BabelBlock

Traducteur
automatique

Projet réalisé par Nicolhas DELVOYE et Alexis LE GAL

dans le cadre du module Android

ENSSAT 2020

Sommaire

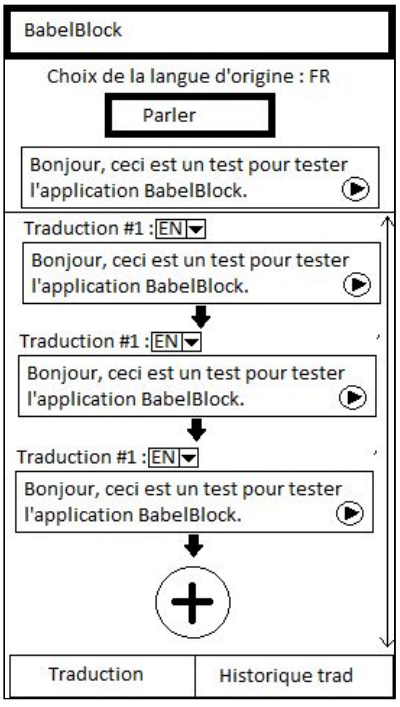
Introduction

Le but du projet est de réaliser une application Android en Kotlin. L'application devra permettre d'effectuer, à partir d'un message audio dans une langue, une traduction et une restitution audio vers une autre langue.

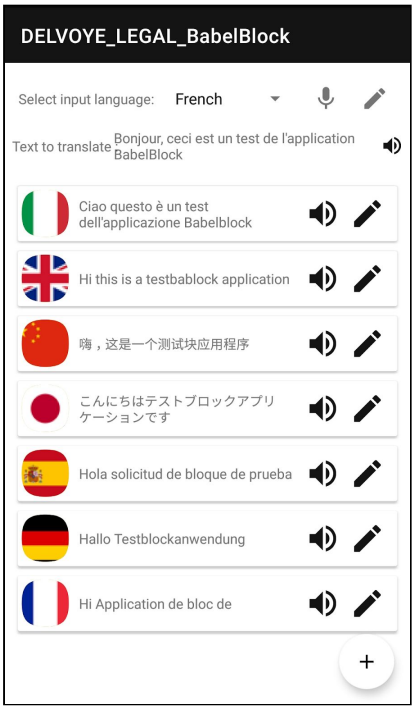
Interface utilisateur

Après analyse du sujet, nous avons réalisé une première maquette qui tend à ressembler à l'application réalisée.

Maquette

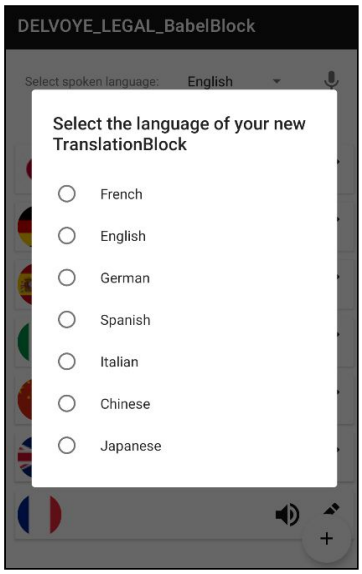


Réalisé



L'écran principal de l'application utilise un `ConstraintLayout` avec différents blocs organisés. Certains éléments sont disposés en fonction des autres éléments dans le but de les placer sur l'écran. Ainsi, l'application est aussi bien fonctionnelle dans un format portrait que dans un format paysage.

On peut notamment retrouver une `Toolbar`, un bloc "Input de la chaîne de traduction", un bloc "Texte à traduire" mais également la `RecyclerView` contenant les différents `TranslationBlock`.



Enfin, on retrouve un `FloatingActionButton`. Lors d'un clic sur ce dernier, un `MaterialDialog` permettant de choisir la langue du nouveau bloc de traduction (`listItemsSingleChoice`) s'ouvre. Après sélection d'une langue, un nouveau bloc de translation est inséré en base de données.

Services mis en place et fonctionnement

Afin de réaliser le projet, nous avons mis en place les services suivants (à l'aide des TPs mis à disposition) :

- SpeechRecognizer
- Translator
- TextToSpeech

SpeechRecognizer

Afin que l'utilisateur puisse choisir la langue de départ, nous avons créé un Intent `RecognizerIntent.ACTION_RECOGNIZE_SPEECH` permettant de reconnaître la langue sélectionnée.

Les paramètres du Intent sont les suivants :

- EXTRA_LANGUAGE_MODEL : `LANGUAGE_MODEL_FREE_FORM`
- EXTRA_LANGUAGE : `locale.toString()`
- EXTRA_LANGUAGE_PREFERENCE : `locale.toString()`
- EXTRA_ONLY_RETURN_LANGUAGE_PREFERENCE : `locale.toString()`

Lors d'un appui sur le bouton "microphone", l'Intent démarre. Dès lors que le service considère que la prononciation est terminée, la phrase est affichée à l'écran et la traduction chaînée dans les différents blocs débute.

Si le service ne détecte pas de phrase, une erreur est affichée dans une SnackBar :

Translator

La traduction s'effectue à l'aide de MLKit, bibliothèque de Machine Learning pour mobile proposée par Google.

Afin de chaîner les traductions, nous avons dû adapter le code fourni afin de rendre séquentiel le travail de traduction. En effet, un bloc ne peut être traduit que si le bloc précédent l'a été.

Pour cela, nous avons remplacé la callback par un objet retourné de type `Deferred<String>` (sur lequel on peut appeler une méthode `.await()`). Cela nous permet d'attendre la fin de la traduction pour effectuer la traduction qui suit.

Cette adaptation nous oblige à exécuter le code dans une **Coroutine**, la tâche sera donc lancée en parallèle de notre activité principale.

TextToSpeech

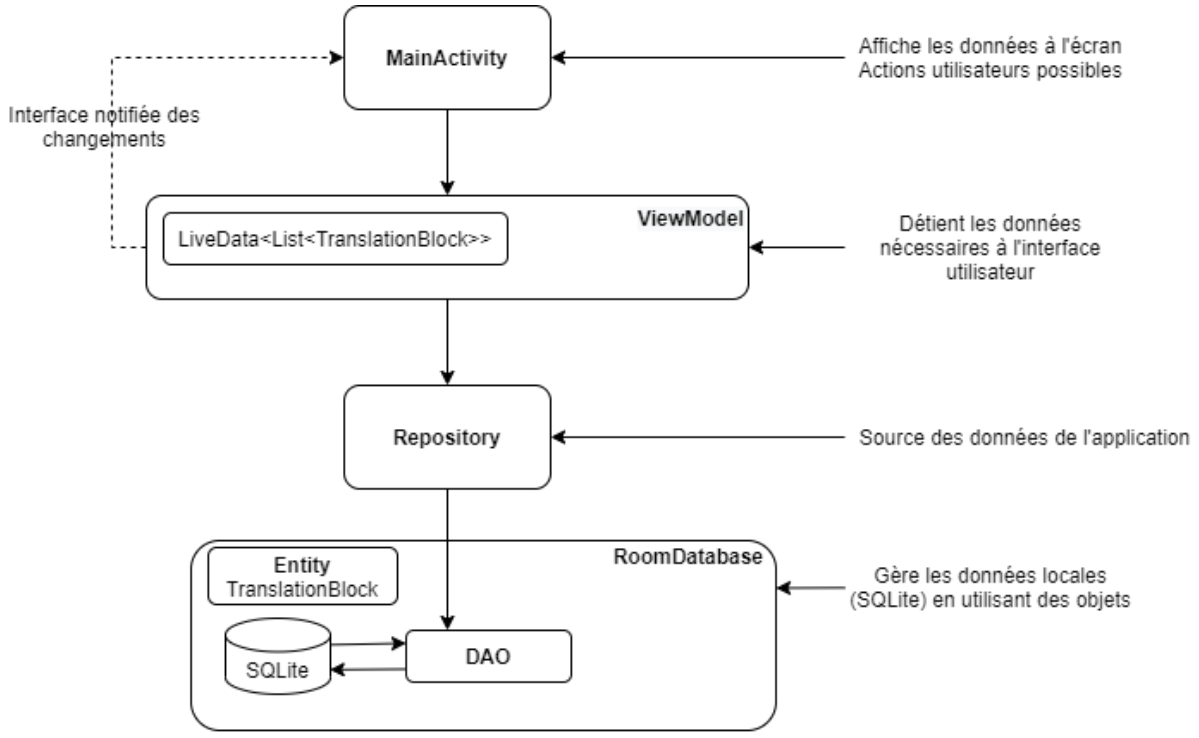
La transformation du texte vers un signal audio est réalisée par le `TTS_SERVICE` qui est situé dans le package `speech.tts` d'Android.

Le service est appelé lors d'un clic sur un des boutons "haut parleur". Une `suspendCoroutine` est alors lancée.

Architecture fonctionnelle

L'application est constituée d'une unique activité : `MainActivity`. Cette activité possède un `RecyclerView` qui contient les données de la base de données.

Les données sont reliées à l'aide d'un modèle `MVVM : Model View ViewModel`.



Installation de l'application

Afin de tester notre application, assurez-vous de posséder un téléphone (ou un émulateur) sous Android 4.4 ou supérieur (Min SDK version = 19).

APK

Vous avez la possibilité d'installer notre application via l'APK disponible sur notre GitHub à l'adresse suivante : **https://github.com/ndelvoye/BabelBlock.git**

Android Studio

Vous pouvez également tester notre application à l'aide d'Android Studio dans sa dernière version.

Sur l'écran de lancement de l'application, cliquez sur **Get from Version Control** et entrez l'URL suivante : **https://github.com/ndelvoye/BabelBlock.git**

Une fois le projet importé, vous pouvez build le projet et le démarrer sur votre appareil/émulateur.

Tests réalisés

Tests unitaires

Nous n'avons mis en place qu'un squelette de tests unitaires à l'aide de JUnit par manque de temps consacré à cette tâche.

Tests utilisateurs

Cependant, nous avons cherché à tester intégralement l'application à la main afin de nous assurer qu'elle ne contenait pas de bugs (ou très peu).

Possibles améliorations

Le projet dispose encore de plusieurs pistes d'amélioration que nous aurions pu réaliser dans le cas où nous aurions eu plus de temps pour développer l'application.

WorkManager

Nous aurions pu gérer un WorkManager permettant de traduire un bloc à l'aide du bloc précédent. Cela nous aurait notamment permis de stopper la chaîne de traduction en plein milieu de son exécution.

Drag & drop amélioré

Nous aurions pu améliorer le drag & drop afin de déplacer un `TranslationBlock` de + d'une place à la fois. La cause de ce point négatif sur l'application est intervenue lors de l'implémentation du modèle `MVVM`. En effet, à chaque update d'un `TranslationBlock` en base de données, le `RecyclerView` est actualisé (car le listener `observe` sur les données `LiveData` détecte des changements).

Gestionnaire d'historique des traduction

Nous aurions pu ajouter une table contenant l'historique des différentes traductions réalisées par l'utilisateur. Cet historique aurait pu être consultable depuis une autre activité (accessible via un menu en bas de l'application ou dans la toolbar).

Détection automatique de la langue

Nous aurions pu mettre en place le choix **Automatically detected** dans le Spinner. Cela aurait permis de détecter la langue de la phrase en entrée de la chaîne de traduction au lieu d'obliger l'utilisateur à choisir une langue.

Cela aurait notamment pu être utile dans le cas où l'utilisateur ne sait pas de quelle langue il s'agit.

Tests unitaires

Dans l'état actuel des choses, notre application a été intégralement testée à la main. Le fait de rajouter des tests unitaires nous permettrait, par exemple, de refactor certaines features fonctionnelles actuellement (exemple : la chaîne de traduction) sans casser l'intégralité de l'application.

Gestion de projet

Dans le but de travailler en binôme, nous avons mis en place deux outils primordiaux quant à la gestion de ce projet.

GitHub

Nous avons pu mettre en place un repository GitHub. Sur celui-ci, nous avons essayé de créer des branches dès que nos travaux pouvaient entrer en collision. À la fin du développement d'une branche, nous effectuions une relecture du code afin de la merge sur la branche **master**.

Trello

Nous avons également pu mettre en place un tableau Trello afin d'indiquer les différentes tâches à réaliser.

Conclusion

Ce projet nous a permis de développer une application Android en Kotlin de A à Z. En effet, nous avons pu:

- Concevoir une interface (notamment à l'aide de `ConstraintLayout`)
- Mettre en place différents services (reconnaissance vocale, traduction à l'aide de MLKit ou encore TextToSpeech)
- Réaliser une persistance des données à l'aide du modèle `MVVM` (`Model View ViewModel`)
- Réaliser différents tests unitaires à l'aide de `JUnit`
- Réaliser des traitements asynchrones à l'aide de coroutines