|  |
| --- |
| Projet JoQuiz |

Table des matières

[1 Analyse préliminaire 3](#_Toc499021832)

[1.1 Introduction 3](#_Toc499021833)

[1.2 Objectifs 3](#_Toc499021834)

[1.3 Planification initiale 3](#_Toc499021835)

[2 Analyse / Conception 4](#_Toc499021836)

[2.1 Concept 4](#_Toc499021837)

[2.2 Stratégie de test 4](#_Toc499021838)

[2.3 Risques techniques 4](#_Toc499021839)

[2.4 Planification 5](#_Toc499021840)

[2.5 Dossier de conception 6](#_Toc499021841)

[3 Réalisation 7](#_Toc499021842)

[3.1 Dossier de réalisation 7](#_Toc499021843)

[3.2 Description des tests effectués 12](#_Toc499021844)

[3.3 Erreurs restantes 14](#_Toc499021845)

[3.4 Liste des documents fournis 15](#_Toc499021846)

[4 Conclusions 14](#_Toc499021847)

[5 Annexes 15](#_Toc499021848)

[5.1 Résumé du rapport du TPI / version succincte de la documentation **Erreur ! Signet non défini.**](#_Toc499021849)

[5.2 Sources – Bibliographie **Erreur ! Signet non défini.**](#_Toc499021850)

[5.3 Journal de travail **Erreur ! Signet non défini.**](#_Toc499021851)

[5.4 Manuel d'Installation **Erreur ! Signet non défini.**](#_Toc499021852)

[5.5 Manuel d'Utilisation **Erreur ! Signet non défini.**](#_Toc499021853)

[5.6 Archives du projet **Erreur ! Signet non défini.**](#_Toc499021854)

*NOTE L’INTENTION DES UTILISATEURS DE CE CANEVAS:  
Toutes les parties en italiques sont là pour aider à comprendre ce qu’il faut mettre dans cette partie du document. Elles n’ont donc aucune raison d’être dans le document final.*

*De plus, en fonction du type de projet, il est tout à fait possible que certains chapitres ou paragraphes n’aient aucun sens. Dans ce cas il est recommandé de les retirer du document pour éviter de l’alourdir inutilement.*

# Analyse préliminaire

## Introduction

Réalisé dans le cadre de mon Pre-tpi, ce projet consiste en une application mobile android pour réaliser des quiz. Ce programme codé en Kotlin, langage nouveau pour moi, me permettra de m’ouvrir a plus d’options pour différents projets à venir.

## Objectifs

* L’application doit être codée en Kotlin
* L’utilisateur doit pouvoir choisir entre plusieurs thèmes donnés
* Une fois le quizz choisi il peut choisir entre plusieurs réponses sous forme d’un QCM
* A la fin du quizz l’utilisateur doit pouvoir voir son meilleur résultat et la date de passation
* Lors de l’ouverture de l’application, les quizz non effectués doivent être affichés en premier
* Le temps de réponse doit également être enregistré
* L’utilisateur doit pouvoir refaire le quizz pour voir les bonnes réponses
* Des questionnaires devraient pouvoir être affichés en ajoutant un fichier de donnée à l’application.

## Planification initiale

Les deux premières semaines seront consacrées à l’analyse du projet

Les 4 prochaines à l’implémentation

Et les deux dernières au tests et documentation

20% d’analyse

50% d’implémentation

10% de Tests

20% de documentation

# Analyse / Conception

## Concept

Voir maquette en annexe

## Stratégie de test

L’application sera testée sur 3 appareils par moi-même et sur 3 autres par :  
  
Joshua Surico

Edward Stewart

Kendra Crystalle Gueissaz

## Risques techniques

Ne maitrisant pas le langage Kotlin pour le moment le plus gros risque est un manque de compétence.  
Je travaille activement sur mon temps libre pour apprendre le langage de mon côté pour palier a ce problème.

## Planification

Semaine 1 **1er février au 2 février**

Planification initiale

Semaine 2 **12 février au 16 février**

Planification initiale

Analyse du projet

Semaine 3 **19 février au 23 février**

Création de l’architecture pour le codeCréation de la base de l’application

Ajout de plusieurs scènes

Semaine 4 **26 février au 1 mars**

Création du system de Quiz en lui-même

Ajout d’un score de fin

Lecture des Quiz par fichier XML

Ajout de Quiz dans l’application

Semaine 5 **4 mars au 8 mars**

Affichage des différents Quiz par score

Enregistrement des scores

Try des Quiz par score max

Semaine 6 **11 mars au 15 mars**

Ajout fonction pour refaire le quiz en affichant les bonnes réponses

Ajout de vision de l’historique des scores par quiz

Semaine 7 **18 mars au 23 mars**

Test du projet

Finalisation du design

Semaine 8 **25 mars au 29 mars**

Correction de bug connu

Documentation du projet

## Dossier de conception

### Choix de l’IDE

L’application sera intégralement codée depuis l’IDE Android studio Code.

C’est l’IDE officiel de google pour le développement Android. Et prend en charge nativement Kotlin ce qui me donne accès a beaucoup de fonctionnalité intéressante pour notamment le débogage.

Android studio possède également un émulateur android inclus ce qui me permettra de tester JoQuiz sur une multitude d’OS et format d’écran.

### Base de données

JoQuiz utilisera comme base de données plusieurs fichier XML. Un pour faire l’historique des scores accompli sur les différents quiz et récupèrera dans un dossiers plusieurs fichiers XML créé par l’utilisateur qui contiendront les quiz et question en eux-mêmes

### Diagramme de flux du point de vue utilisateur

Voir Document joint

### Maquette du projet

Voir Document joint

# Réalisation

## Dossier de réalisation

### Version d’Android

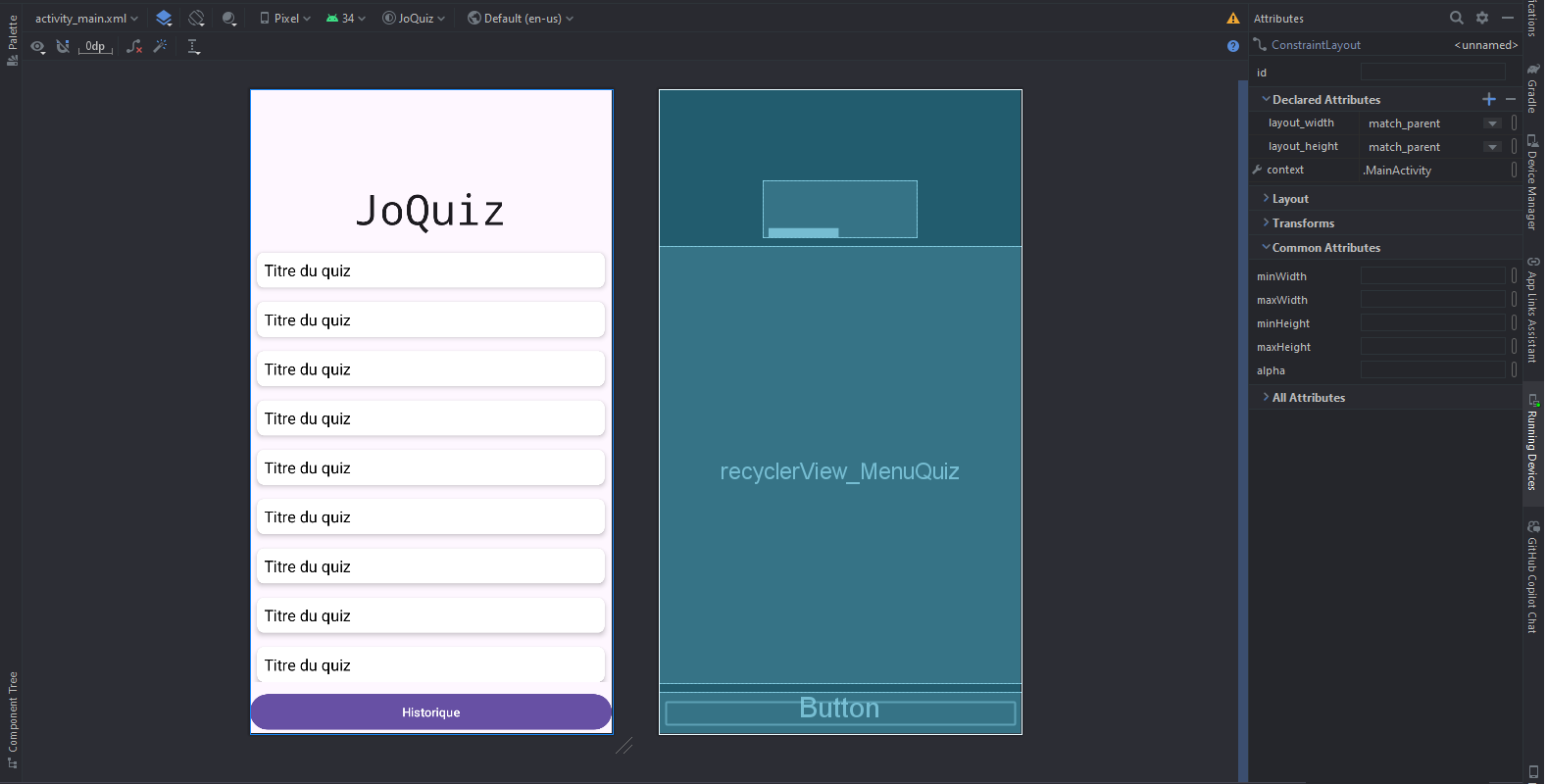
Pour créer mon projet j’ai choisi comme SDK minimum l’api 21 qui corresponds à Android 5.0 (Lollipop version).

Cela a pour but que l’application puisse fonctionner sur le maximum de téléphone android (99%).

### Création des scènes

J’ai commencé par créer toutes les scènes de mon projet afin de faciliter le développement de la suite de mon projet.  
Séparé en 4 fichiers ces « Activité » sont codé en XML grâce au mode « Design » d’Android studio et stocké dans le dossier « layout ».

* Activity\_main.xml
* Historique.xml
* Quiz.xml
* Résultat.xml



Mode layout d'android studio

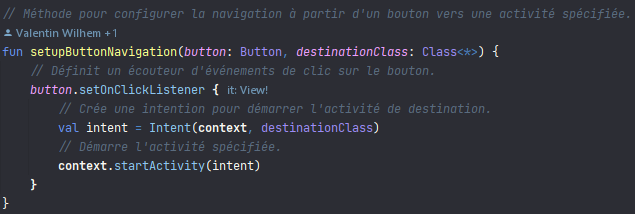
### Navigation entre différentes scènes

Afin de pouvoir naviguer entre différentes scènes a l’aide de bouton au cours du projet j’ai créé la classe NavigationHandler plutôt que de placer dans chaque classe ou on trouve de la navigation la même fonction.

Elle est composée de quatre fonctions :

* **setupButtonNavigation** : Attribue une action de navigation à un bouton spécifique, permettant le passage à une nouvelle activité définie. Cette méthode associe dynamiquement un bouton à une page cible, initiée lors de l'activation du bouton.
* **goToFirstQuiz** : Navigue vers la scène la première scène de l’activité “Question”, tout en communiquant a la classe correspondante l’id du quiz cliqué.
* **quizHandler** : Fonctionne comme goToFirstQuiz, avec l'ajout d'une variable permettant de transmettre à quel question l’utilisateur se trouve et une autre passant l'état des réponses de l'utilisateur sous forme d'un tableau de booléens (BooleanArray) afin de former à la fin les résultats du quiz.
* **goToResult** : Navigue vers la scène de résultat en communiquant notre tableau de booléens et l’id du quiz afin d’afficher les informations requises.

Cette classe est capitale car elle me permet de transmettre des variables entre différentes activité en utilisant des « Intent ». Toute autre méthode de transmission de donnée entre activité étant beaucoup trop complexe pour mes connaissance et l’ampleur du projet, j’ai donc opté pour cette option simple.



La fonction setupButton, les autres fonctions ce bases sur celle-ci.

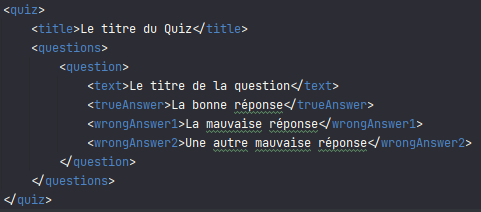
### Chargement des quiz depuis un fichier xml

Afin de gérer l’ajout de plusieurs quiz, j’ai concu la classe QuizRepository pour charger les données des quiz à partir d'un fichier XML. Cela centralise le processus de chargement et de stockage des quiz, évitant ainsi la redondance du code et facilitant la maintenance.

Voici les Methodes principales de la classe, réutilisée dans le reste du projet :

* **loadQuizzesFromXML** : Lit le fichier XML, extrait les quiz, leurs questions et réponses, puis les stocke dans une liste.
* **parseQuestion** : Analyse chaque question du fichier XML pour récupérer le texte, la bonne réponse et les mauvaises réponses.
* **getAllQuizzes** : Donne accès à la liste des quiz chargés pour être utilisés dans l'application.

Cette classe rend l'accès aux données des quiz facile et organisé pour une meilleures cohérence dans le code.



La template du fichier XML que j'utilise pour stocker mes quiz

### Système de quiz

La classe Questions est conçue pour afficher les questions de nos quiz et gérer les réponses de l'utilisateur. Cette activité permet d'interagir directement avec les quiz chargés avec QuizRepository et offre une expérience utilisateur interactive.

Éléments principaux de la classe :

* **Initialisation et récupération des données :** Dès le lancement de l'activité, elle récupère les identifiants du quiz et de la question actuelle passés via la classe NavigationHandler. Ces informations sont essentielles pour afficher le bon quiz et la bonne question à l'utilisateur.
* **Affichage des questions et des réponses** : Utilise les données du quiz sélectionné pour mettre à jour l'interface utilisateur avec le titre du quiz et le texte de la question actuelle. Les options de réponse sont également préparées et affiché sur les boutons de l’activité.
* **Gestion des interactions utilisateur** : Les boutons de réponse écoutent les clics des utilisateurs. Chaque clic sur une réponse enregistre la sélection et décide de la navigation : soit vers la question suivante du quiz, soit vers l'écran de résultats si c'est la dernière question.
* **Enregistrement des résultats** : À la fin du quiz, le score de l'utilisateur est calculé et enregistré dans la base de données. L'utilisateur est ensuite dirigé vers l'écran de résultats où son score est présenté.
* **Mélange des réponses** : Pour chaque question, les réponses sont mélangées avant d'être affichées, garantissant ainsi que l'ordre des réponses varie à chaque fois pour l'utilisateur.

La classe Questions joue un rôle crucial pour le fonctionnement des quiz en eux même, C’est ici que ce trouve la majeure partie de la logique de l’application.

### Affichage des réponses

J’ai créé pour afficher l’écran de réponse une classe basique, « Resultat », qui vas simplement afficher le résultat et le nombre de points maximal en récupérant les données des quiz via la classe NavigationHandler ainsi que trois boutons pour recommencer la partie, afficher les réponses [non implémenté] et pour retourner au menu principal.

### Création d’une base de données pour créer un historique

La gestion des scores dans notre application est assurée par une base de données locale, créée dans le fichier « ScoreResult.kt ». Grâce à l'utilisation de Room, une bibliothèque de persistance de données qui simplifie l'accès à la base de données SQLite sur Android. Cette solution offre une abstraction de la couche d'accès aux données tout en permettant une intégration fluide avec le reste de l'application.

Composants principaux de ScoreResult :

* **ScoreResult** : Une entité représentant un score. Chaque score a un identifiant unique, un titre correspondant au quiz, et un score numérique. Déclarée avec l'annotation @Entity, cette classe définit la structure de la table scores dans la base de données.
* **ScoreDao** : Un Data Access Object (DAO) fournissant les méthodes nécessaires pour insérer de nouveaux scores et récupérer tous les scores existants. Les annotations @Insert et @Query définissent respectivement les opérations d'insertion et de requête sur la base de données.
* **AppDatabase** : La base de données de l'application, étendant RoomDatabase. Elle déclare les entités présentes dans la base de données et fournit une instance du DAO pour accéder aux données. Le pattern Singleton est utilisé pour garantir qu'une seule instance de la base de données est créée à travers l'application.

Plugins utilisés en plus de ceux de base dans Android Studio :

* **Room** : Room est utilisé pour simplifier l'accès à la base de données et les opérations CRUD (Create, Read, Update, Delete). En offrant une couche d'abstraction au-dessus de SQLite, Room facilite la gestion des bases de données en vérifiant au moment de la compilation les requêtes SQL, réduisant ainsi le risque d'erreurs à l'exécution.
* **Kotlin Coroutines** : Les coroutines de Kotlin sont utilisées pour effectuer des opérations asynchrones, comme les insertions dans la base de données, sans bloquer le thread principal de l'application. Cela permet de maintenir une expérience utilisateur fluide même lors de l'accès à des données potentiellement longues à charger ou à enregistrer.

L'utilisation de Room et des coroutines de Kotlin répond à un besoin d'efficacité et de fiabilité dans la gestion des données. Room offre une intégration puissante avec Kotlin, y compris le support des coroutines, rendant le code plus concis, lisible, et moins sujet aux erreurs.

## Description des tests effectués

Afin d’assurer que l’application fonctionne correctement et qu’aucun crash ne survienne j’ai fait installer et tester JoQuiz a plusieurs personnes de mon entourage sur divers téléphone Android.  
Ayant au départ choisi de faire tester mon application a seulement trois personnes, j’ai finalement opté à faire tester JoQuiz au maximum de personnes possible pour avoir un résultat beaucoup plus précis.

**Les bugs suivants ont été relevé puis résolu dans la version finale :**

Bug de récursivité avec les recycleurs quand la largeur d’écran n’est pas conventionnelle.



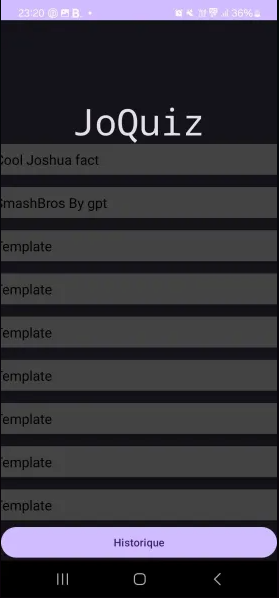
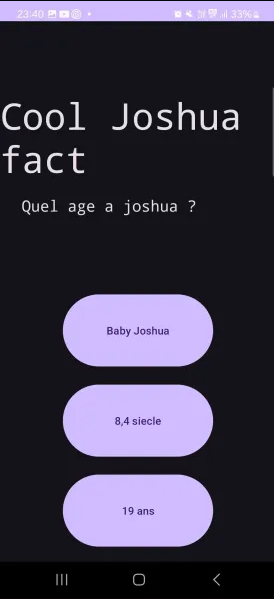


Figure 2 problèmes avec le recylcer de l'historique

Figure 2 Problèmes avec le recycler du menu principal

Problème d’affichage titre quand on lance un quiz, il devrait être centré



Propriétaire et Appareil sur lequel l’app a été testé :  
  
Samsung S23 : Fernando Chirino

Samsung S23 Ultra : Levy Stoller

Galaxy A13 : Nessim Moutaouakkil

Samsung S8+ : Joshua Surico

Vivo Y52 5G : Arthur Botteman

Samsung S24+ : Mahé Lavaud

## Erreurs restantes

Bug en mode paysage :  
  
L’application n’a pas été adaptée au mode paysage du téléphone. Voir annexe

Cela pour conséquence que si l’utilisateur tourne son téléphone pour une raison ou une autre et qu’il a le mode paysage activé l’application ne pourras pas s’afficher correctement.  
J’envisage de faire une version de mes activités spécialement pour le mode paysage, Il est également possible de simplement désactiver cette option afin d’éviter toute confusion chez l’utilisateur.  
  
Le bouton « voir réponses » ne fait rien par manque de temps.

Si j’avais eu plus de temps, je l’aurais faire retourner au début du quiz mais avec les boutons de couleurs différentes en fonctions de si le résultat est juste ou faux.  
Cela n’a pas été fait tout de suite car cela me forçait a changer la logique de ma classe « Questions » ce dont je n’ai pas eu le temps.

# Conclusions

Objectif réussi / échoué

* L’application doit être codée en Kotlin
  + Réussi
* L’utilisateur doit pouvoir choisir entre plusieurs thèmes donnés
  + Réussi
* Une fois le quizz choisi il peut choisir entre plusieurs réponses sous forme d’un QCM
  + Réussi
* A la fin du quizz l’utilisateur doit pouvoir voir son meilleur résultat et la date de passation
  + Echoué l’utilisateur a seulement un écran ou il peut voir son et le score maximal possible.
* Lors de l’ouverture de l’application, les quizz non effectués doivent être affichés en premier
  + Echoué
* Le temps de réponse doit également être enregistré
  + Echoué
* L’utilisateur doit pouvoir refaire le quizz pour voir les bonnes réponses
  + Echoué l’utilisateur peut refaire le quizz mais les bonnes réponses ne seront pas affichées
* Des questionnaires devraient pouvoir être affichés en ajoutant un fichier de donnée à l’application
  + L’utilisateur peut modifier le fichier XML qui contient tous les quiz si il vas fouiller dans les fichiers, mais une solution d’ajout de fichier n’a pas été implémenté.

Point positif en rapport avec ce projet.

Je suis très content car je me suis considérablement amélioré dans la maitrise du langage Kotlin et j’ai réussi à faire un projet fonctionnel et utilisable par n’importe quel utilisateur. Je constate le temps qu’il me faut maintenant pour coder quelque chose beaucoup plus rapide et fluide qu’au début alors que je peinais à comprendre le fonctionnement de ce langage nouveau pour moi.

Je suis également agréablement surpris du fait de devenir de plus en plus indépendant des aides et documentations, j’ai fait très attention a toujours chercher a comprendre le code que j’écrivais ou copiait de tutoriel/sources 100% et cela porte visiblement ces fruits !

Points négatifs en rapport avec ce projet.

J’ai eu beaucoup de difficulté au début du projet a apprendre le language.

J’ai perdu beaucoup de temps du a la maladie, L’armée qui me convoque la journée ou j’ai toutes mes périodes de TPI et les nombreux problèmes de trains qui m’ont fait perdre de nombreuse périodes.  
  
Cela m’a fait grandement perdre en motivation pour ce projet.

Je me suis grandement éloigné des objectifs principaux dans le but de rendre en priorité un produit qui fonctionne plutôt que rien.

# Annexes

## Sources – Bibliographie

**Android Tutorial (Kotlin) - 30 - SQLite Database Creation and Insertion**

De : CodeAndroid

<https://youtu.be/OxHNcCXnxnE?si=pR-mbGz_Gx66D086>

**How to develop app to parse XML data - Android Kotlin**

De : Dr. Parag Shukia

<https://youtu.be/bPtrRCkPPFo?si=t3ILcLP9SFtR2rpQ>

**RecyclerView With Item Click Listener in Kotlin : ( Android Tutorial 2022 )**

De : CodingSTUFF

<https://youtu.be/WqrpcWXBz14?si=Yct-5GrvYRiwS221>

**Creating a RecyclerView that handles Click Events in Android Studio (Kotlin 2020)**

De : Indently

<https://youtu.be/ai9rSGcDhyQ?si=MupczYM0hhrfGUdD>

**Kotlin Course - Tutorial for Beginners**

De : freeCodeCamp.org

<https://youtu.be/F9UC9DY-vIU?si=9Ud6biwNBrvoh6N4>

<https://www.practicalcoding.net/blog/categories/kotlin>

<https://chat.openai.com/>

## Liste des documents fournis

* Dossier de projet
* Journal de travail
* Manuel d’installation
* Manuel d’utilisation