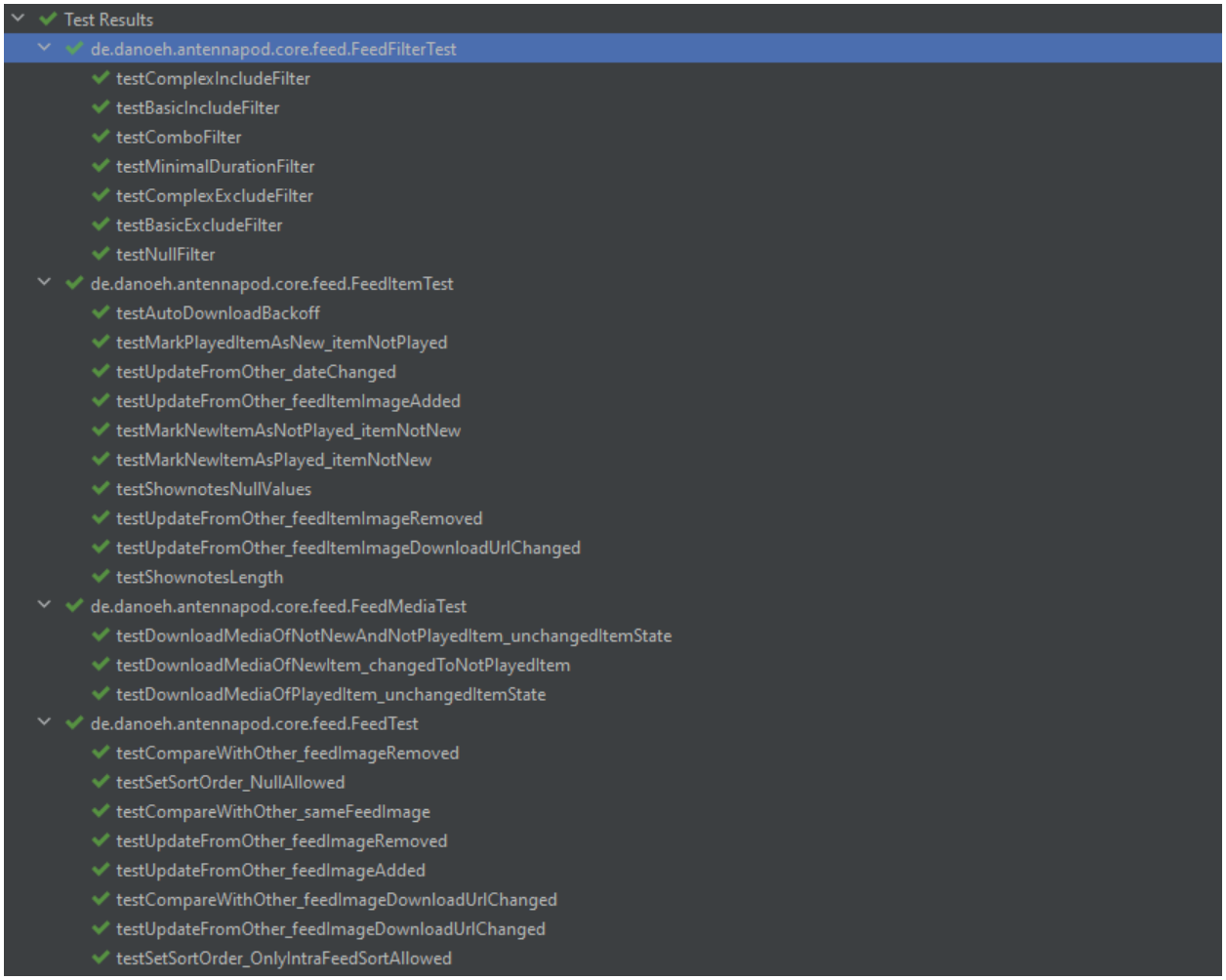

Figure 18 : Résutat des tests du module core d'AntennaPod (suite)

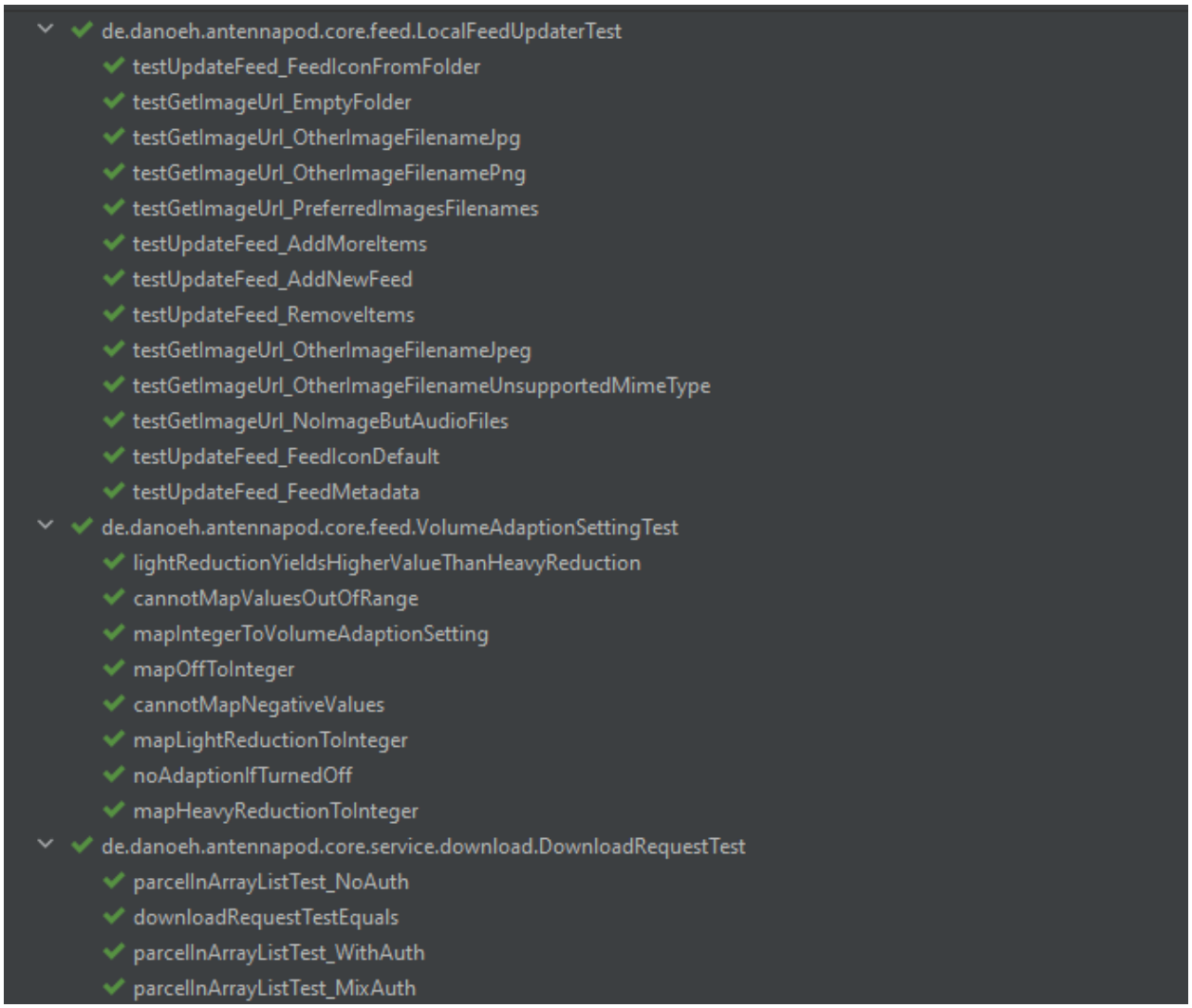

Figure 19 : Résutat des tests du module core d'AntennaPod (suite 2)

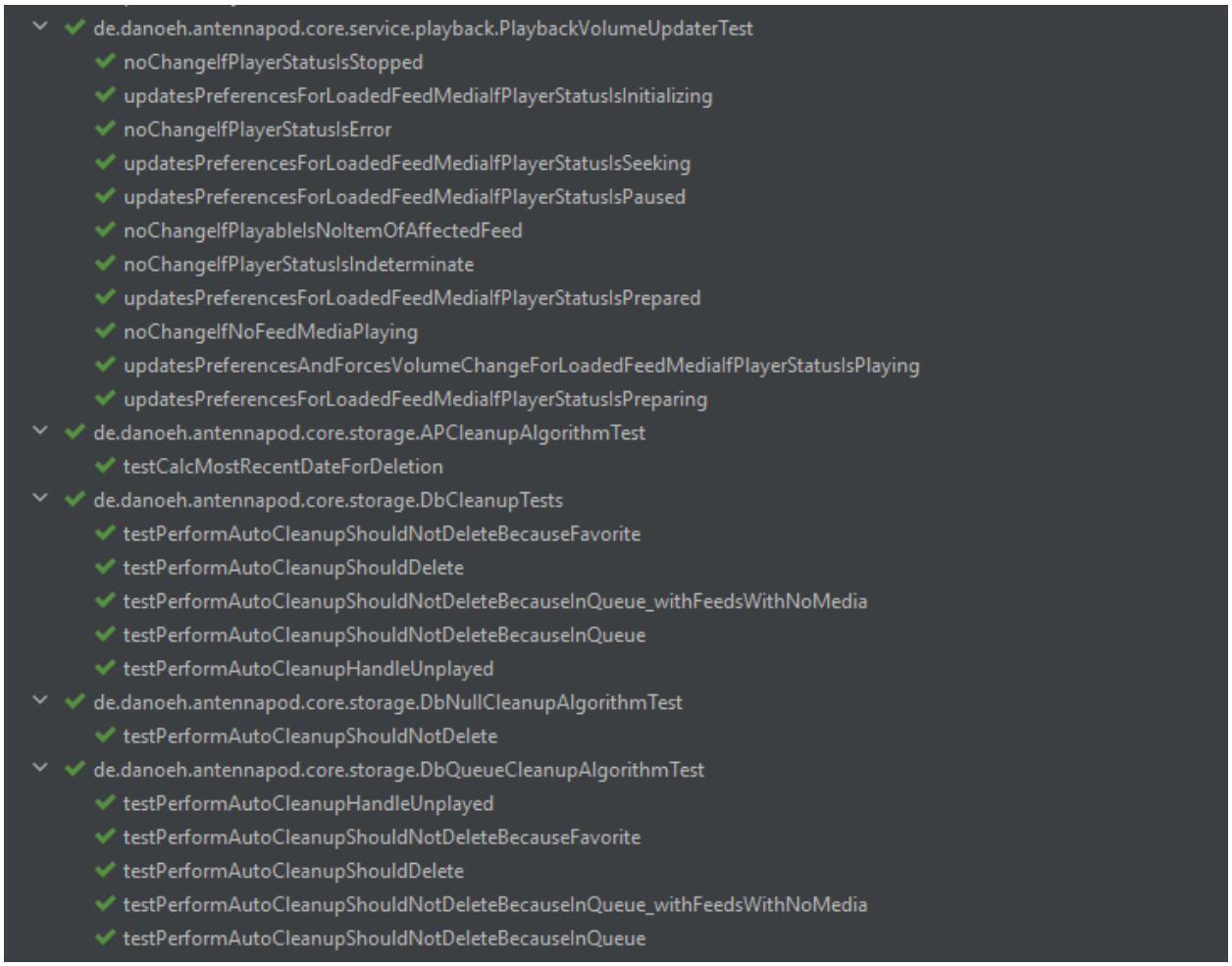

Figure 20 : Résutat des tests du module core d'AntennaPod (suite 3)

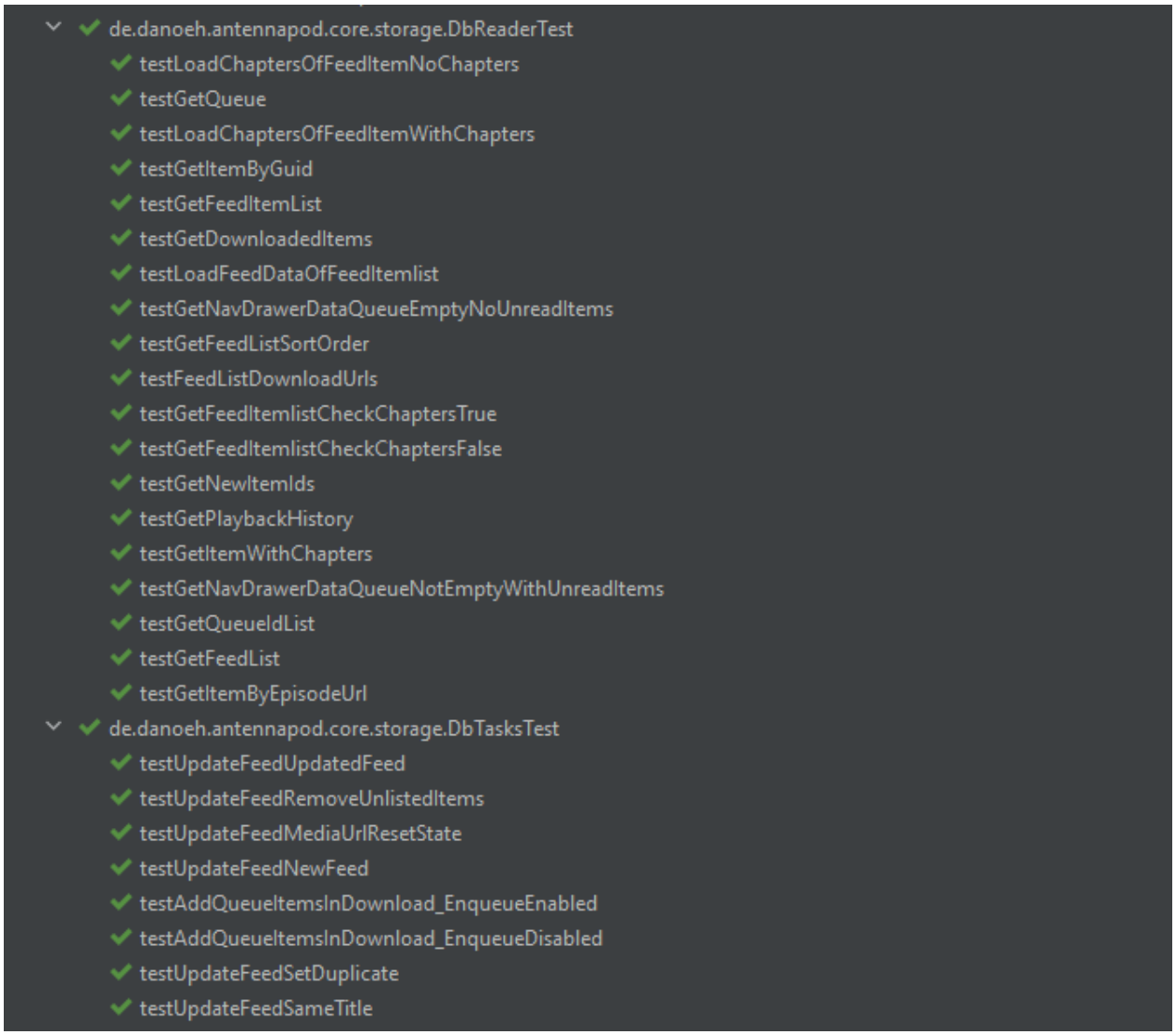

Figure 21 : Résutat des tests du module core d'AntennaPod (suite 4)

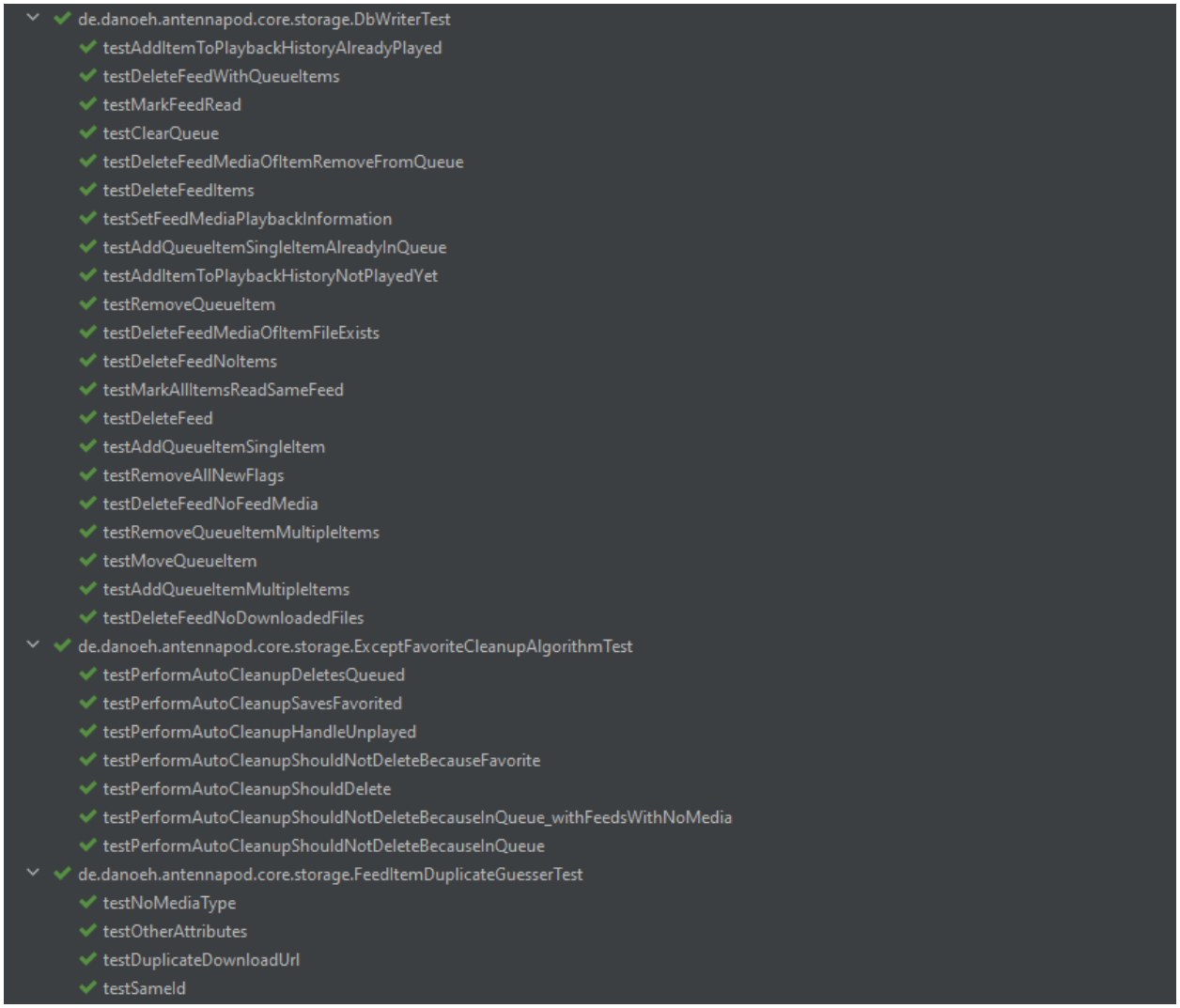

Figure 22 : Résutat des tests du module core d'AntennaPod (suite 5)

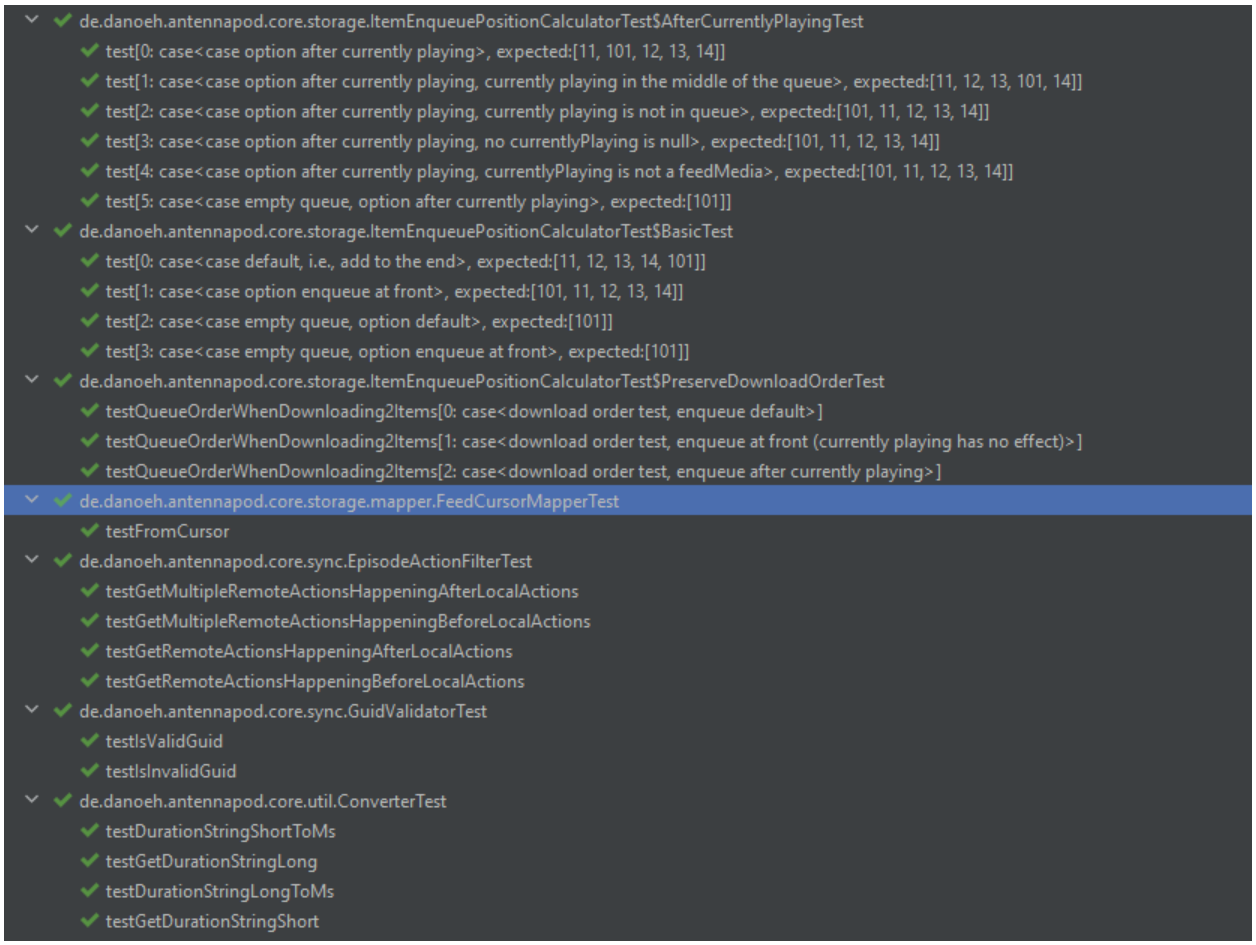

Figure 23 : Résutat des tests du module core d'AntennaPod (suite 6)

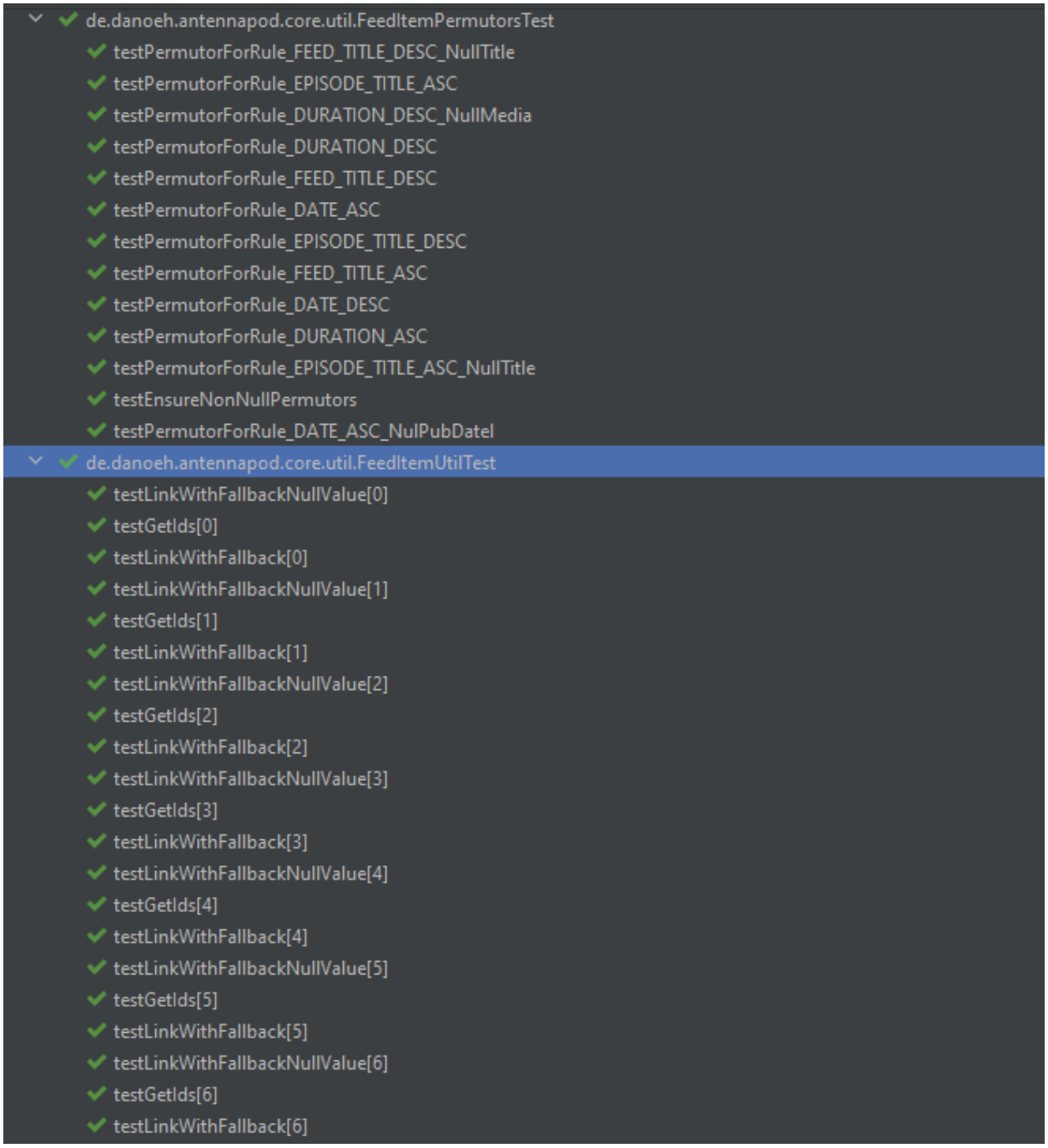

Figure 24 : Résutat des tests du module core d'AntennaPod (suite 7)

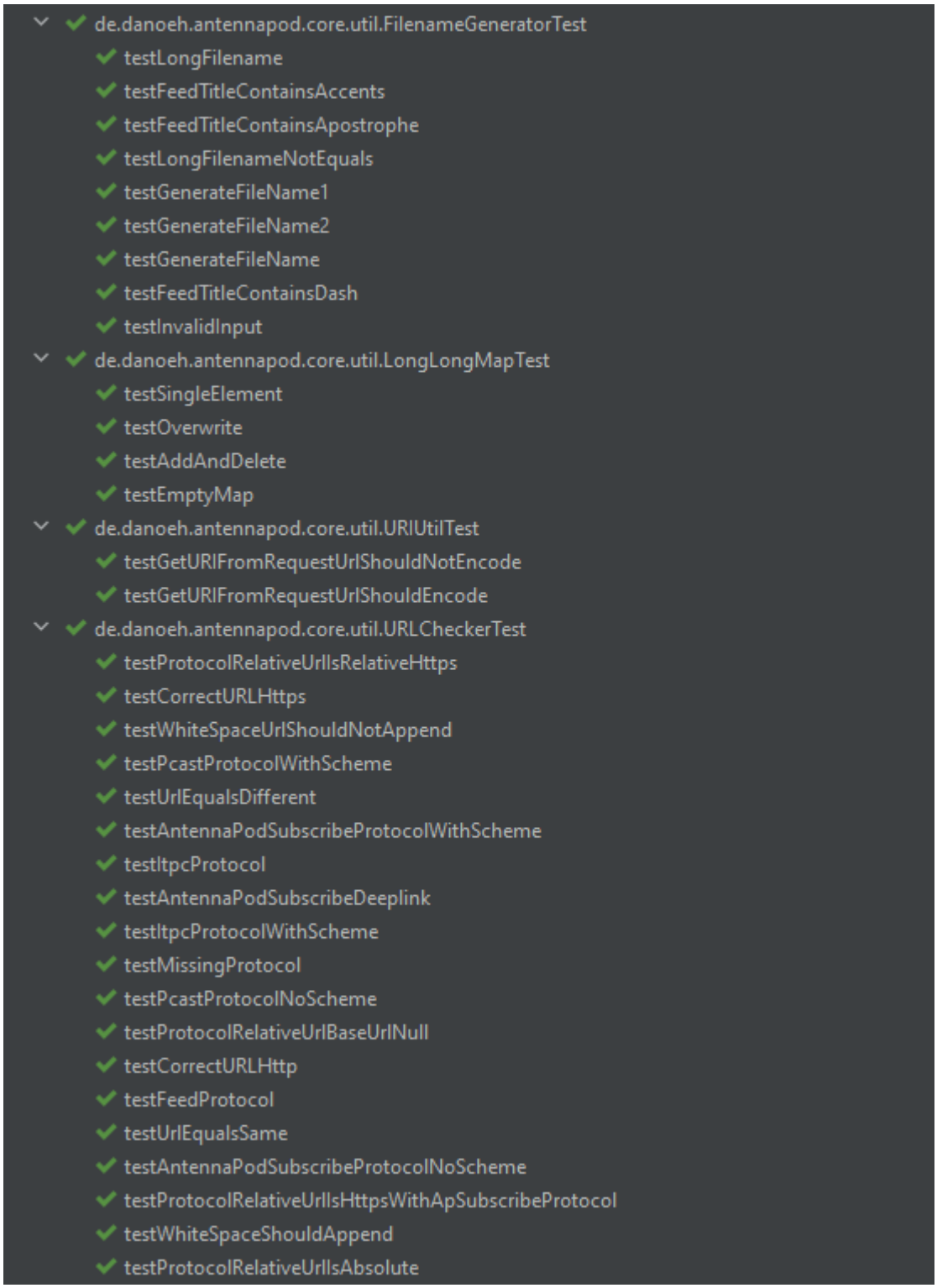

Figure 25 : Résutat des tests du module core d'AntennaPod (suite 8)

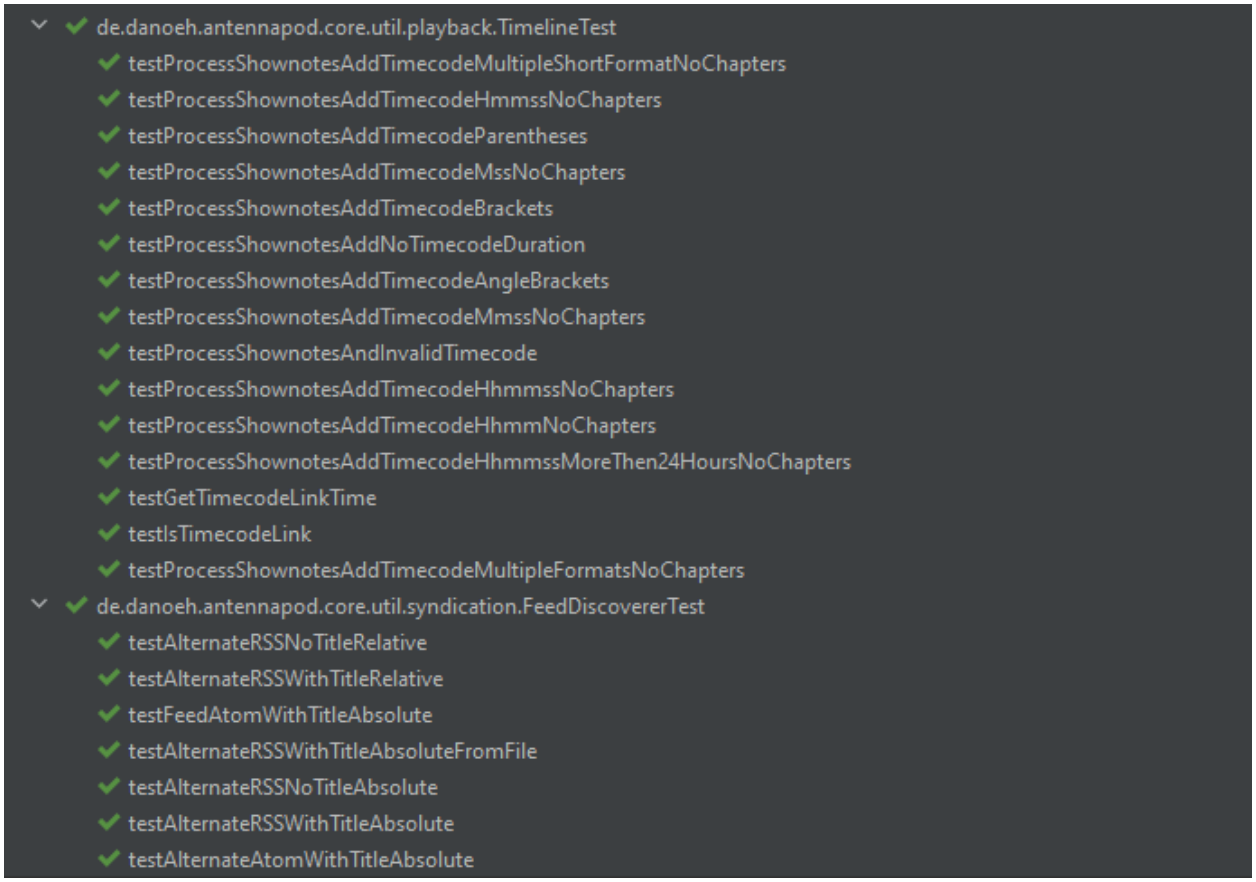

Figure 26 : Résutat des tests du module core d'AntennaPod (suite 9)

Explication des tests

Partie 2 (Testing) : J’ai eu des problèmes à faire fonctionner les tests de app localement. Alors, j’ai dû lancer les tests sur deux émulateurs différents pour que chaque test passe au moins une fois. On peut voir que les tests sur les dialogues ne passent pas sur un émulateur personnalisé. Ceci est le cas, car la taille de l'écran est non-conventionnelle. Nous pouvons donc aboutir à la conclusion que les tests passeraient sur un autre émulateur. Pour les tests de *playback*, il faudrait un émulateur d'un téléphone plus récent qui aurait plus de facilité à se connecter à l’internet.