Question 3 : Plan d’intégration continue

Q1

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 réutilisé ce pipeline. 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 :

Graphical user interface, text, application, chat or text message

Description automatically generated

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.

Q2

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. Cela signifie que le plan s’appliquera aussi bien sur un nouveau module qu’on ajouterait au projet, tant que celui-ci 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. 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.

Q3

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>