
Projet d'Application Mobile

ADVANCED MOBILE APPLICATIONS

May 30, 2022

University of Applied Sciences and Arts Western Switzerland
Computer Science

Rosset Denis, Rémy Macherel

1 Résumé

Ce rapport décrit le projet réalisé durant le cours MA-AdMoApp du master MES HES-SO. Il était requis pour cette application d'être context-aware et doit employer au moins un capteur du téléphone. Toute cette application est développée à l'aide du Framework Flutter et programmé en langage Dart.

2 Introduction et but de l'application

Comme décrit dans le cahier des charges (voir en annexe) nous avons dans le cadre du cours MA-AdMoApp conçu une application permettant d'accorder un instrument de musique. L'application se compose de trois vues:

- La page d'accordage
- La page de statistiques
- La page de paramètres

Cette application fut développée en utilisant les connaissances vues dans le cours et le langage Dart.

3 Descriptif des pages

3.1 Page d'accordage

La page d'accordage permet, comme son nom l'indique d'enregistrer via le capteur audio du téléphone le son émis par l'instrument et d'afficher la note correspondante. Elle affiche également un graphique live sur lequel sont tracées les courbes des notes jouées. Une fois l'accordage terminé et satisfaisant, l'utilisateur a la possibilité d'enregistrer les statistiques de l'accordage qui comprennent :

- La date et l'heure de l'accordage
- La durée de l'accordage
- La localisation (si les permissions de localisation sont accordées)
- Le graphique des notes jouées

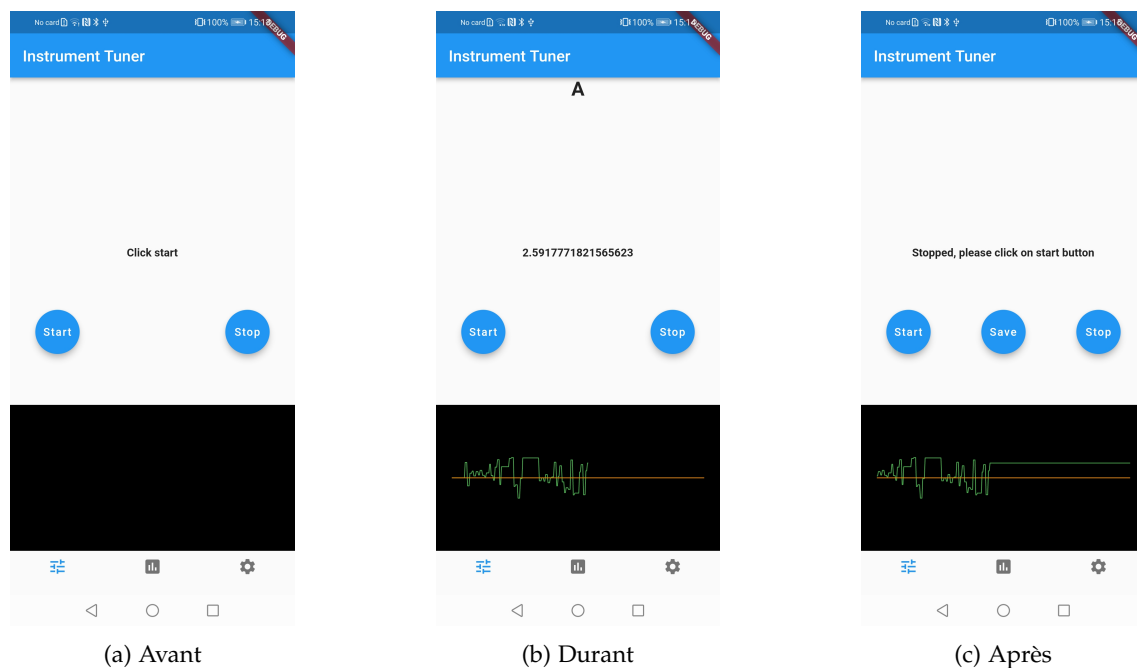


Figure 1: Différents états de la page tuner

On peut observer sur les figures ci-dessus les différents états de la page d'accordage. La première montre l'état dit de repos (avant le lancement de l'accordage), la deuxième montre l'état durant l'accordage avec l'affichage de la note jouée, la différence de fréquence entre la note ciblée et la note jouée, ainsi qu'un graphique live des notes jouées.

Sur la dernière figure une illustration de l'état lorsque le bouton stop a été pressé. On observe que le graph reste affiché et que le bouton permettant de sauvegarder la statistique apparaît. Celui-ci redisparaît lors d'un appui sur start.

3.2 Page de stats

Toutes les statistiques enregistrées sont disponibles dans la page des Stats sous forme d'une liste avec pour chaque statistique sa date et sa durée.

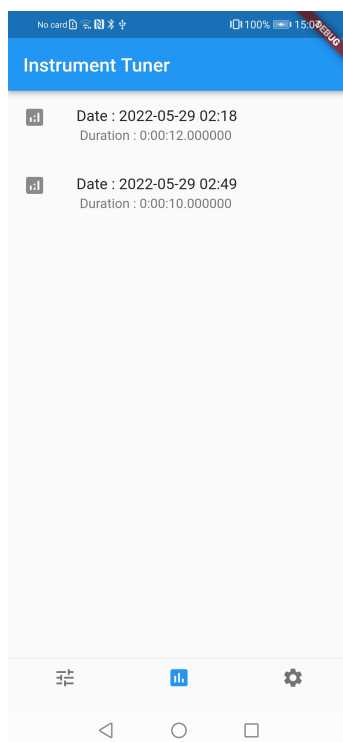


Figure 2: Page de statistiques

Un clic (touche) sur une des statistiques permet d'en afficher les détails. Si une localisation a été enregistrée pour la statistique, l'adresse sera écrite sur la page de détails.

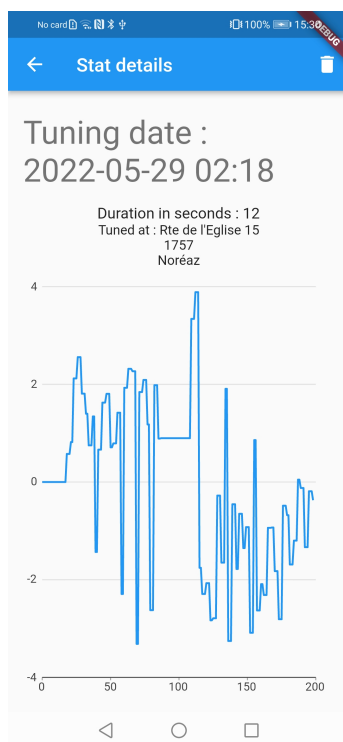


Figure 3: Affichage des détails d'une statistique d'accordage

Sur cette page existe également une option pour supprimer la statistique de la liste de celles enregistrées.

3.3 Page de paramètres

Dans l'accordage d'un instrument il existe deux paramètres important afin de réaliser celui-ci, ces paramètres sont:

- Le type d'instrument
- La fréquence de référence d'accordage

Afin de définir ces paramètres, nous avons créé une page permettant de les définir.

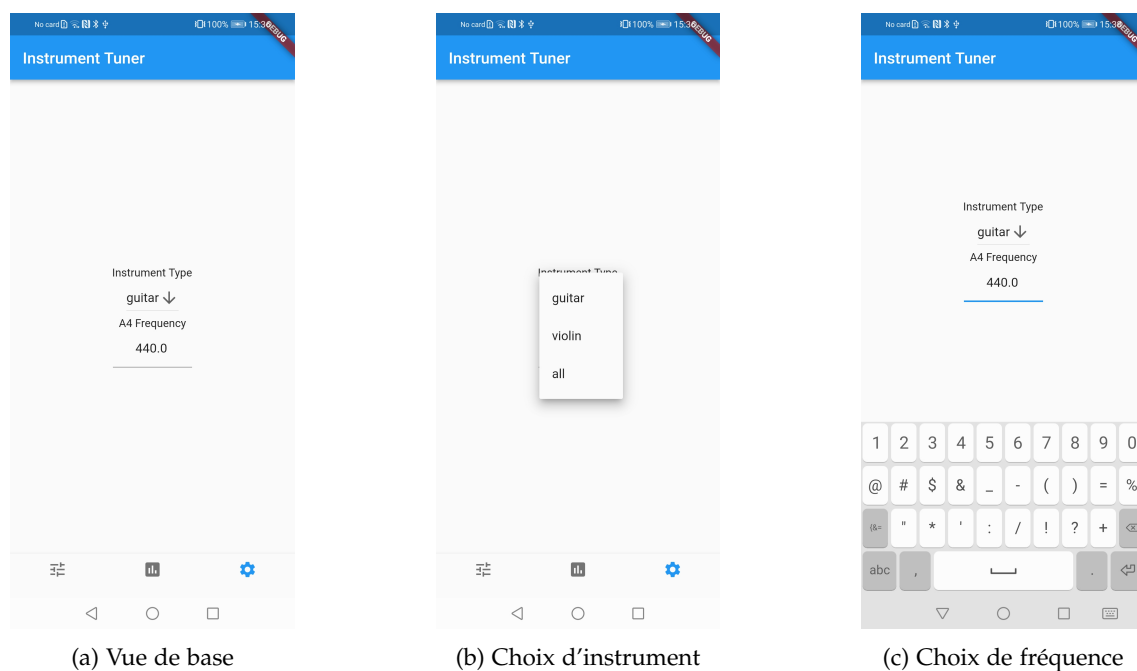


Figure 4: Différents états de la page settings

Pour le choix de la fréquence nous avons utilisé les modes de validation des widgets flutter afin de s'assurer que seuls des nombres peuvent être inscrits dans ce champ.

4 Ingénierie logicielle

4.1 FlowChart de la page tuner

Le fonctionnement de la page tuner est celui décrit dans le flowchart de la figure suivante. Lorsque l'on appuie sur start, l'outil de capture audio démarre et enregistre les données du micro du téléphone. Le son capturé est ensuite analysé par la librairie *PitchUp Dart* (voir plus loin) afin d'en ressortir la note jouée ainsi que la différence de fréquence avec la note de référence. Ces valeurs sont ensuite affichées sur l'écran et également ajoutées au graphique. Ce processus tourne ainsi jusqu'au moment où l'on appuie sur le bouton stop. Cet appui a pour effet de stopper tout enregistrement et afficher le bouton permettant la sauvegarde des statistiques. En cas d'appui sur le bouton de sauvegarde, les données statistiques de l'accordage sont enregistrées dans une base de données Hive.

Pour les autres pages le fonctionnement est plus simple car en effet pour la page stats il s'agit d'afficher la liste des statistiques enregistrées et donner la possibilité de voir les détails ou supprimer une statistique. Pour ce qui est de la page des paramètres on ne peut que modifier les valeurs de ceux-ci.

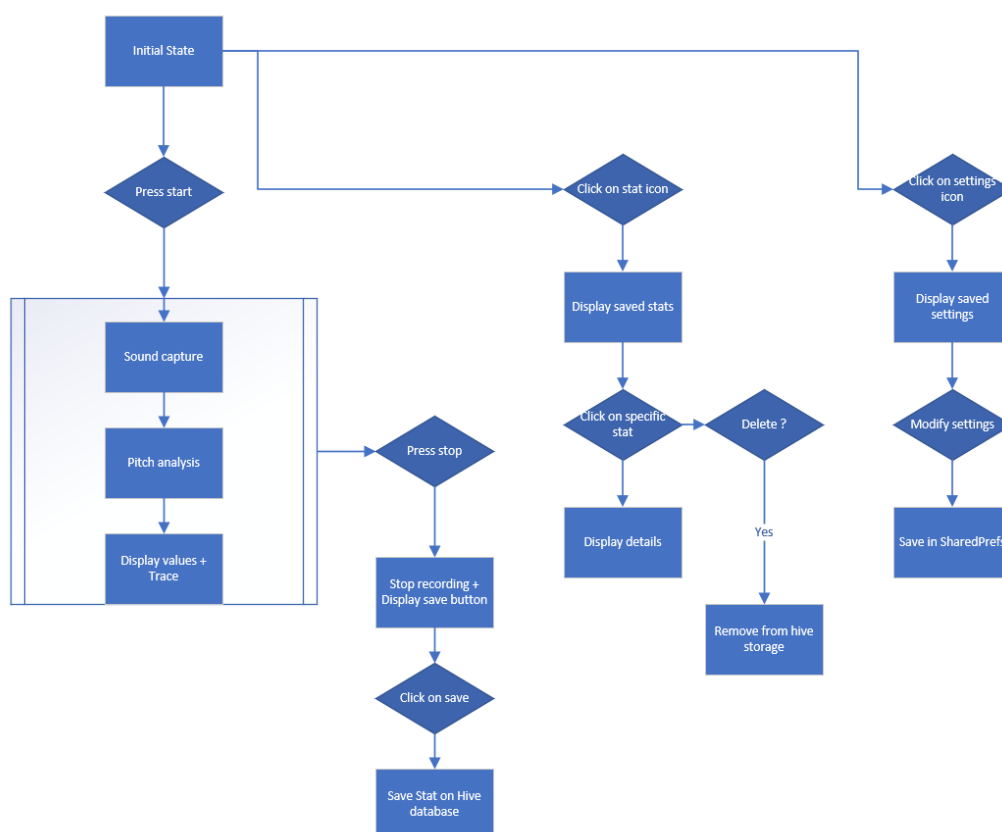


Figure 5: Tuner Page Flowchart

4.2 Architecture

Pour structurer notre application, nous avons utilisé le principe des layers. Nous avons donc séparé selon le principe les données, les domaines et l'affichage.

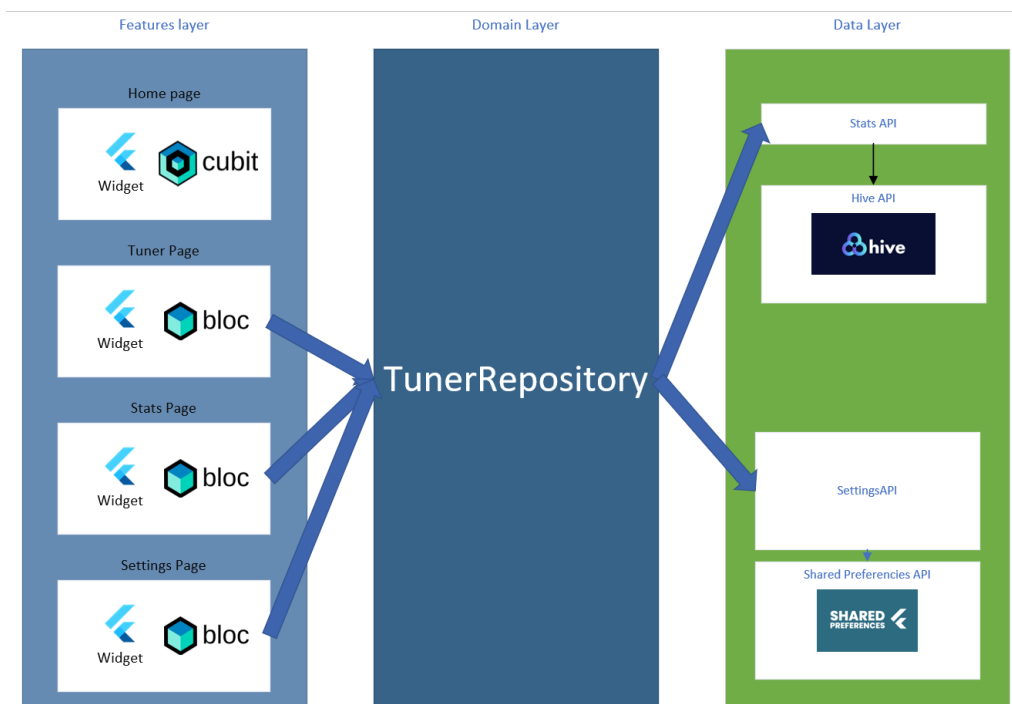


Figure 6: Application layers

4.3 Technologies utilisées

4.3.1 Base de donnée Hive

Pour le stockage des statistiques nous avons utilisé une base de données de type *Hive* comme vu en cours. Hive est une base de données purement écrite en langage Dart et est rapide et légère. Cette technologie possède également un API puissant et intuitif permettant de l'utiliser assez facilement. Hive est basé sur un concept de Box ou chaque Box peut être vue comme une table SQL classique. Dans notre cas, une box pour la classe *TunerStats* est ouverte afin de stocker les stats. La classe elle-même fût développée en respectant les critères de Hive (voir cours Flutter-Storage p.26) afin qu'un *TypeAdapter* (Sorte de modèle de traduction de la classe pour la base de données Hive) puisse être généré avec la commande `flutter pub run build_runner build`. Nous avons dû implémenter nos propres *TypeAdapter* pour le type durée car ce types de données n'est pas supporté par Hive.


```
/// Adapter for duration in seconds
class DurationAdapter extends TypeAdapter<Duration>{
    @override
    int typeId = 5;
    @override
    Duration read(BinaryReader reader){
        return Duration(seconds: reader.read());
    }

    @override
    void write(BinaryWriter writer, Duration obj){
        writer.write(obj.inSeconds);
    }
}
```

Figure 7: Duration TypeAdapter

4.3.2 Stockage SharedPreferences

Pour ce qui est des paramètres d'accordage, nous avons décidé d'utiliser les *SharedPreferences*, d'une part afin d'expérimenter une deuxième technique de stockage vue en cours, et d'autre part car ce sont celles-ci qui sont recommandées dans le cas de stockage persistant de données simples comme par exemple des paramètres. Le défaut dans notre cas était que ce type de stockage ne permet de stocker que des données de type int, double, bool, string, List<String> et dans notre cas le type d'instrument était non compris, nous avons donc dû traiter différemment et stocker ceci sous forme de String pour ensuite à la lecture caster en *InstrumentType*.

4.3.3 Bloc Provider

Pour la gestion des états des paramètres, des statistiques ainsi que de la logique de la page de l'accordeur, nous avons utilisé le système des *Bloc Provider* également vu en cours. Un Bloc est une classe basée sur des événements afin de déclencher des changements d'états. Pour ce qui est des statistiques, nous avons donc défini quatre événements différents afin de gérer les différentes situations possibles lesquelles sont :

- Demande d'abonnement
- Edition de la statistique
- Suppression de la statistique
- Chargement du détail de la statistique

Ces quatre états nous permettent donc de gérer les états relatifs aux statistiques. Le chargement du détail a été implémenté car le stockage de la position est effectué à l'aide d'un *GeoPoint* (voir doc), et donc lorsque l'on veut récupérer l'adresse nous devons utiliser le package *geocoding* (voir doc) et plus particulièrement la fonction *placemarkFromCoordinates* permettant de récupérer une adresse sur la base des coordonnées latitude et longitude. La fonction permettant de récupérer l'adresse est une fonction de type *async* et son exécution prend un petit temps et nous avons donc du passer par un état intermédiaire afin de récupérer cette adresse. Nous aurions également pu stocker uniquement l'adresse directement sous forme de String ce qui aurait évité de devoir coder un *TypeAdapter* pour stocker un *GeoPoint* dans Hive, mais la variété de propriétés disponibles dans un *GeoPoint* ainsi que la potentielle possibilité future d'afficher le point sur une carte nous semblait plus intéressant.

Pour ce qui est des paramètres, nous n'avons eu à implémenter que deux événements qui sont :

- Demande d'abonnement
- Paramètres edités

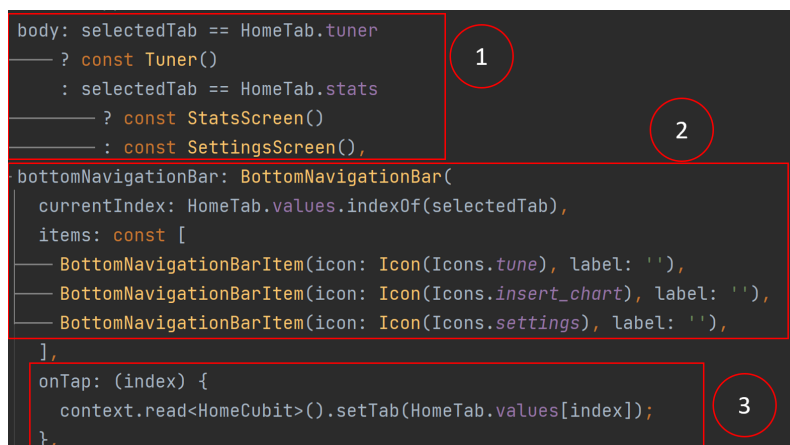
Ces deux événements permettent donc de charger les paramètres enregistrés ainsi que de modifier ceux-ci lorsqu'ils sont édités.

Pour ce qui est de l'accordeur, il y a plus d'événements possible car son fonctionnement est bien plus complexe.

- Demande d'abonnement
- Demande de permissions pour le microphone
- Démarrage de l'accordage
- Arrêt de l'accordage
- Rafraichissement du live chart
- Rafraichissement des textes affichés

4.3.4 Cubit provider

Nous avons utilisé la technologie du Cubit Provider pour la gestion des changements de page. Ce provider est plus simple que le *Bloc* et permettait d'implémenter facilement la gestion des pages. Il suffit en effet de déclarer un état du cubit pour chaque page possible (dans notre cas trois) puis lors d'un changement de modifier le state du cubit afin qu'il recharge la page souhaitée.



```
body: selectedTab == HomeTab.tuner
  ? const Tuner()
  : selectedTab == HomeTab.stats
  ? const StatsScreen()
  : const SettingsScreen(),

bottomNavigationBar: BottomNavigationBar(
  currentIndex: HomeTab.values.indexOf(selectedTab),
  items: const [
    BottomNavigationBarItem(icon: Icon(Icons.tune), label: ''),
    BottomNavigationBarItem(icon: Icon(Icons.insert_chart), label: ''),
    BottomNavigationBarItem(icon: Icon(Icons.settings), label: ''),
  ],
  onTap: (index) {
    context.read<HomeCubit>().setTab(HomeTab.values[index]);
  },
```

Figure 8: Utilisation du Cubit Home

On peut observer sur la figure 8 trois parties de l'utilisation du Cubit :

1. Affichage de la page souhaitée en fonction de l'état du Cubit
2. Affichage dans une barre de navigation des possibilités d'écrans
3. Changement de l'état du Cubit (et donc de la page) en fonction d'une touche sur un des icônes.

4.3.5 Repository

Nous avons utilisé comme layer de domain un Repository permettant l'accès aux api de stockages des données pour chaque widget en ayant la nécessité.

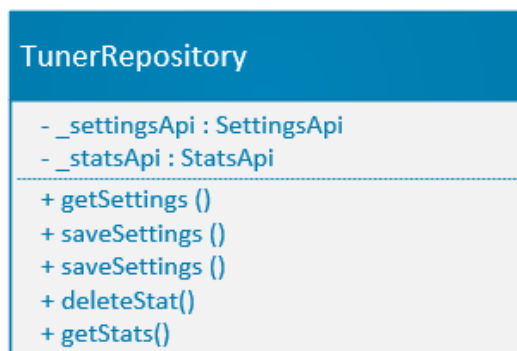


Figure 9: Diagramme UML de la classe TunerRepository

4.3.6 Librairie externe PitchUpDart et PitchDetector

Afin de calculer le *pitch* (hauteur de la note) sur la base des données capturées par le capteur audio du smartphone, nous avons utilisé deux bibliothèques externes développées par *TechPotatoes* qui permettent ce calcul et nous évitaient ainsi de devoir effectuer les calculs nous-même (incluant transformée de fourrier, analyse de signal, etc.).

Dans les codes sources initiaux de la librairie, un seul instrument (guitare) était défini, nous avons donc décidé d'ajouter directement les codes sources de cette librairie dans notre projet afin de pouvoir les modifier et ajouter des instruments. Les codes sources se trouvent donc dans un dossier nommé *pitchDetector_lib* avec un *readme.txt* indiquant les liens menant aux codes sources initiaux et mentionnant la source pour des raisons de copyright. (les liens sont les suivants : *pitchDetectorDart* et *pitchUpDart*).

Nous avons donc effectué pour seules modifications l'ajout de nouveaux instruments. Pour cette librairie il est important de préciser pour chaque instrument la hauteur minimale et maximale de la note possible ainsi que le "nom" de ces notes sous forme d'une liste de Strings.

```
case InstrumentType.violin:
    _minimumPitch = 196.0;
    _maximumPitch = 2637.0;
    _noteStrings = [
        "C",
        "C#",
        "D",
        "D#",
        "E",
        "F",
        "F#",
        "G",
        "G#",
        "A",
        "A#",
        "B"
    ];
    break;
```

Figure 10: Déclaration du type violon

La librairie nous fournit un accès à la différence entre la note jouée actuellement et la note de référence de l'application. Cette différence est un chiffre entre -10 et +10 en fonction d'à quel point la note jouée est loin de la note de référence.

Graphique

Les valeurs fournies permettent de dessiner un graphique grâce à la librairie oscilloscope.

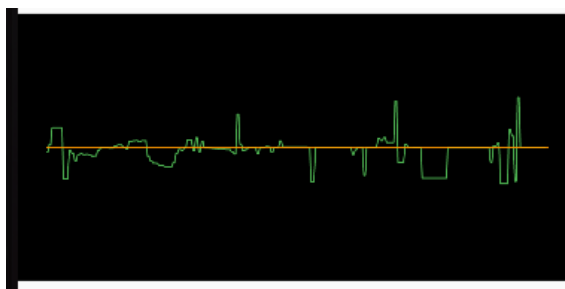


Figure 11: Graphique généré par l'application lorsqu'on accorde un instrument

Ce graphique indique en vert ce qui est enregistré par le microphone et en jaune la "note de référence". Vu que l'accordeur donne directement la distance avec la note, on a juste à utiliser

cette valeur avec son intervalle de temps pour pouvoir l'afficher sur le graph et avoir un système très intuitif à utiliser.

4.3.7 Geolocalisation

Afin d'ajouter une possibilité de géolocalisation à la sauvegarde de nos statistiques, nous avons utilisé le package Geolocator. Dans un sens, il nous permet de vérifier les permissions d'accès à la localisation et, si besoin de les demander, puis il nous sert à récupérer les coordonnées actuelles du smartphone (dans le cas où les permissions ont été accordées). Afin de stocker ceci dans Hive, nous ne gardons que la latitude et la longitude sous forme de double. Lors de l'affichage ensuite du détail de la stat nous utilisons le package GeoCoding afin de récupérer l'adresse sur la base des coordonnées. L'utilisation de deux librairies différentes vient de notre part car nous souhaitons expérimenter plusieurs possibilités.

4.3.8 Multi-langue

Pour l'implémentation du multi-langue, nous avons utilisé la librairie localization. Cette librairie permet de récupérer la valeur de la langue du téléphone et d'afficher le contenu de l'application traduit au préalable.

```
static const _localizedValues = <String, Map<String, String>>{  
  'en': {  
    'title': 'Instrument Tuner',  
    'stats': 'Stats',  
    'settings': 'Settings',  
  },  
  'fr': {  
    'title': 'Accordeur d\'instrument',  
    'stats': 'Statistiques',  
    'settings': 'Paramètres',  
  },  
};
```

Les traductions sont faites dans un dictionnaire contenant les langues contenant elles-mêmes un dictionnaire avec chaque mot à traduire. Ensuite, pour pouvoir accéder à la traduction du titre par exemple, il suffit d'appeler :

```
LocalizationTraductions.of(context).title
```

Le contexte est nécessaire à chaque fois si on veut que l'application puisse être traduite sans devoir redémarrer l'application lorsque la langue du téléphone change.


5 Protection des données

Toutes les données de notre application sont stockées dans des bases de données locales au téléphone. Nous n'avons ainsi pas de problèmes à respecter les normes et attributs éthiques de la protection des données. De plus, nous ne manipulons quasiment aucune donnée sensible (mis à part éventuellement la localisation, mais qui peut être refusée par l'utilisateur).


6 Personas

Nous avons dans notre cas défini deux personas qui symbolisent les deux types d'utilisateurs que nous verrons pour utiliser notre application. Ces deux profils sont bien différents et nous

ont permis de designer notre application afin d'essayer de satisfaire au mieux les deux.

<div>CHARLES</div> <div>Professional musician</div> 	<div>Behaviors</div> <div>Extrovert Smart Likes to talk about music Always getting out</div>
<div>Demographics</div> <div>52 y.o. Divorced 3 children Lives in a small village Bachelor of art in Music</div>	<div>Needs and goals</div> <div>Reliability Accuracy Early adopter Statistics</div>

(a) Persona 1 : Charles

<div>JANE</div> <div>Amateur musician</div> 	<div>Behaviors</div> <div>Shy Spend a lot of time home Tech friendly Daily uses a computer/smartphone</div>
<div>Demographics</div> <div>27 y.o. Lives alone 2 cats Small appartement in big city Librarian</div>	<div>Needs and goals</div> <div>Tune her guitar No stats Simple and accessible app No expenses</div>

(b) Persona 2 : Jane

Figure 12: Personas

note : les images proviennent de <https://this-person-does-not-exist.com/en>

Ces deux profils très différents nous ont permis de nous rendre compte de la variété de personnes pouvant être intéressées par notre application. Dans les deux cas, elle doit répondre aux critères de chacun

7 Tests utilisateurs des fonctionnalités

Pour valider le bon fonctionnement de l'application, nous avons effectué des tests utilisateurs. Les premiers tests sont des tests des différents fonctionnalités que nous avons mis en place. Les seconds tests sont les tests SUS vu en cours. Pour la réalisation de ces tests, nous avons contacté quelques uns de nos amis qui ont un instrument de musique (particulièrement guitare) et nous leur avons demandé de l'accorder en utilisant cette application. Puis de répondre à un questionnaire et de nous faire un feedback sur leurs impressions.

7.1 Test des fonctionnalités

Pour ce test, nous avons demandé aux utilisateurs de tester les fonctionnalités listées ci-dessous puis nous avons noté sur 5 (1 = a eu beaucoup de mal, 5 a eu beaucoup de facilité) en fonction de l'aisance avec laquelle ils ont réalisés les tâches.

Test utilisateur des fonctionnalités :	Personne 1	Personne 2	Personne 3
Démarrer l'accordeur	5	5	5
Accepter les permissions gps + microphone	5	3	4
Accorder un instrument	5	4	2
Changer les settings	4	5	4
Sauvegarder les statistiques	3	3	4
Sélectionner une statistique	5	5	5
Lire les statistiques	5	5	5
<i>Total</i>	37	35	33

Figure 13: Tableau récapitulatif des tests utilisateurs des fonctionnalités

7.2 Test SUS

Dans le domaine de l'ingénierie des systèmes, l'échelle d'utilisabilité des systèmes (SUS) est une échelle de Likert simple, à dix points d'attitude, qui donne une vue d'ensemble des évaluations subjectives de l'utilisabilité. Elle a été développée par John Brooke chez Digital Equipment Corporation au Royaume-Uni en 1986 comme un outil à utiliser dans l'ingénierie de l'utilisabilité des systèmes électroniques.

	Personne 1 (Fais souvent de la musique)	Personne 2 (fais un peu de musique)	Personne 3 (ne fais pas de musique)
1. I think that I would like to use this system frequently.	4	3	1
2. I found the system unnecessarily complex.	1	2	2
3. I thought the system was easy to use.	5	5	2
4. I think I would need some assistance to be able to use this system.	1	3	5
5. I found the various functions in this system were well integrated	4	4	2
6. I thought there was too much inconsistency in this system	2	2	2
7. I would imagine that most people would learn to use this system very quickly	5	5	3
8. I found the system very cumbersome to use	2	2	2
9. I felt very confident using the system	5	5	4
10. I feel like the system could do with some improving to make it a more enjoyable experience	4	3	3
SUS score	82.5	75	45

Figure 14: Tableau des résultats des utilisateurs System Usability Scale

7.3 Commentaires des utilisateurs

7.3.1 Personne 1

Étant percussionniste et plus spécifiquement timbalier, j'ai souvent besoin de savoir à quel point mes instruments se désaccordent en fonction de l'endroit où ils sont entreposés. Pour moi, cette application réalise totalement ce que j'aimerais qu'elle réalise. Bien que je n'aie pas forcément besoin de tous les types d'instruments (les 12 demi-tons me suffisent) les fonctionnalités de l'application restent très utiles. Les statistiques sont vraiment intéressantes et me permettent de garder un suivi.

7.3.2 Personne 2

J'ai utilisé l'application pour accorder ma guitare et elle a très bien fonctionné. Le graphique permet de voir en temps réel à quel point on est proche ou non de la note correcte. Un défilement des notes à la place de juste les avoir affichés lorsqu'elles sont jouées pourrait être pratique pour savoir à quel point on est loin de son but. Les statistiques et les options ne me serviraient pas dans la vie de tous les jours mais j'imagine qu'elles peuvent servir pour des musiciens plus expérimentés.

7.3.3 Personne 3

C'était la première fois pour moi que j'accordais une guitare. Le graphique m'a beaucoup aidé à me rendre compte de ce que je devais faire. Le système de lettres m'a paru un peu obscure. En effet je n'ai jamais réellement fait de solfège et donc la note sous forme de texte ne me parle que très peu. Peut être qu'un autre système pourrait être plus facile à utiliser.

8 Conclusion

Ce projet a permis de mettre en pratique tous ce que nous avons vu en cours tout en nous laissant réaliser une application de notre choix. Cette manière de proposer les projets est très motivante et permet de s'impliquer d'avantage. Nous avons pu réutiliser toutes les notions vues en cours également en se basant sur les exemples fournis pour chaque technologie.

Le fait d'utiliser les patterns de Bloc Provider furent initialement compliqués pour nous, en effet nous avons rencontré pas mal de difficultés à les mettre en place. Cependant une fois ceci fait, nous avons pu remarquer que l'ajout d'une fonctionnalité (par des évènements et états) s'en trouvait grandement facilité.

Nous avons été relativement satisfaits par le score SUS obtenu. En effet le résultat pour une personne ne pratiquant pas de musique était passablement faible. Cependant cela confirme aussi nos observations du début et notre volonté de fournir une application pour des personnes à l'aise ou non dans la musique mais tout de même ayant un intérêt pour celle-ci et la pratiquant. Le projet à été très intéressant à réaliser et a permis de bien comprendre comment les applications mobiles fonctionnaient via flutter. Nous pouvons maintenant réaliser des applications complexes destinées à différentes plates-formes (android, apple, windows, etc.) ce qui permet d'être très versatile dans le domaine du développement logiciel.