



Ecole Polytechnique de l'Université François Rabelais de Tours

Département Informatique

64 Avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

[polytech.univ-tours.fr](http://polytech.univ-tours.fr)



Projet Recherche & Développement

2019-2020

## Caractérisation de la voix

Entreprise

[CHU Bretonneau & Polytech Tours](#)

Etudiant

[Steven MARTIN \(DI5\)](#)



Tuteur académique

[Pascal MAKRIS](#)

# Liste des intervenants

## Entreprises

Polytech Tours 64 avenue Jean Portalis 37200 Tours, France <a href="http://polytech.univ-tours.fr">polytech.univ-tours.fr</a>	CHU Bretonneau 2 Boulevard Tonnellé 37000 Tours, France <a href="http://chu-tours.fr">chu-tours.fr</a>
--	---

Nom	Email	Qualité
Steven MARTIN	<a href="mailto:steven.martin@etu.univ-tours.fr">steven.martin@etu.univ-tours.fr</a>	Étudiant DI5
Pascal MAKRIS	<a href="mailto:pascal.makris@univ-tours.fr">pascal.makris@univ-tours.fr</a>	Tuteur académique, Département Informatique
Franck MARMOUSET	<a href="mailto:f.marmouset@chu-tours.fr">f.marmouset@chu-tours.fr</a>	Client



# Avertissement

Ce document a été rédigé par Steven MARTIN susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Pascal MAKRIS susnommé le tuteur académique.

Le CHU Bretonneau de Tours est représenté par Franck MARMOUSET susnommé le client du projet.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non-respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assume l'entière responsabilité de la véracité de ces propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



## Pour citer ce document

Steven MARTIN, *Caractérisation de la voix*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2019-2020.

# Table des matières

<b>Liste des intervenants</b> .....	2
<b>Avertissement</b> .....	3
<b>Pour citer ce document</b> .....	4
<b>Table des matières</b> .....	5
<b>Table des figures</b> .....	8
<b>Introduction</b> .....	10
Acteurs, enjeux et contexte .....	10
Objectifs .....	10
Hypothèses .....	11
Bases méthodologiques .....	11
<b>Description générale</b> .....	12
Environnement du projet .....	12
Caractéristiques des utilisateurs .....	12
Fonctionnalités du système .....	13
Structure générale du système .....	14
<b>Etat de l’art</b> .....	16
La voix et ses paramètres .....	16
Une application nous servant de base : Praat .....	20
Système existant .....	24
<b>Analyse et conception</b> .....	28
Identification des problèmes de l’algorithme de calcul de période .....	28
Trois algorithmes de calcul de périodes .....	29
Algorithme Sf .....	29
Algorithme $\Delta f$ .....	30
Algorithme Yin.....	30
Implémentation d’une base de données optimisée .....	35
Pistes d’amélioration de l’application .....	36
Réseau de neurones .....	36
Bibliothèques de traitement de signal.....	36
<b>Mise en œuvre</b> .....	37
Structure d’un projet Android et cycle de vie .....	37

Base de données .....	40
Interface.....	43
Analyse de la voix et enregistrement.....	52
TarsosDSP .....	53
Méthodes de calcul pour les caractéristiques de la voix.....	54
Implémentation de l'algorithme Yin .....	54
Permissions.....	59
Tests .....	59
<b>Bilan et conclusion</b> .....	61
Fait et Reste à faire .....	61
Organisation prévisionnelle.....	62
Organisation finale .....	62
Bilan qualité .....	62
Bilan auto-critique sur la gestion de projet.....	62
<b>Bibliographie</b> .....	63
<b>Annexes</b> .....	64
Description des interfaces externes du logiciel .....	64
Interfaces matériel/logiciel .....	64
Interfaces hommes/machine .....	64
Interfaces logiciel/logiciel .....	67
Spécifications fonctionnelles.....	67
Définition de la fonction « Enregistrer sa voix ».....	67
Définition de la fonction « Visualiser le graphe de suivi » .....	68
Définition de la fonction « Visualiser l'historique » .....	68
Spécifications non fonctionnelles.....	69
Contraintes de développement et conception .....	69
Contraintes de fonctionnement et d'exploitation.....	69
Gestion de projet .....	71
Tâches Kanban : Trello .....	71
Diagramme de Gantt : GanttProject prévisionnel.....	72
Diagramme de Gantt : GanttProject final .....	76
Détails et compléments d'analyse .....	78
Algorithme de calcul de période existant .....	78
Algorithme de calcul de shimmer existant .....	80
Algorithme de calcul de jitter existant.....	81
Algorithme de calcul de F0 (fréquence fondamentale) existant.....	82

Documentation développeur.....	83
I. Modélisation du projet.....	83
1. Modélisation UML du projet .....	83
2. Base de données.....	85
II. Environnement de développement et langage.....	86
III. Versions et librairies utilisées.....	86
1. Versions utilisées et supportées.....	86
2. Librairies utilisées .....	87
2.1. TarsosDSP .....	87
2.2. MPAndroidChart.....	88
IV. Permissions.....	89
V. Tests.....	89
VI. Intégration continue et gestion de version .....	89
VII. Installation.....	92
Documentation utilisateur .....	92
I. Installation.....	92
II. Lancement de l'application .....	92
III. Fenêtre principale .....	92
IV. Fenêtre d'historique.....	95
V. Fenêtre de suivi .....	96
Cahier de tests .....	97

# Table des figures

Figure 1 - Diagramme de cas d'utilisation de l'application mobile .....	13
Figure 2 - Diagramme de classe de l'application existante .....	14
Figure 3 - Diagramme de classes de la nouvelle application.....	15
Figure 4 - Schéma représentant le conduit vocal.....	16
Figure 5 - Schéma représentatif du shimmer .....	17
Figure 6 – Menu principal de Praat .....	20
Figure 7 - Fenêtre de visualisation et d'édition d'un signal.....	21
Figure 8 - Menu vers Voice Report.....	22
Figure 9 - "Voice Report" / Fenêtre de sortie répertoriant les valeurs des paramètres de la voix spécifiés auparavant.....	22
Figure 10 - Menu principal de l'application existante .....	24
Figure 11 - Ecran après enregistrement .....	25
Figure 12 - Ecran d'analyse de l'enregistrement .....	26
Figure 13 - MCD de la base de données existante .....	27
Figure 14 - Comparatif des résultats de l'application avec ceux de Praat .....	28
Figure 15 - Algorithme Sf.....	29
Figure 16 - Algorithme $\Delta f$ .....	30
Figure 17 - Exemple de signal vocal et affichage de similarité entre deux pics .....	31
Figure 18 - Graphe d'autocorrélation en fonction du lag.....	31
Figure 19 - En haut, résultat de la fonction différence sur un signal ; en bas, résultat de la fonction différence normalisée par moyenne cumulée .....	33
Figure 20 – Pourcentage d'erreurs possibles sur chaque étape de l'algorithme .....	34
Figure 21 - MCD simplifié de la base de données .....	35
Figure 22 - Exemple de structure de projet Android (source : <a href="https://www.tutlane.com/tutorial/android/android-app-project-folder-structure">https://www.tutlane.com/tutorial/android/android-app-project-folder-structure</a> ) .....	37
Figure 23 - Logo de Gradle .....	38
Figure 24 - Schéma explicatif d'un cycle de vie d'une activité .....	39
Figure 25 - Classe Helper pour créer la base de données SQLite .....	40
Figure 26 - Constructeur de l'Helper .....	40
Figure 27 - Méthode onCreate de l'Helper permettant de créer la base de données .....	41
Figure 28 - Classe DBManager permettant de manipuler la base de données.....	41
Figure 29 - Méthode d'ajout d'un enregistrement dans la base de données .....	42
Figure 30 - Méthode permettant de récupérer tous les enregistrements de la base de données.....	43
Figure 31 - Différentes formes de l'écran d'accueil.....	44
Figure 32 - Énumération contenant les différents états du bouton micro de la page d'accueil.....	45
Figure 33 - Progression de la barre de progression via un Thread.....	45
Figure 34 - Sauvegarde d'un enregistrement dans la base de données et sur le téléphone .....	46
Figure 35 - Méthode permettant de créer une notification .....	46
Figure 36 - Affichage de la notification cliquable.....	47
Figure 37 - Page d'historique.....	48

Figure 38 - Adapter permettant de personnaliser la liste pour contenir des objets Record .....	48
Figure 39 - Exemple de graphes créés via MPAndroidChart .....	49
Figure 40 - Ajout de données au graphique .....	49
Figure 41 - Affichage du graphe .....	49
Figure 42 - Méthode permettant de configurer un graphique .....	50
Figure 43 - Page de suivi .....	51
Figure 44 - Ajout de ligne limites à ne pas franchir indiquant les seuils pathologiques .....	51
Figure 45 - Affichage des dates en abscisse .....	52
Figure 46 - Méthode pour récupérer les dates à partir des noms des fichiers enregistrés .....	52
Figure 47 - Dispatcher utilisant le microphone par défaut du téléphone .....	53
Figure 48 - Processus permettant d'écrire les données captées par le microphone (données audio) dans un fichier .....	53
Figure 49 - Format de l'audio .....	54
Figure 50 - Processus permettant de détecter les fréquences de la voix en temps réel en utilisant l'algorithme Yin.....	54
Figure 51 - Classe Yin implémentant l'interface PitchDetector.....	54
Figure 52 - Fonction getPitch() implémentant l'algorithme Yin .....	55
Figure 53 - Fonction difference() équivalent à l'étape 2 de l'algorithme Yin .....	56
Figure 54 - Fonction cumulativeMeanNormalizedDifference() équivalent à l'étape 3 de l'algorithme Yin .....	56
Figure 55 - Fonction absoluteThreshold() équivalent à l'étape 4 de l'algorithme Yin .....	57
Figure 56 - Fonction parabolicInterpolation équivalent à l'étape 5 de l'algorithme Yin.....	58
Figure 57 - Fichier de configuration pour l'intégration continue via GitHub Actions .....	60
Figure 58 - Nouvel écran d'accueil .....	64
Figure 59 - Ecran d'historique des enregistrements .....	65
Figure 60 - Ecran de suivi de la voix .....	66
Figure 61 - Diagramme d'activités de l'application .....	67
Figure 62 - Trello (Tableau organisationnel des tâches) .....	71
Figure 63 - Export Gantt prévisionnel, Page 1 (Intro) .....	72
Figure 64 - Export Gantt prévisionnel, Page 2 (Tâches 1) .....	72
Figure 65 - Export Gantt prévisionnel, Page 3 (Tâches 2) .....	73
Figure 66 - Export Gantt prévisionnel, Page 4 (Ressources) .....	73
Figure 67 - Export Gantt prévisionnel, Page 5 (Diag. Gantt) .....	74
Figure 68 - Export Gantt prévisionnel, Page 6 (Diag. Ressources) .....	75
Figure 69 - Export Gantt final - Général .....	76
Figure 70 - Export Gantt final .....	77
Figure 71 - Algorithme de calcul de période existant .....	79
Figure 72 - Algorithme de calcul de shimmer existant.....	80
Figure 73 - Algorithme de calcul de jitter existant .....	81
Figure 74 - Algorithme de calcul de F0 existant .....	82

# 1

## Introduction

Ce document vise à aborder l'ensemble des spécifications ainsi que la planification concernant le projet de recherche et développement (PR&D) s'intitulant « Caractérisation de la voix » réalisée dans le cadre de la dernière année du cycle d'ingénieur à l'école Polytech 'Tours.

### 1

#### Acteurs, enjeux et contexte

Ce projet met en jeu différents acteurs. En effet, l'expression du besoin a été effectuée par le docteur Franck MARMOUSET du CHRU Bretonneau de Tours qui est donc le maître d'ouvrage (MOA) et le client de ce projet. La réalisation du projet, quant à elle, est assignée à Steven MARTIN, Fabien PUPETTO (4A) et Yohan GAURIAT (4A) les étudiants désignés maîtres d'œuvre (MOE). L'auteur de ce document est Steven MARTIN. La supervision de la rédaction de ce cahier de spécifications est réalisée par M. Pascal MAKRIS.

Ce projet est réalisé dans le cadre d'une demande de plus en plus forte de pouvoir faire ses bilans de santé soi-même pour savoir quand consulter. En effet, le docteur Franck MARMOUSET du CHRU Bretonneau de Tours qui est à l'origine du projet voulait détecter les irrégularités dans la voix de ses patients pour ensuite leur proposer une visualisation en fonction du temps de l'« état » de la voix en se basant sur différents critères. Le projet existe déjà depuis un moment sous forme d'application Android et plusieurs groupes ont travaillé dessus avant. Etant donné qu'il existe un autre groupe d'étudiants (en 4A) sur le sujet cette année, des tâches seront divisées puis distribuées entre nous.

### 2

#### Objectifs

L'objectif du projet est d'améliorer l'application Android déjà existante pour la rendre utilisable par n'importe qui de manière fiable (correction de l'algorithme, corrections de bugs et amélioration de l'interface existante). Cette application, installable sur n'importe quel smartphone ou tablette disposant d'un microphone, enregistre la voix des patients, traite le signal émis par la voix et affiche des informations évoluant au cours du temps pour donner une idée à l'utilisateur de l'état de sa voix.

Cette application mesure les variations de fréquence dites « jitter » et d'amplitude « shimmer » et peut sauvegarder les enregistrements sous forme de fichiers audio pour les réutiliser. Il est aussi possible d'analyser des enregistrements externes à l'application.

Pour résumer, l'application devra comporter les fonctionnalités suivantes :

- Enregistrement de la voix et analyse en tâche de fond
- Gestion des enregistrements : possibilité de supprimer un enregistrement mal fait
- Affichage des informations caractérisant la voix via un graphique de suivi en fonction du temps
- Sauvegarde des enregistrements dans une base de données temporelle

Les demandes du client évoluent et ont déjà été plutôt mal spécifiées jusqu'ici donc il se peut que certaines caractéristiques de l'application soient manquantes, ce qui pourrait remettre en cause la partie décrivant les fonctions. Si les demandes changent, il faudra voir si ces changements sont possibles et s'ils le sont, les changements seront faits au niveau de l'application et au niveau de ce document.

De plus, si l'amélioration de l'algorithme et des interfaces présentes prend beaucoup trop de temps il faudra simplement continuer d'utiliser l'existant car les résultats théoriques sont globalement bons mais pas parfaits.

Enfin, si la version d'Android utilisée par les patients ne correspond pas aux versions sur lesquelles ont été testées l'application (version 9 et 10), il faudra qu'ils mettent à jour leur système.

Dans le cas du développement d'une application Android, nous avons décidé de réutiliser le langage existant qui est Java. Les sources du logiciel seront versionnées grâce à un dépôt GitHub créé pour l'occasion. Pour ce qui est de la modélisation des fonctionnalités du système, nous allons utiliser le langage UML.

Afin de conduire et gérer ce projet, l'outil GanttProject sera utilisé pour réaliser la planification des tâches et le suivi de celles-ci grâce à un diagramme de Gantt. Pour compléter cela, un tableau Trello listant les tâches à faire, les tâches en cours et les tâches faites sera créé et utilisé.

Côté méthodologie, en général nous allons mettre en place une méthode en cascade car le fonctionnement du PRD (première partie de spécifications puis développement) correspond parfaitement à cette méthode.

## 2

# Description générale

### 1

## Environnement du projet

Ce projet est une application Android, il faudra donc posséder un smartphone ou une tablette Android compatible. Au lancement, l'outil demande différents accès primordiaux pour son bon fonctionnement : l'accès au microphone pour enregistrer la voix et l'accès aux dossiers locaux pour sauvegarder les enregistrements audios. La sauvegarde des enregistrements audios est possible grâce à une base de données SQLite embarquée. La visualisation de l'état de la voix va se faire au travers d'un graphe répertoriant l'évolution de certains critères de la voix en fonction du temps.

Ce projet s'intégrera donc dans l'existant en reprenant les différentes implémentations de calcul effectuées et en les améliorant mais aussi en recréant totalement l'interface qui n'est pour le moment pas conforme aux attentes du client.

### 2

## Caractéristiques des utilisateurs

Ce projet visera qu'un seul et unique type d'utilisateur : les patients de l'hôpital. Ces utilisateurs n'ont pas spécialement et n'ont pas besoin de connaissance en informatique, ils utiliseront l'application mobile qui se doit d'être la plus simple et la plus intuitive possible pour n'importe qui en utilisant des éléments d'interface communs à toute application (icônes classiques, boutons simples, ...). Ces utilisateurs devront utiliser l'application de manière régulière pour suivre l'évolution de leur voix.

L'utilisateur devra simplement accepter la demande d'autorisation d'accès aux fichiers locaux et au microphone du téléphone pour enregistrer la voix et stocker les fichiers.

### 3 Fonctionnalités du système

Lors de ce projet, une seule partie est donc développée : celle de l'utilisateur.

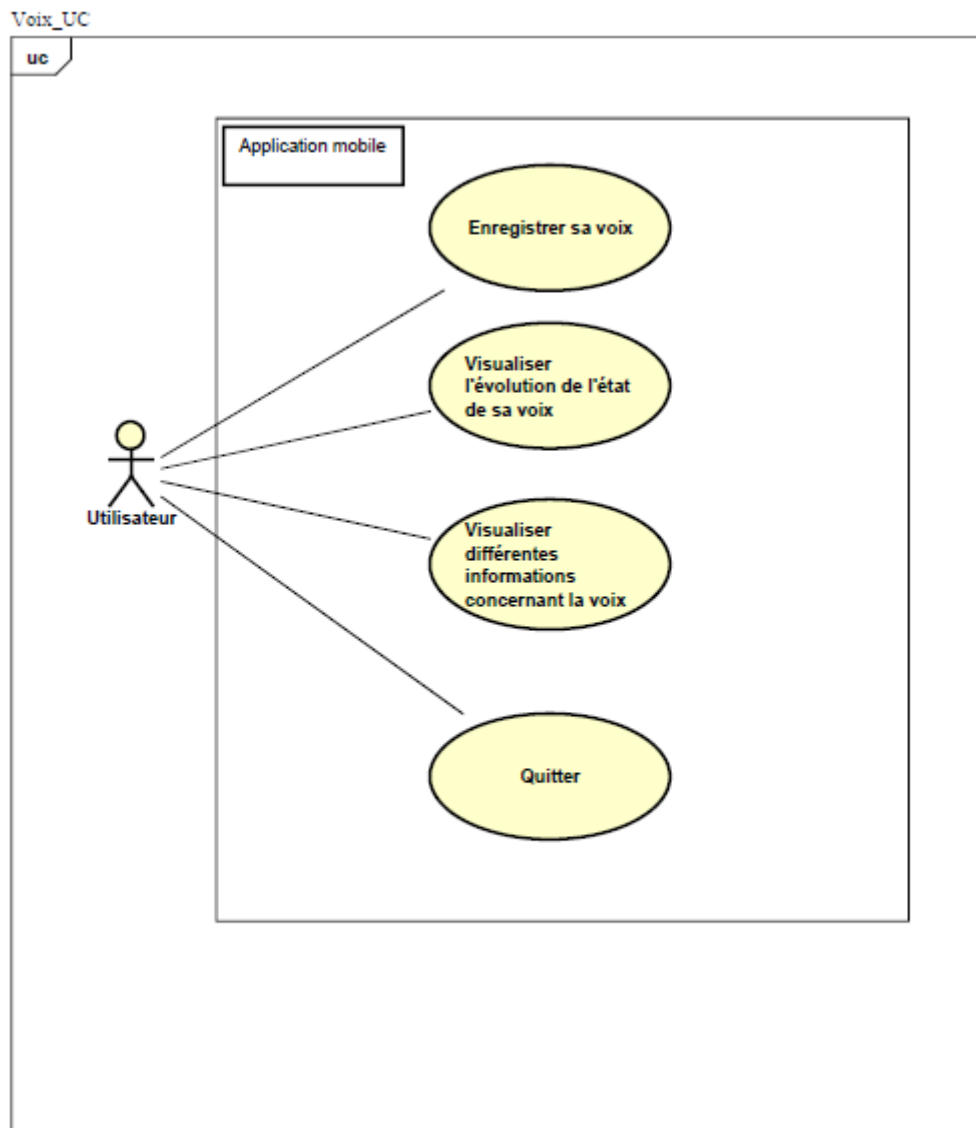


Figure 1 - Diagramme de cas d'utilisation de l'application mobile

L'utilisateur de l'application pourra utiliser celle-ci afin d'effectuer différentes actions à travers les fonctionnalités de l'application. Ainsi, il pourra tout d'abord enregistrer sa voix. Cet enregistrement sera ensuite affiché sur un graphe correspondant à l'évolution de l'état de sa voix qu'il pourra consulter. L'utilisateur pourrait aussi, pour mieux comprendre comment analyser le graphique, accéder à une fenêtre d'aide répertoriant différentes informations concernant la voix en général et concernant les points à analyser sur le graphe produit. De plus, l'utilisateur pourra accéder aux options de l'application lui permettant de configurer l'affichage. Enfin, il pourra quitter l'application quand il le souhaite.

Le système sera structuré selon un modèle MVC :

- Modèle comprenant les données définissant l'enregistrement audio et les données représentant un enregistrement dans la base de données
- Vue comprenant l'interface de l'application comprise dans les ressources
- Contrôleur comprenant toutes les activités (en développement Android, cela signifie les actions que vont effectuer chaque déclencheur d'action), les calculs des paramètres de la voix à analyser, les enregistreurs audio, les dessinateurs de graphe selon les différents points calculés, le convertisseur en fichier traitable par Android (en effet, suite à l'enregistrement nous obtenons des fichiers audio bruts en format PCM qui ne sont pas lisibles via le lecteur Android de base, il faut donc le convertir en un format lisible par Android qui sera ici le format WAV) et toute la gestion de la base de données.

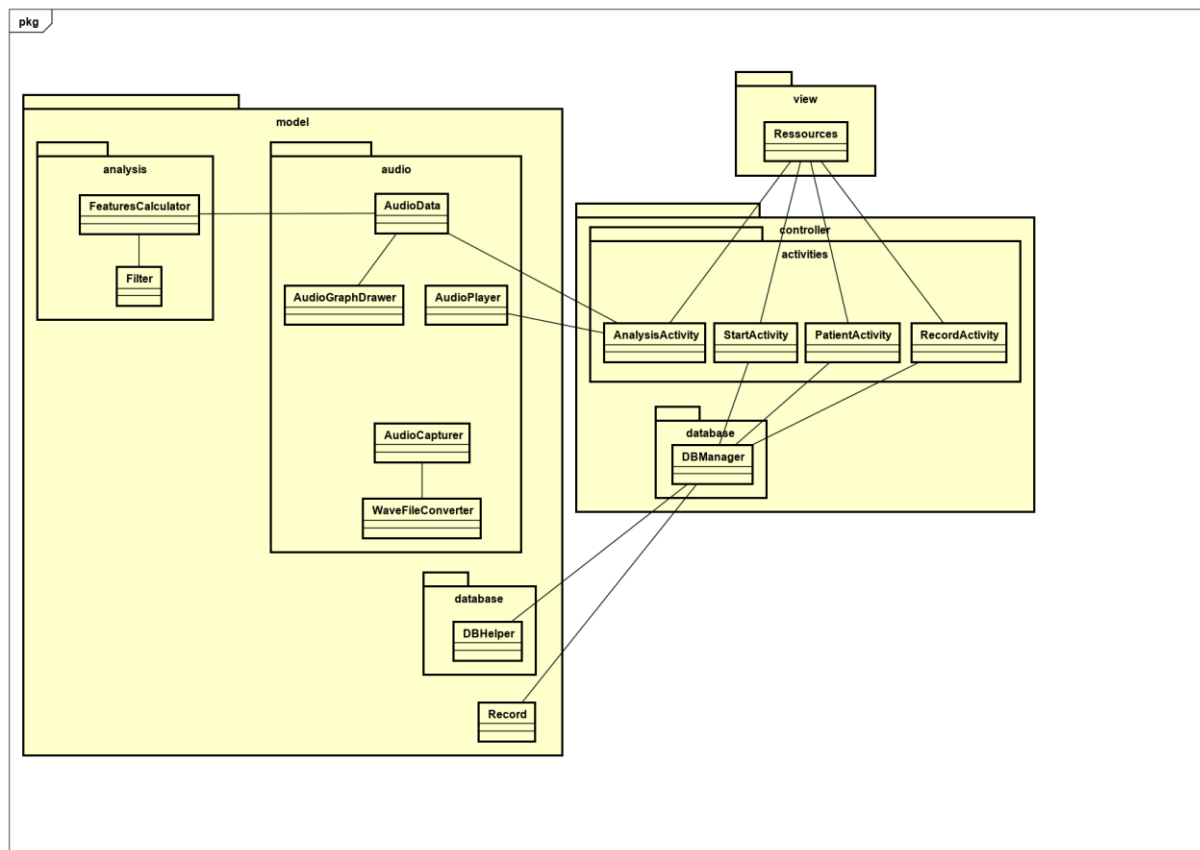


Figure 2 - Diagramme de classe de l'application existante

Le package activities correspond à toutes les manipulations sur l'interface en Android. Ainsi, ce package va manipuler les vues stockées dans les ressources (en XML). Chaque activité correspond à un écran. Ici, étant donné que c'est le diagramme correspondant à l'application existante des fenêtres comme Patient pour la gestion de patients sont présentes mais elles ne devraient plus l'être comme on va le voir dans le prochain diagramme de classe représentant l'application à créer.

Le modèle et le contrôleur ont tous deux un package database, c'est normal car la classe DBHelper permet de créer les tables (elle crée donc le modèle sur lequel on va exécuter des requêtes) et la classe DBManager exécute des requêtes directement vers ces tables.

La classe FeaturesCalculator permet de calculer le jitter, le shimmer, F0 et les périodes. La classe Filter permet de créer un filtre passe-bas et un filtre passe-haut pour la gestion des signaux bruités.

Passons maintenant au diagramme de classes correspondant à la nouvelle application qui ne gère plus les patients et intègre de nouvelles sous-activités, appelées Fragments :

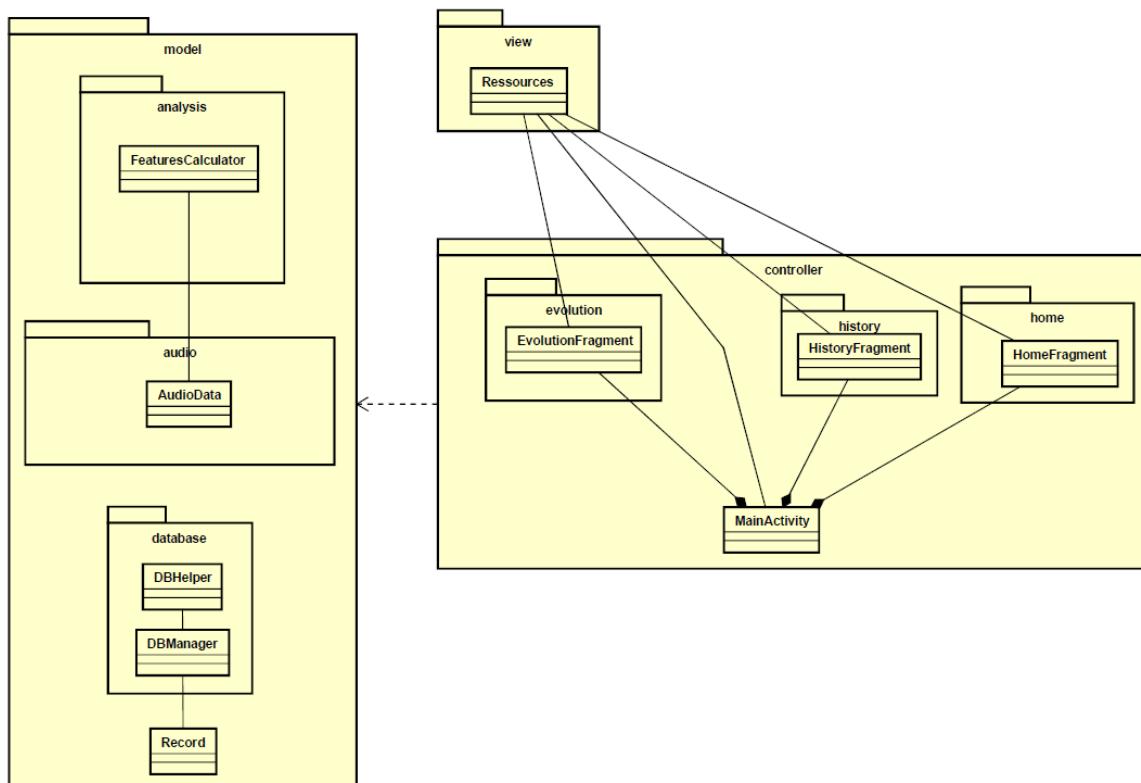


Figure 3 - Diagramme de classes de la nouvelle application

Comme dit auparavant, ce diagramme de classe est similaire mais n'intègre plus la gestion de patients et ajoute deux nouvelles fenêtres : l'historique et le suivi. Le fragment « EvolutionFragment » correspond à l'écran de suivi du patient lui affichant un graphique des différentes valeurs prises lors des enregistrements et lui permettant de savoir s'il est malade ou non.

## 1 La voix et ses paramètres

La production de la voix requiert trois choses : le son produit en parlant (appelé phonation), la résonance et l'articulation.

La phonation provient des vibrations des cordes vocales (créées lorsqu'un flux d'air pénètre dans le larynx). Elle est amplifiée et modifiée par les organes résonateurs (la gorge, les cavités buccales et les passages nasaux) et par les organes articulateurs (la langue, le palais et les lèvres) afin que des mots reconnaissables soient prononcés.

L'appareil vocal est souvent décrit comme étant un modèle source-filtre où la source est constituée du larynx, alimenté en air par les poumons par l'intermédiaire de la trachée.

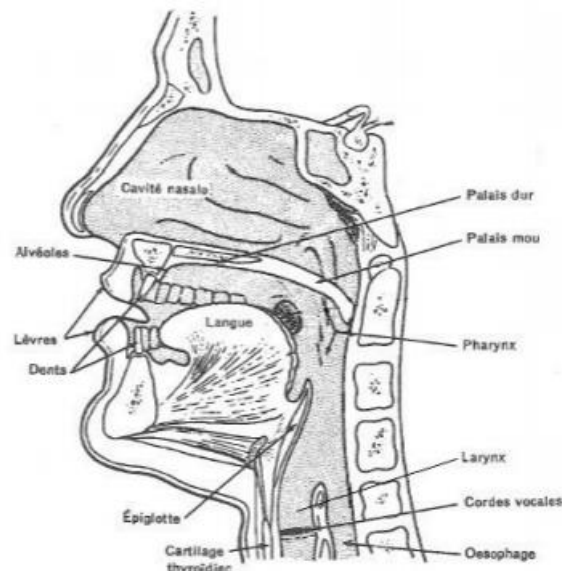


Figure 4 - Schéma représentant le conduit vocal

Les conditions de santé des cordes vocales influent sur la qualité de la voix. En effet, si les cordes vocales sont inflammées des problèmes peuvent survenir et obstruer les vibrations ce qui va causer des problèmes dans l'émission de la voix. Les cordes vocales vibrant difficilement, il se peut que la personne tente de parler mais qu'aucun son ne sorte de sa bouche. Dans le cas de voix malade, le signal émis par la voix comporte des fluctuations anormales (la voix est tremblante, enrouée ou rugueuse par exemple). Les problèmes de la voix les plus communs sont la paralysie des cordes vocales, un œdème au niveau des cordes vocales ou encore la dystonie laryngée.

Beaucoup d'études ont été faites concernant la détection de différents types de problèmes sur la voix. De ces études sont ressortis les paramètres utilisés couramment pour détecter des variations au niveau de la voix : le shimmer, le jitter et la fréquence fondamentale notée F0. Ces paramètres vont être spécifiés un à un dans ce document.

Par ailleurs, il faut savoir qu'il existe d'autres paramètres moins importants mais qui permettent d'avoir une vision plus précise de l'état de la voix du sujet étudié. Ces paramètres sont le quotient de perturbation d'amplitude (APQ<sup>1</sup>), le quotient de perturbation de tonalité (PPQ<sup>2</sup>), le ratio harmonique-bruit (HNR<sup>3</sup>), l'énergie du bruit normalisé, les indices de turbulences vocales, les indices de phonation douce, les tremblements d'amplitude et de fréquence, et bien d'autres encore...

Le HNR peut être un paramètre utile pour connaître le taux de bruit dans le signal étudié. Si ce paramètre dépasse les 20dB, le signal est considéré comme bruité. Il est calculé de cette manière :

$$\frac{\sum_{k=0}^{N-1} h[k]^2}{\sum_{k=0}^{N-1} n[k]^2}$$

où k est l'indice d'échantillonnage, h et n sont des signaux discrets et N le nombre de périodes

Parlons maintenant des paramètres utilisés dans ce projet qui sont le shimmer, le jitter et la fréquence fondamentale F0.

Dans un premier lieu, nous allons présenter ce qu'est le shimmer puis nous allons passer à l'explication de ce qu'est le jitter et enfin nous allons détailler ce qu'est la fréquence fondamentale de la voix.

Le shimmer, noté Shim, correspond à la variation d'amplitude dans la voix. Il représente la variation relative entre chaque période des amplitudes pics-à-pics. Un schéma représentatif de ce paramètre est disponible ci-dessous :

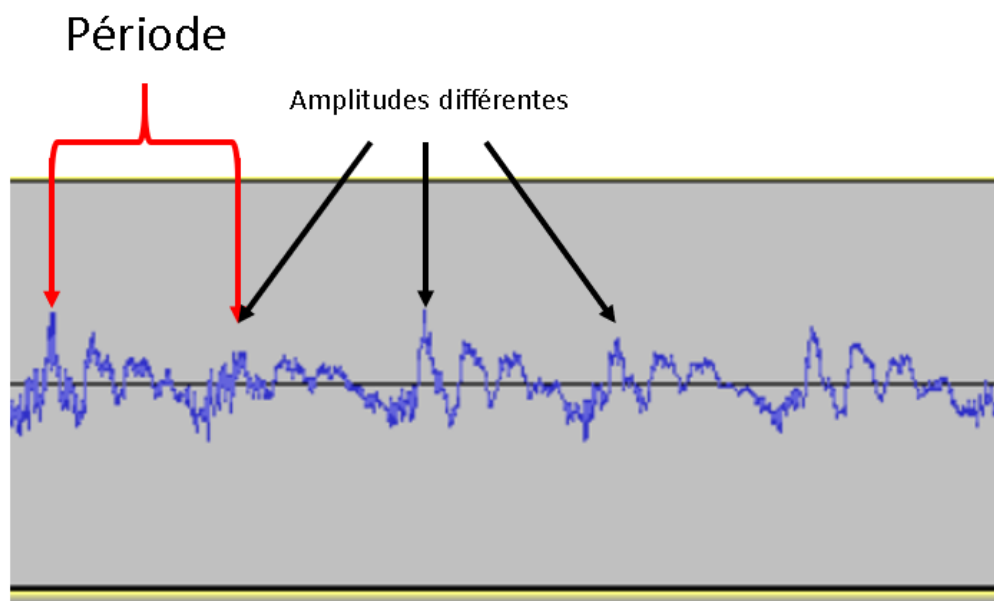


Figure 5 - Schéma représentatif du shimmer

<sup>1</sup> Amplitude Perturbation Quotient

<sup>2</sup> Pitch Perturbation Quotient

<sup>3</sup> Harmonic-to-Noise Ratio

Ce paramètre possède différentes méthodes de calcul selon les besoins, elles sont décrites ci-dessous :

- Shimmer(dB) exprime la variabilité des amplitudes pic à pic en décibels, c'est-à-dire la moyenne absolue en logarithme de base 10 de la différence entre les amplitudes de périodes consécutives multipliée par 20 :

$$\frac{1}{N-1} \sum_{i=1}^{N-1} \left| 20 \log \left( \frac{A_{i+1}}{A_i} \right) \right|$$

où A est l'amplitude pic à pic extraite et N le nombre de périodes

- Shimmer(relative) est défini comme étant la moyenne absolue des différences entre les amplitudes des périodes consécutives, le tout divisé par l'amplitude moyenne :

$$\frac{\frac{1}{N-1} \sum_{i=1}^{N-1} |A_i - A_{i+1}|}{\frac{1}{N} \sum_{i=1}^N A_i}$$

où A est l'amplitude pic à pic extraite et N le nombre de périodes

- Shimmer(apq3) est la moyenne absolue des différences entre les amplitudes d'une période et de la moyenne de ses 3 plus proches voisins, le tout divisé par l'amplitude moyenne
- Shimmer(apq5) est la moyenne absolue des différences entre les amplitudes d'une période et de la moyenne de ses 5 plus proches voisins, le tout divisé par l'amplitude moyenne
- Shimmer(apq11) est la moyenne absolue des différences entre les amplitudes d'une période et de la moyenne de ses 11 plus proches voisins, le tout divisé par l'amplitude moyenne

Le shimmer qui est utilisé dans l'application est le Shimmer relatif car il permet d'avoir une valeur représentant l'intégralité du signal, ce qui est dans notre cas primordial. Le seuil pathologique lié à ce calcul de shimmer est 3.810%.

Le jitter, noté Jitt, correspond à la variation de fréquence dans la voix. Il représente la variation moyenne absolue entre des périodes consécutives, le tout divisé par la période moyenne.

Ce paramètre possède différentes méthodes de calcul selon les besoins, elles sont décrites ci-dessous :

- Jitter(absolue) représente la variation entre chaque cycle de la fréquence fondamentale, c'est-à-dire la moyenne absolue des différences entre chaque période consécutive :

$$\frac{1}{N-1} \sum_{i=1}^{N-1} |T_i - T_{i+1}|$$

où les  $T_i$  sont les périodes et N est le nombre de périodes

- Jitter(relative) est défini comme étant la moyenne absolue des différences entre les valeurs des périodes consécutives, le tout divisé par la période moyenne :

$$\frac{\frac{1}{N-1} \sum_{i=1}^{N-1} |T_i - T_{i+1}|}{\frac{1}{N} \sum_{i=1}^N T_i}$$

où les  $T_i$  sont les périodes et N est le nombre de périodes

- Jitter(rap<sup>4</sup>) est défini comme étant la moyenne absolue des différences entre les valeurs des périodes et de la moyenne entre la période en cours et ses 2 voisins, le tout divisé par la période moyenne.
- Jitter(ppq<sup>5</sup>) est une mesure de la moyenne des périodes entre une période et la moyenne de ses plus proches voisins, le tout rapporté à la période moyenne du signal étudié :

$$\frac{\frac{1}{N-4} \sum_{i=3}^{N-2} \left| T_i - \frac{(\sum_{j=1}^5 T_j)}{5} \right|}{\frac{1}{N} \sum_{i=1}^N T_i}$$

où T est la période du signal et N le nombre de périodes

Pour les mêmes raisons que pour le shimmer, l'application utilise le jitter relatif. Le seuil pathologique lié à ce calcul de jitter est 2.040%.

Maintenant, concernant la fréquence fondamentale de la voix il faut savoir que celle-ci peut être comprise entre 80 et 200 Hz (en moyenne 120Hz) pour un homme adulte et entre 160 et 260Hz (en moyenne 210Hz) pour une femme adulte. Ces valeurs varient en fonction de l'âge, du sexe et en fonction du timbre naturel de notre voix.

<sup>4</sup> Relative Average Perturbation

<sup>5</sup> Period Perturbation Quotient

## 2 Une application nous servant de base : Praat

Praat est un outil open-source de traitement de signal vocal. Cet outil, écrit en C++, est très utile pour travailler sur des études phonologiques.

L'outil étant open-source, les méthodes de calcul du shimmer et du jitter sont récupérables et analysables.

Voici une capture d'écran de ce à quoi ressemble Praat :

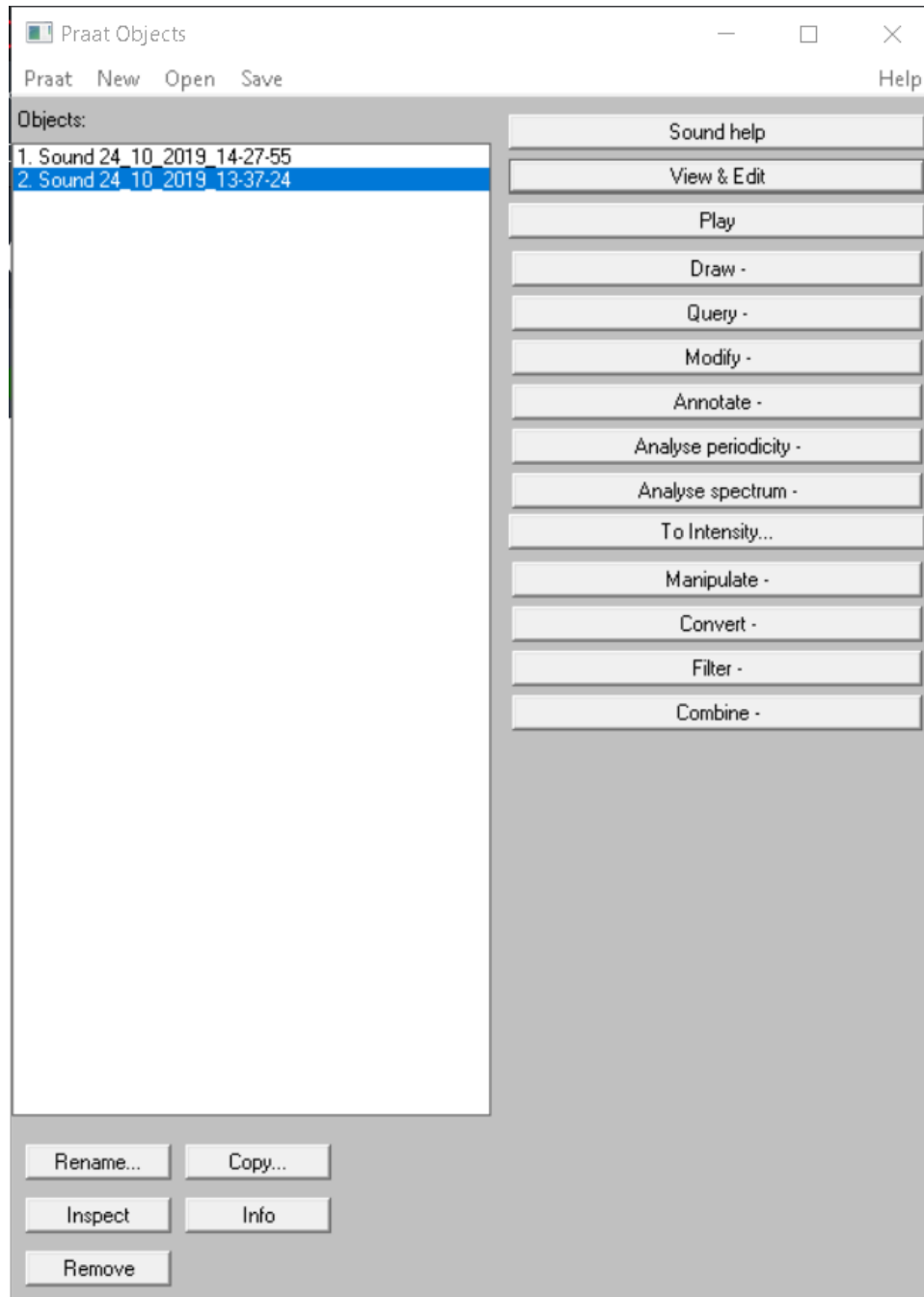


Figure 6 – Menu principal de Praat

Dans le menu principal, il est possible de faire différentes choses comme copier, analyser, filtrer ou combiner des signaux obtenus à partir de fichiers sous l'extension .WAV.

On peut par exemple, en cliquant sur « View & Edit », afficher le signal en lui-même et faire du traitement dessus en sélectionnant des parties et en cliquant sur l'analyse souhaitée.

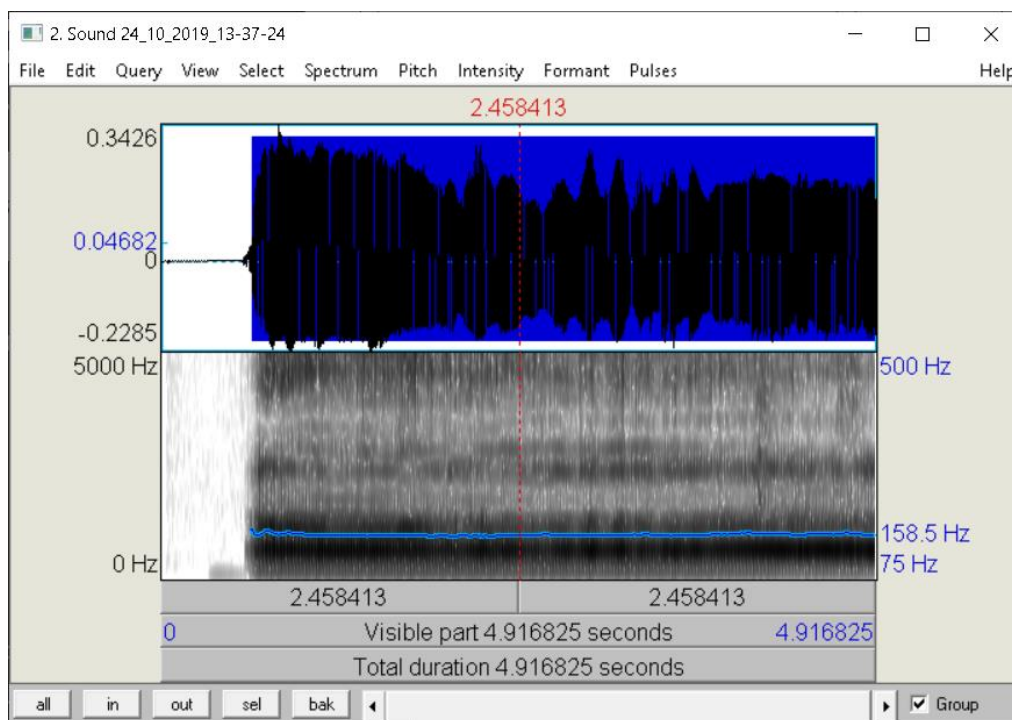


Figure 7 - Fenêtre de visualisation et d'édition d'un signal

Sur ce même signal, une fonctionnalité nous intéresse particulièrement : le calcul des paramètres vocaux. Pour accéder à cette fonctionnalité, il suffit d'aller dans « Pulses » puis de cliquer sur « Voice report ».

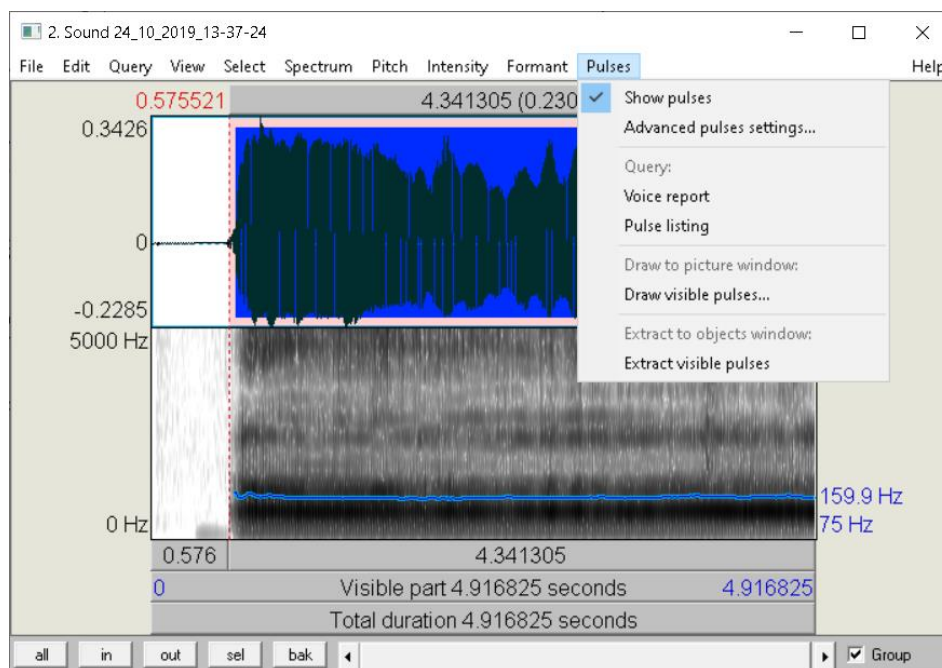


Figure 8 - Menu vers Voice Report

Une fenêtre s'ouvre indiquant tous les paramètres intéressants pour une analyse vocale :

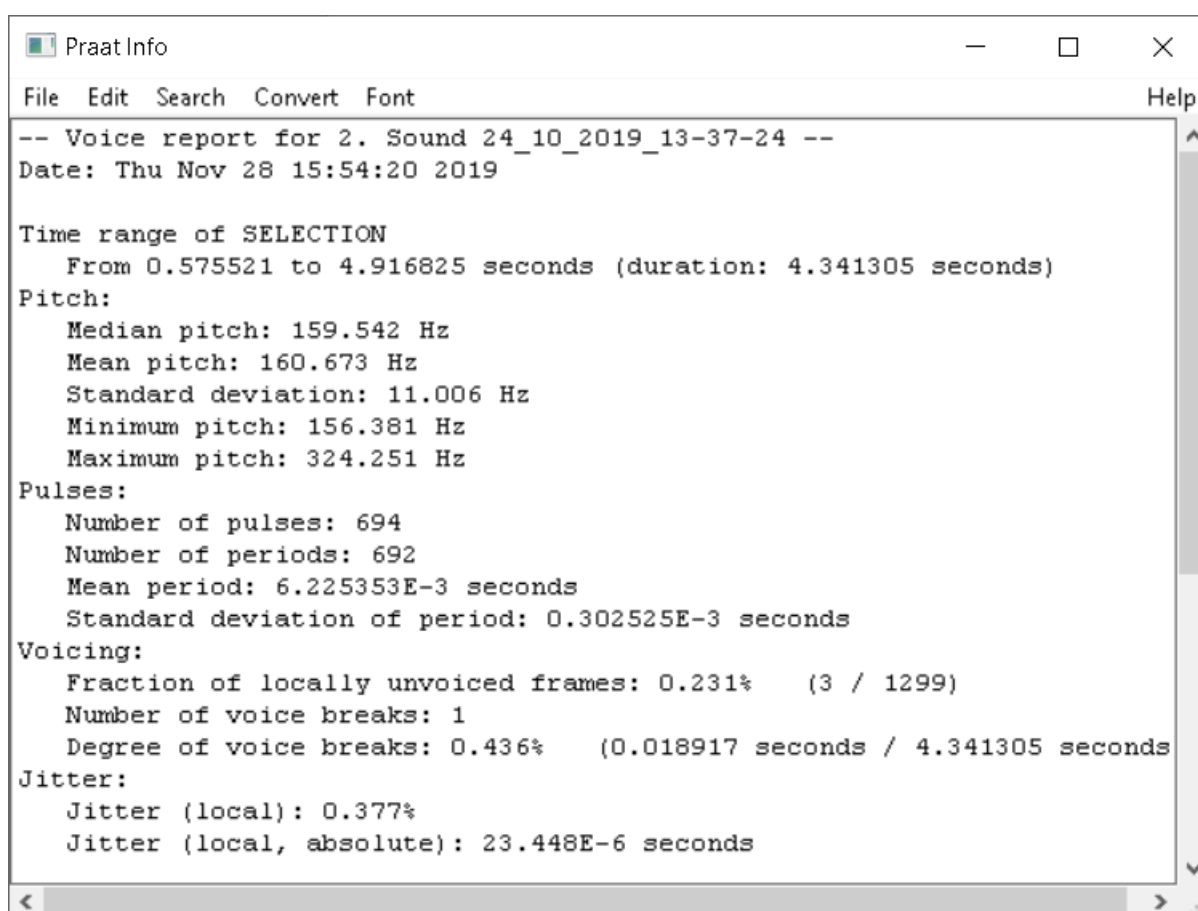


Figure 9 - "Voice Report" / Fenêtre de sortie répertoriant les valeurs des paramètres de la voix spécifiés auparavant

Grâce à cette fenêtre, il est possible d'obtenir des valeurs auxquelles on pourra comparer les valeurs sorties par notre application. Sachant que nous avons utilisé une méthode précise de calcul pour le jitter et pour le shimmer, il faut faire attention de ne pas comparer deux valeurs provenant de méthodes différentes. Dans notre cas, l'utilisation de paramètres relatifs (shimmer et jitter) implique la comparaison avec les valeurs provenant de Jitter (local) et Shimmer (local) dans Praat.

Praat est donc utilisé dans ce projet comme base pour savoir si les valeurs obtenues grâce aux algorithmes implémentés dans l'application sont correctes ou non.

Une interface pour le projet a déjà été créée. Voici l'écran d'accueil :

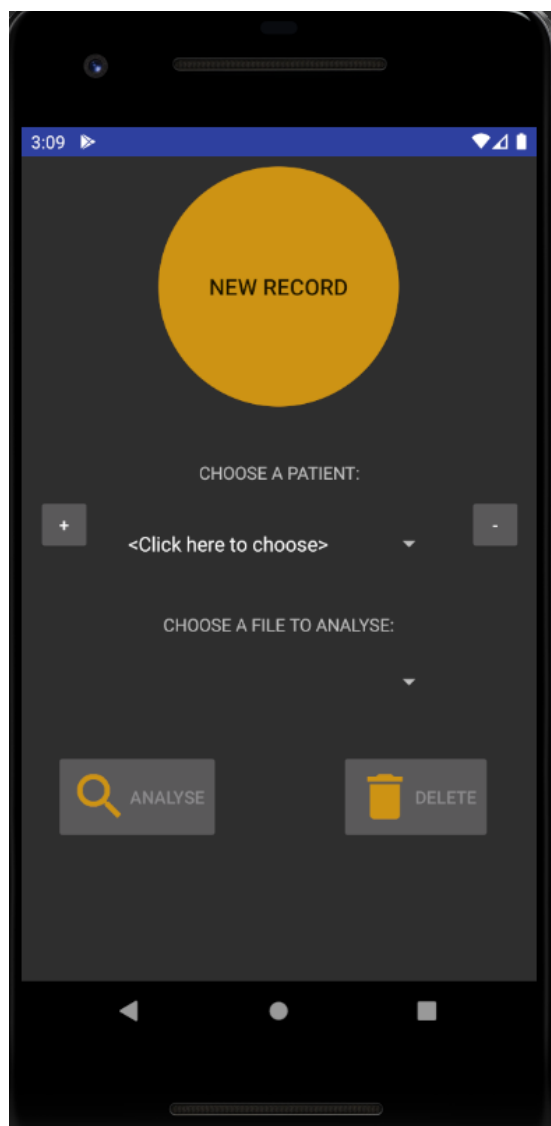
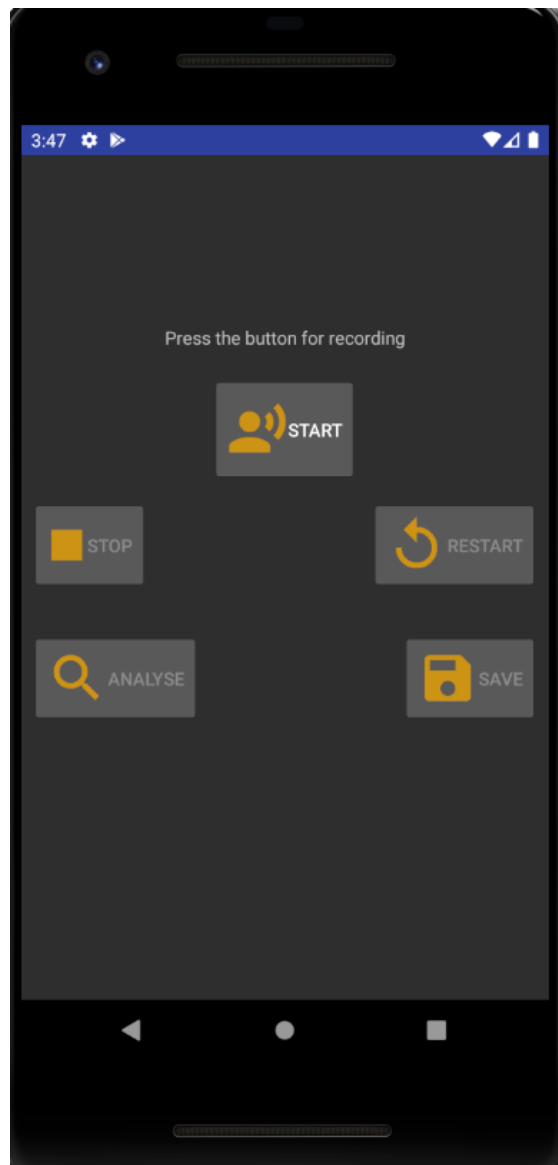


Figure 10 - Menu principal de l'application existante

L'écran d'accueil permet plusieurs choses : l'enregistrement de la voix, le choix d'un fichier à analyser, le choix d'un fichier à supprimer ainsi que le choix d'un patient.

L'enregistrement implique un changement de vue vers celle-ci :



*Figure 11 - Ecran après enregistrement*

Cette vue permet réellement l'enregistrement via le microphone de l'appareil utilisé en cliquant sur « Start ». Une fois ce bouton cliqué, les autres boutons deviennent activables et ne sont donc plus grisés.

Le bouton « Stop » permet d'arrêter l'enregistrement avant les 5 secondes prévues. Le bouton « Restart » permet de ne pas sauvegarder l'enregistrement effectué et de relancer un nouvel enregistrement.

Le bouton « Analyse » permet d'accéder à la fenêtre d'analyse du signal produit sans sauvegarder l'enregistrement.

Le bouton « Save » permet de sauvegarder l'enregistrement dans la base de données. Cet enregistrement sera ensuite sélectionnable sur la page d'accueil et sera disponible pour analyse.

La page d'analyse du signal est présentée comme ceci :

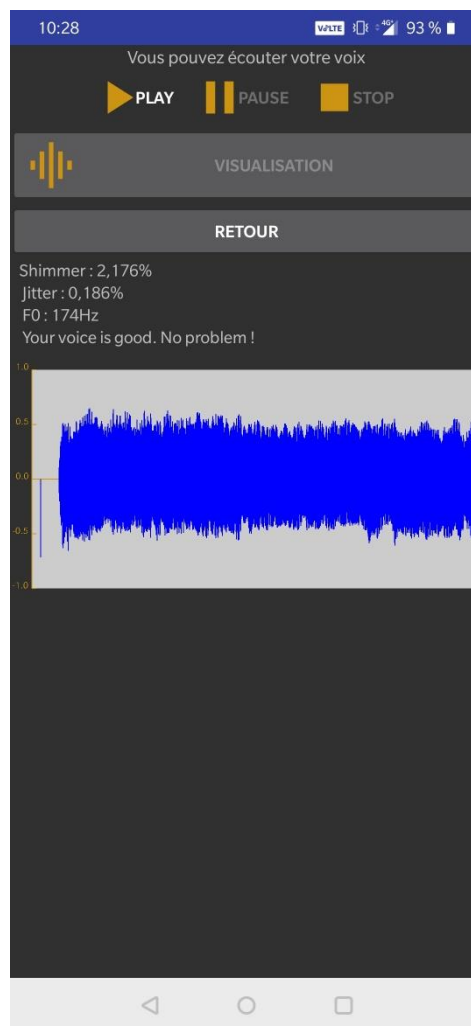


Figure 12 - Ecran d'analyse de l'enregistrement

Une liste des paramètres calculés est affichée (Shimmer, Jitter et la fréquence fondamentale F0) pour permettre au médecin de consulter les valeurs.

De plus, une phrase est affichée selon ces valeurs :

- « Your voice is good. No problem ! » indique à l'utilisateur que sa voix ne porte aucun problème pathologique. Cette phrase apparaît lorsque les valeurs n'ont pas atteint le seuil pathologique.
- « Your voice is not perfect. You'd better see a doctor. » indique à l'utilisateur que sa voix porte un problème pathologique et qu'il devrait consulter un médecin. Cette phrase apparaît dans le cas où les valeurs sont au-dessus du seuil pathologique.
- « Please test your voice again because there is some problem with the data ! » apparaît lorsque le signal n'est pas analysable, c'est-à-dire qu'il est trop bruité ou que quelque chose perturbe le microphone.

Enfin, un graphique représentant le signal audio étudié est affiché pour donner des indications directes au médecin.

Côté base de données, nous avons seulement deux tables : une table Patient et une table Voices. La table Patient répertorie tous les patients enregistrés à partir de l'application. La table Voices quant à elle répertorie tous les enregistrements effectués à partir de l'application. Ainsi, nous avons un MCD comme ceci :

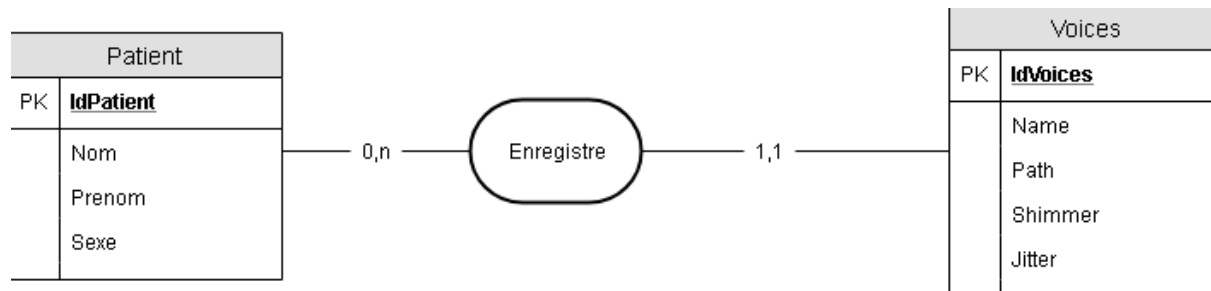


Figure 13 - MCD de la base de données existante

La table Patient stocke les informations principales concernant un patient et la table Voices stocke les informations sur le chemin vers lequel l'enregistrement a été sauvegardé sur le système ainsi que les différentes valeurs calculées pour le jitter et le shimmer.

Enfin, côté algorithmique, nous avons différentes implémentations existantes d'algorithmes de calcul du jitter, du shimmer, de F0 ainsi que des périodes disponibles en annexe (voir Annexe 5).

# 4

## Analyse et conception

Dans cette partie, nous allons tout d'abord nous intéresser aux différents problèmes de l'algorithme de calcul de période existant en proposant des premières solutions puis nous allons parler de l'implémentation d'une base de données optimisée pour stocker des enregistrements enfin nous allons parler des différentes possibilités d'amélioration de l'application (implémentation de bibliothèques, de réseaux de neurones, ...).

### 1 Identification des problèmes de l'algorithme de calcul de période

L'application existante produit des résultats plutôt cohérents (voir Figure 14) mais qui peuvent être améliorés notamment sur l'aspect du calcul de période d'un signal.

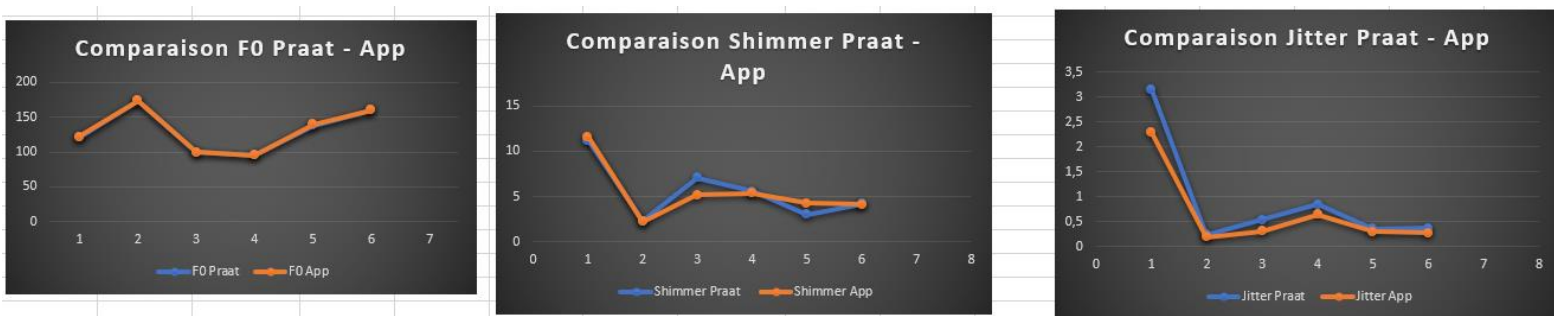


Figure 14 - Comparatif des résultats de l'application avec ceux de Praat

Dans notre cas, nous utilisons 5000 points pour calculer une période, ce qui pose deux problèmes :

- Que faire s'il n'est pas possible d'obtenir 5000 points sur le signal ?
- Que faire si les 5000 points ne représentent pas entièrement le signal ?

## 2 Trois algorithmes de calcul de périodes

Il existe déjà quelques algorithmes créés pour calculer des périodes efficacement. Trois d'entre eux ont retenu mon attention et pourront être implémentés pour améliorer le calcul des périodes présent.

### 2.1 Algorithme Sf

Cet algorithme permet d'évaluer la position d'une période  $f(t_{n+i})$  par rapport à  $f(t_i)$  et  $f(t_{i+1})$  où  $f$  correspondrait au signal étudié et  $t_n$  l'instant auquel on souhaite calculer la période. L'algorithme se présente comme ceci :

**Table 1. Sf algorithm**

**Input:** data set  $D$ , initial guess of the period ( $T_0$ ), sampling rate ( $t_1$ )

**Output:** An approximate value of the period ( $T^*$ ) or "Insufficient data"

```

 $sp \leftarrow 0, nsp \leftarrow 0$ 
 $Ni \leftarrow \text{floor}((7/8)(T_0/t_1))$ 
 $Nf \leftarrow \text{ceiling}((7/6)(T_0/t_1))$ 
for  $n = Ni: Nf$ 
  for  $i = 0 : n$ 
    if  $f(t_{n+i}) \leq f(t_{i+1}) \ \& \ f(t_{n+i}) \geq f(t_i)$ , or  $f(t_{n+i}) \geq f(t_{i+1}) \ \& \ f(t_{n+i}) \leq f(t_i)$ 
       $sp \leftarrow sp + 1$ 
    else
       $nsp \leftarrow nsp + 1$ 
    end
     $Sf(n) \leftarrow (sp - nsp)/n$ 
  end
end
if  $\max(Sf) \geq 0.0$ 
   $m \leftarrow n$  that maximizes  $Sf$ 
   $T^* \leftarrow (m + 0.5)t_1$ 
else
  "Insufficient data"
end

```

Figure 15 - Algorithme Sf

Le point  $f(t_{n+i})$  est mélangé (noté  $sp$  dans l'algorithme) si le point est entre  $f(t_i)$  et  $f(t_{i+1})$ . Sinon, ce point sera catégorisé comme étant un point non-mélangé (noté  $nsp$ ). La meilleure période « candidate » maximise la différence entre les points mélangés et les points non-mélangés.

$Sf(n)$  correspond à une mesure de la périodicité des valeurs entre deux sections consécutives pour une valeur de période candidate  $t$ . L'algorithme retourne une valeur  $T^* = (m+0.5)*t_1$  de telle sorte que  $m$  maximise  $Sf(n)$  sur l'intervalle  $[tNi, tNf]$ .

## 2.2 Algorithme $\Delta f$

Cet algorithme se base sur l'équation suivante permettant d'obtenir le maximum des différences entre chaque couple de valeurs de périodes voisines :

$$\Delta f(t) = \sum_{t_i \leq t} \max(|f(t_{n+i}) - f(t_i)|, |f(t_{n+i}) - f(t_{i-1})|)$$

La meilleure période sera donc celle qui minimisera cette différence. L'algorithme correspondant en pseudo-code est disponible ci-dessous :

**Table 2.  $\Delta f$  algorithm**

**Input:** data set  $D$ , initial guess of the period ( $T_0$ ), sampling rate ( $t_l$ )

**Output:** An approximate value of the period ( $T^*$ )

$N_i \leftarrow \text{floor}((7/8)(T_0/t_l))$

$N_f \leftarrow \text{ceiling}((7/6)(T_0/t_l))$

for  $n = N_i : N_f$

    compute  $\Delta f(n)$

end

$m \leftarrow n$  that minimizes  $\Delta f$

$T^* \leftarrow (m + 0.5)t_l$

Figure 16 - Algorithme  $\Delta f$

## 2.3 Algorithme Yin

Cet algorithme provient directement de la littérature et utilise des méthodes déjà en place et qui ont prouvé leur efficacité comme l'autocorrélation par exemple. Cette méthode d'autocorrélation est la méthode qui, de base, était utilisée pour l'application et qui est la méthode la plus utilisée dans les cas de recherche de période sur les voix. L'algorithme Yin vient ajouter des couches à cette méthode pour diminuer l'erreur le plus possible.

Cet algorithme se déroule donc en plusieurs étapes :

1. L'autocorrélation : méthode qui permet de comparer un signal avec lui-même décalé d'un temps  $\tau$  qu'on appelle « lag ».

Pour illustrer ce principe, voici un signal vocal :

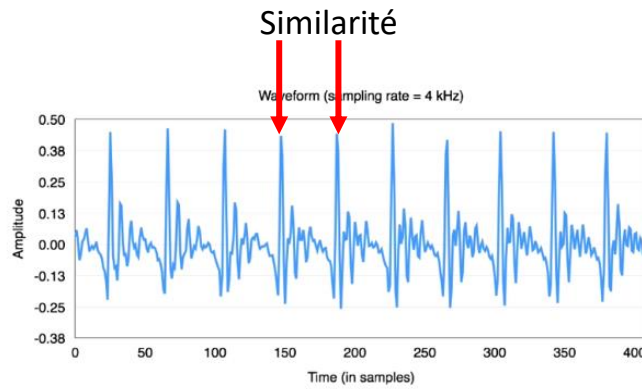


Figure 17 - Exemple de signal vocal et affichage de similarité entre deux pics

Comme nous pouvons le voir, même si le signal n'est pas parfaitement périodique, il est possible d'observer des similarités entre chaque pic. Ainsi, la méthode d'autocorrélation va ici permettre de prendre ce signal de base et de le décaler petit à petit d'un lag  $\tau$ . La méthode va retourner le taux de similarité au signal par rapport à lui-même en fonction du lag  $\tau$ . Nous obtenons donc un graphique comme ceci :

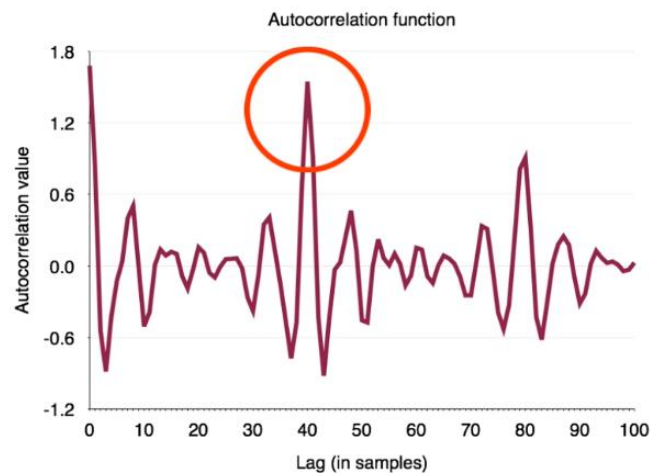


Figure 18 - Graphe d'autocorrélation en fonction du lag

L'algorithme d'autocorrélation se présente comme ceci :

$$r'_t(\tau) = \sum_{j=t+1}^{t+W-\tau} x_j x_{j+\tau}$$

Où  $t$  est le temps,  $W$  la fenêtre d'intégration (si on veut prendre seulement une partie du signal à décaler),  $x$  correspond au point de la courbe et  $\tau$  correspond au lag.

2. La différence, appelée  $d(\tau)$  : correspond à la somme des carrés des différences entre un point et lui-même décalé d'un lag  $\tau$ . Cette étape se base sur le fait qu'un signal périodique est invariant suivant un décalage dans le temps  $T$ . Nous obtenons donc ceci :

$$x_t - x_{t+T} = 0, \quad \forall t.$$

Si l'on prend la somme au carré de cette équation suivant une fenêtre d'intégration  $W$  et qu'on moyenne par rapport à cette fenêtre, nous obtenons l'équation suivante dont le résultat est équivalent à précédemment (c'est-à-dire que la valeur n'a pas changé au cours du décalage de temps  $T$  et donc que l'on a une période) :

$$\sum_{j=t+1}^{t+W} (x_j - x_{j+T})^2 = 0.$$

En prenant cette équation en compte, il est possible de trouver des périodes inconnues. Pour ceci, il suffit de chercher le  $\tau$  pour lequel la fonction différence suivante vaut 0 :

$$d_t(\tau) = \sum_{j=1}^W (x_j - x_{j+\tau})^2.$$

3. La fonction de différence normalisée par moyenne cumulée, appelée  $d'(\tau)$  : cette méthode est ajoutée ici car les signaux vocaux ne sont pas parfaitement périodiques et auront donc des différences différentes de zéro dans la majorité des cas. Une première solution serait donc d'établir une limite proche de zéro correspondant au seuil à partir duquel on considère qu'une période a été trouvée. Le problème est ici que les écarts entre les résultats de la fonction différence sont variables et peuvent être parfois trop élevés ou ne pas correspondre à une période trouvée. Ainsi, comme l'indique le document source de cet algorithme, une solution a été proposée pour remplacer la fonction de différence par une fonction de différence normalisée par moyenne cumulée :

$$d'_t(\tau) = \begin{cases} 1, & \text{if } \tau=0, \\ d_t(\tau) / \left[ (1/\tau) \sum_{j=1}^{\tau} d_t(j) \right] & \text{otherwise.} \end{cases}$$

Cela change drastiquement de la fonction différence car ici, la sortie de la fonction commence à 1 au lieu de 0, semble rester grand sur des lags très petits et diminue en-dessous de 1 quand  $d(t)$ , la fonction différence, tombe en dessous de la moyenne des différences.

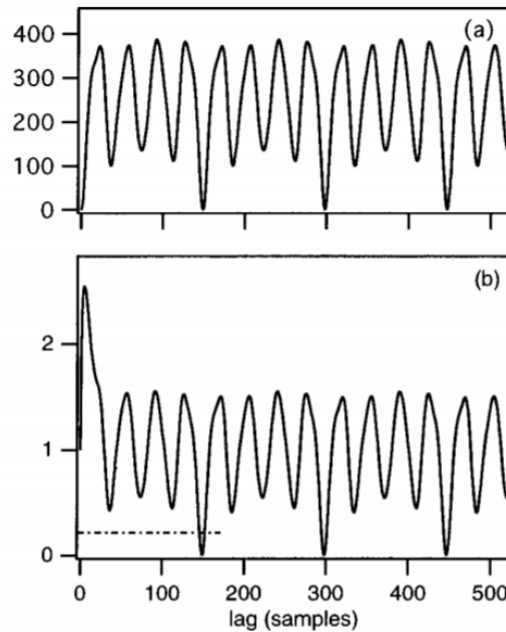


Figure 19 - En haut, résultat de la fonction différence sur un signal ; en bas, résultat de la fonction différence normalisée par moyenne cumulée

4. Le seuil absolu : pour régler le problème que les pics de période choisis sont principalement des grands pics et peuvent ne pas correspondre exactement à une période. C'est ce que l'on appelle une erreur subharmonique. Ainsi, la méthode d'ajouter un seuil absolu permet de choisir la plus petite valeur de  $\tau$  qui donne le minimum de  $d'$  (la fonction différence normalisée par moyenne cumulée) plus petit que le seuil défini. Si aucun minimum est trouvé, le minimum global est choisi.
5. L'interpolation parabolique : chaque minimum local de  $d'(\tau)$  et leurs voisins les plus proches sont représentés par une parabole et l'ordonnée du minimum interpolé est utilisé dans le processus de sélection de différence. L'abscisse quant à elle permet d'avoir une estimation de la période. Néanmoins l'estimation de la période ici est légèrement biaisée à cause des différentes transformations faites auparavant, pour avoir une période correcte l'algorithme Yin utilise le minimum de la fonction de différence  $d(\tau)$  à la place de  $d'(\tau)$ .
6. L'estimation de la meilleure différence  $d'(\tau)$  locale : recherche d'un minimum de  $d'(T_\theta)$  où  $\theta$  est compris entre  $t - T_{\max} / 2$  et  $t + T_{\max} / 2$ ,  $t$  l'indice correspondant au temps,  $T_\theta$  est l'estimation au temps  $\theta$  et  $T_{\max}$  est la période la plus grande attendue.

D'après la source de cet algorithme, en passant à travers toutes ces étapes, il est possible de passer d'une erreur de 10% environ à une erreur de 0.5% sur la recherche de période.

Version	Gross error (%)
Step 1	10.0
Step 2	1.95
Step 3	1.69
Step 4	0.78
Step 5	0.77
Step 6	0.50

*Figure 20 – Pourcentage d'erreurs possibles sur chaque étape de l'algorithme*

Ces trois algorithmes sont de possibles candidats pour remplacer l'algorithme existant mais nous allons ici nous pencher sur le dernier, l'algorithme Yin car il reprend une méthode très populaire et déjà implémentée dans l'application qu'est l'autocorrélation. Le point commun entre tous ces algorithmes est qu'ils ne prennent pas en compte un nombre d'itération fixe, ce qui est parfait pour corriger le problème des 5000 prises de l'algorithme existant.

Dans notre cas, l'implémentation d'une base de données optimisée n'est pas difficile. L'objectif de l'application est qu'elle soit installée sur le système du patient, il n'y a donc pas besoin de table Patient dans la base de données. Toutes les informations dont le patient aura besoin est déjà dans son système. La seule modification à faire est donc de supprimer la table Patient.

Les enregistrements ayant déjà un nom possédant un timestamp, il est très facile de trier les enregistrements par date en extirpant du nom les dates qui correspondent. Ainsi, nous avons un MCD simplifié comme ceci :

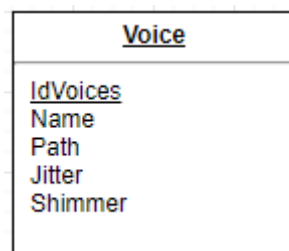


Figure 21 - MCD simplifié de la base de données

La base de données est toujours une BDD SQLite car c'est peu volumineux (très pratique pour le mobile) et l'application ne nécessite pas de stocker des millions d'enregistrements. Si l'on compte un enregistrement par jour, on aurait un millier d'enregistrements en presque 3 ans, ce qui n'est pas alarmant pour le fonctionnement de l'application.

Il serait même possible étant donné la simplicité de la base de données de ne pas en implémenter une et de simplement parcourir le dossier dans lequel les enregistrements sont sauvegardés pour obtenir les timestamps mais cela pourrait poser un problème si l'accès aux dossiers internes n'est pas possible (verrouillé). Par ailleurs, pour que le patient ne manipule pas par erreur les enregistrements dans son système de fichiers, il faudrait cacher ou verrouiller le dossier contenant ces derniers mais cela dépend de chaque appareil donc cette manipulation est difficile voire impossible.

## 4 Pistes d'amélioration de l'application

### 4.1 Réseau de neurones

L'implémentation d'un réseau de neurones permettrait à l'application d'être plus fiables dans les calculs effectués. En effet, il serait possible de catégoriser des types de voix pour ensuite créer un ensemble de possibilités désignant la santé de ces types. Ainsi, pour une voix similaire, on pourrait avoir une voix dite « en bonne santé » et une voix dite « malade » ce qui permettrait que dès lors qu'une personne possédant cette voix s'enregistre, des comparaisons soient faites en plus des différents calculs pour déterminer si réellement le patient est malade ou non.

Malheureusement, pour créer et entraîner un réseau de neurones, il faut beaucoup de données or dans notre cas, le client possède qu'une quarantaine de patients ce qui reste trop peu pour avoir une nette amélioration de la fiabilité de l'application.

### 4.2 Bibliothèques de traitement de signal

Différentes bibliothèques de traitement de signal pour Android (Java) existent. Ces bibliothèques sont listées et décrites ci-dessous :

- SoundTouch : une bibliothèque permettant de changer les paramètres classiques du son (vitesse, hauteur, etc...) écrite en C++, ce qui impliquerait l'utilisation de JNI<sup>6</sup>.
- FFTW : un outil écrit en C permettant de faire rapidement des transformées de Fourier
- Superpowered : une bibliothèque qui est aussi écrite en C++ permettant de faire de multiples choses comme la détection de BPM, la visualisation de graphe correspondant à l'audio joué et le filtrage de signal. C'est donc une bibliothèque orientée pour la musique et l'acoustique.
- TarsosDSP : une bibliothèque Java intégrant une simplification de l'enregistrement audio (fonctionne comme un framework)
- Essentia : un outil en C++ regroupant plein d'algorithmes de traitement de signal (filtrage, transformées de Fourier, détection de pics, etc...). Malheureusement, cet outil n'intègre pas d'algorithme de détection de période ni même de calcul de shimmer ou de jitter.

Ces différents outils sont inutiles dans notre cas, la seule exception étant TarsosDSP qui peut permettre de simplifier le code permettant l'enregistrement audio, car ce sont des bibliothèques plutôt orientées acoustique et musique, ce qui ne correspond pas à ce que l'on cherche. S'il existait une bibliothèque orientée phonétique et étude de la voix, il serait envisageable de l'utiliser.

---

<sup>6</sup> Java Native Interface : une interface permettant d'appeler du code externe C++ en Java

Dans cette partie, nous allons tout d'abord nous intéresser au cœur d'un projet Android et du cycle de vie d'un programme Android puis nous parlerons de l'implémentation de la base de données SQLite, ensuite nous aborderons la partie interface en présentant les différents layouts et la librairie utilisée pour faire des graphes puis nous parlerons de l'implémentation de l'algorithme Yin et des calculs des différentes caractéristiques de la voix, enfin nous verrons les différents tests réalisés et les outils utilisés pour automatiser l'exécution des tests.

L'application est compatible avec toutes les versions supérieures à la version 15 d'Android.

## 1 Structure d'un projet Android et cycle de vie

Pour commencer, nous allons parler de la structure d'un projet Android pour comprendre comment sont liées les différentes parties au sein de l'application. Voici donc un exemple de structure de projet Android :

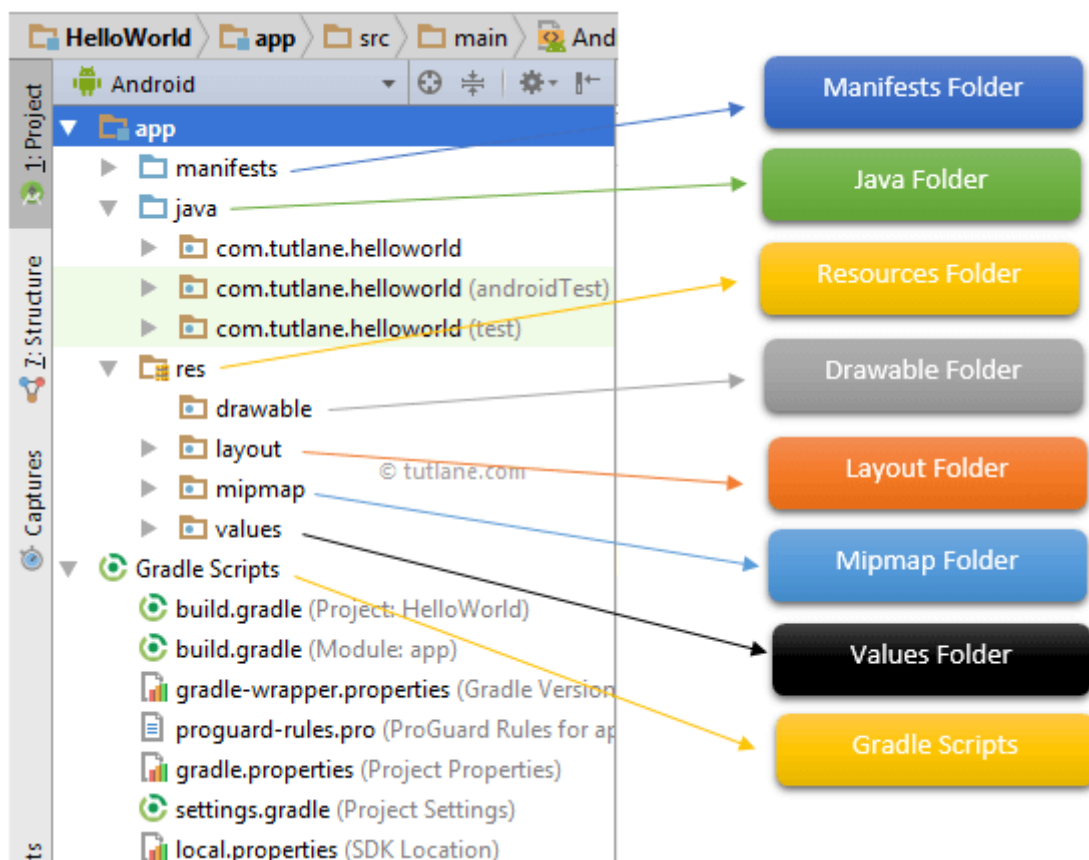


Figure 22 - Exemple de structure de projet Android (source : <https://www.tutlane.com/tutorial/android/android-app-project-folder-structure>)

Un projet Android est composé d'un fichier AndroidManifest qui n'est autre qu'un fichier xml de configuration permettant notamment d'indiquer les différentes permissions à utiliser et quelle activité lancer en premier. En parlant d'activité, une activité Android est comme une « fenêtre » qui peut être appelée au lancement de l'application grâce au fichier Manifest ou par d'autres activités.

Ces activités correspondant aux contrôleurs de l'application ainsi que le modèle de l'application sont disposées dans le dossier Java. Dans ce même dossier, deux dossiers de tests sont présents : androidTest correspond au dossier de tests fonctionnels sur un émulateur ou un appareil connecté et test correspond au dossier de tests unitaires.

Pour le côté interface de l'application, tout se passe dans le dossier res (pour ressources). Ce dossier contient tous les drawable qui sont en fait les images utilisées par l'application, tous les layouts qui seront les différentes interfaces visibles par l'utilisateur, tous les mipmaps correspondant aux icônes utilisées par l'application et des values permettant de facilement modifier une valeur d'une chaîne de caractères dans le projet.

Pour monter le projet, Android utilise par défaut Gradle qui est un moteur de production similaire à Maven qui fonctionne sur Java. Gradle est utilisé pour des soucis d'efficacité en comparaison à Maven. Avec Gradle, il est possible de définir des tâches permettant de faire des choses spécifiques qui seront notamment utilisées ici pour créer des rapports JaCoCo<sup>7</sup>.



Figure 23 - Logo de Gradle

Maintenant que nous avons vu globalement comment était structuré un projet Android, nous allons voir le cycle de vie d'une application Android. Ce point est important pour éviter tout bug lié à un appui sur le bouton Retour du téléphone ou lié à une mise en arrière-plan de l'application.

Comme dit précédemment, une application est composée d'activités. Ces dernières possèdent des méthodes propres leur permettant d'interagir avec les boutons du téléphone. Ces méthodes sont les suivantes :

- onCreate() : est appelé lors de la création de l'activité, c'est ici que tout ce dont on a besoin doit être initialisé.
- onStart() : est appelé lorsque l'activité se lance, ici aussi on pourrait initialiser des objets permettant par exemple de faire un écran de bienvenue.
- onResume() : est appelé lorsque l'activité sort de pause, c'est-à-dire que l'activité est passé en arrière-plan un instant et est revenu en premier plan.
- onPause() : est appelé lorsque l'activité passe en arrière-plan mais reste visible (pour les pop-up ou autres onglets par exemple) ou lorsque l'appareil possède des applications nécessitant beaucoup de mémoire.
- onStop() : est appelé lorsque l'activité est en arrière-plan et n'est plus visible c'est-à-dire lorsque l'on ouvre une autre application depuis le téléphone.

---

<sup>7</sup> Java Code Coverage : outil de couverture de code pour Java

- `onDestroy()` : est appelé lorsque l'application est détruite, c'est-à-dire enlevée de l'arrière-plan.

Voici un schéma provenant de la documentation officielle Android permettant de comprendre comment interagissent ces méthodes :

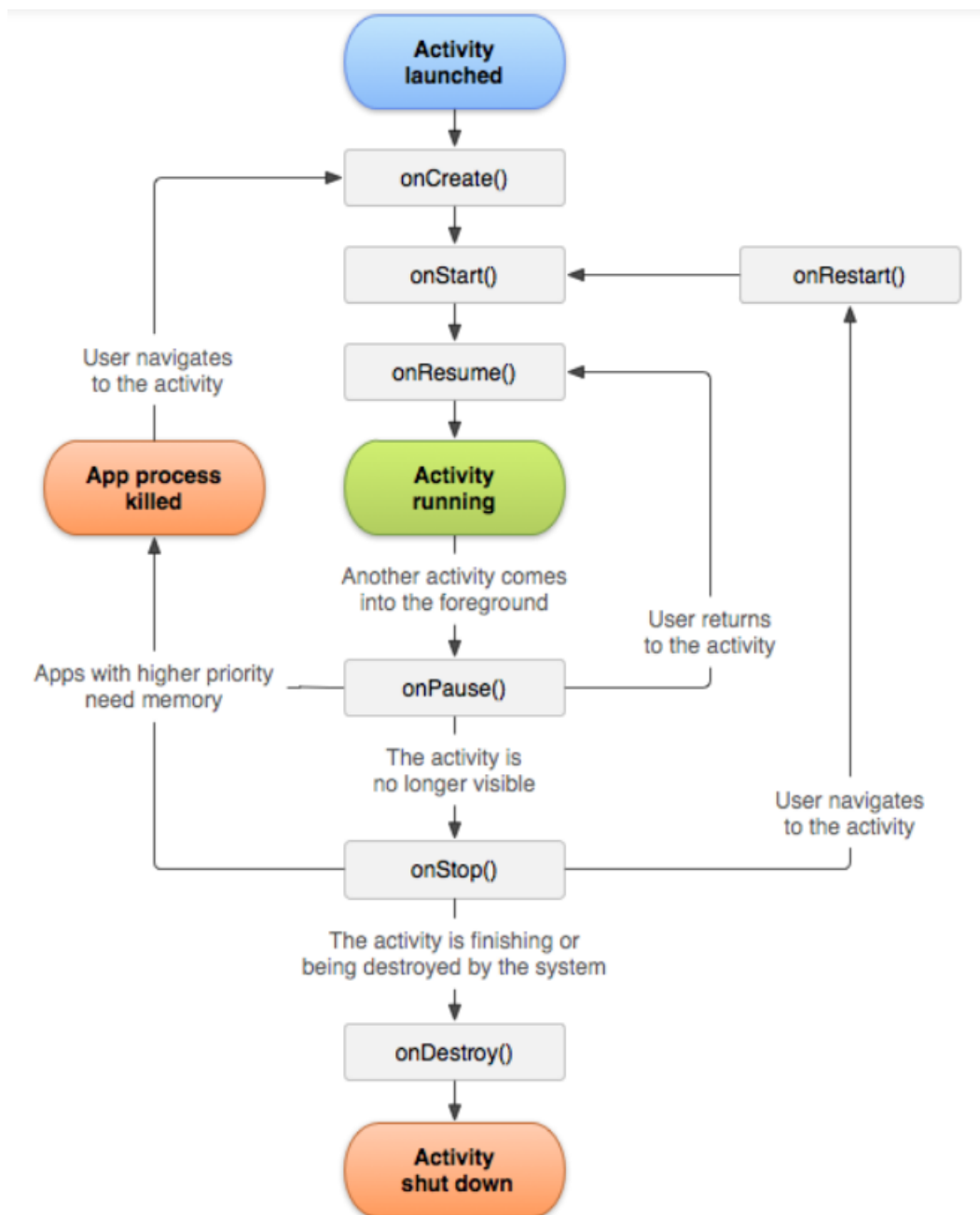


Figure 24 - Schéma explicatif d'un cycle de vie d'une activité

Ce schéma m'a grandement aidé à comprendre pourquoi certains bugs arrivaient notamment lors du `onStop()` que je pensais agir comme le `onDestroy()`. En effet, dans l'application était appelé un manager de la base de données qui, à un moment, doit être fermé. Ce dernier était à la base fermé dans le `onStop()` mais cela causait un problème : si l'on changeait d'application et que l'on revenait sur la nôtre, l'application plantait. Ainsi, ce schéma m'a permis de voir qu'il fallait agir sur le `onDestroy()` pour ce genre d'appel.

## 2 Base de données

La base de données a été implémentée avec SQLite version 3.19. Pour rappel, cette base est seulement constituée d'une table : la table Record. Cette dernière permet de stocker le nom du fichier audio créé, le chemin vers celui-ci ainsi que les caractéristiques de la voix calculés (fréquence fondamentale, shimmer et jitter).

Ainsi, pour simplifier la génération de la base, nous avons créé une classe qui hérite d'une classe SQLiteOpenHelper permettant de créer une base de données SQLite. Dans cette classe sont définis le nom de la base, la version de la base et le nom de la table :

```
/**
 * Class extending SQLiteOpenHelper for easier database generation and recovery.
 */
public class DBHelper extends SQLiteOpenHelper {

    /**
     * Constant representing the database's version.
     */
    private static int DB_VERSION = 1;

    /**
     * Constant representing the database's name.
     */
    private static final String DB_NAME = "voiceDatabase.db";

    /**
     * Constant representing the table's name containing the records.
     */
    static final String TABLE_NAME = "Records";
```

Figure 25 - Classe Helper pour créer la base de données SQLite

Ensuite, pour créer ce helper nous aurons besoin du contexte. Le contexte en Android est une interface permettant de stocker des informations globales dans l'environnement de l'application. Cela permet d'accéder à certaines ressources spécifiques. Pour construire ce helper, nous utiliserons le constructeur de SQLiteOpenHelper via un super().

```
/**
 * DBHelper sole builder.
 *
 * @param context the context to use for locating paths to the database
 */
public DBHelper(Context context) { super(context, DB_NAME, factory: null, DB_VERSION); }
```

Figure 26 - Constructeur de l'Helper

Une méthode de SQLiteOpenHelper a ensuite été éditée pour pouvoir créer notre base. Cette méthode, onCreate() va initialiser la base de donnée avec les bonnes tables comme ceci :

```

/**
 * Override the method for initializing the dataBase
 *
 * i.e. : creates the table containing the records if it doesn't already exist.
 *
 * @param db the database where the table needs to be created
 */
@Override
public void onCreate(SQLiteDatabase db) {
    String sql = "create table if not exists " + TABLE_NAME +
        " (Id integer primary key, Name text, Path text, Jitter real, Shimmer real, F0 real)";
    db.execSQL(sql);
}

```

Figure 27 - Méthode onCreate de l'Helper permettant de créer la base de données

La chaîne de caractères « sql » est une simple requête SQL permettant de créer une table avec les attributs définis.

La base de données étant maintenant créée, nous avons créé une autre classe permettant cette fois-ci d'interagir avec celle-ci : c'est une classe manager. Cette classe a donc comme attributs la classe helper précédemment créée et la base de données obtenue depuis cette dernière pour pouvoir la manipuler.

```

/**
 * The database helper.
 */
private DBHelper helper;

/**
 * The database.
 */
private SQLiteDatabase db;

/**
 * DBManager sole builder.
 *
 * @param context the context for database helper generation
 */
public DBManager(Context context) {
    helper = new DBHelper( context );
    db = helper.getWritableDatabase();
    helper.onCreate( db );
}

```

Figure 28 - Classe DBManager permettant de manipuler la base de données

Plusieurs manipulations communes ont été implémentées pour cette base de données :

- L'ajout d'enregistrement
- La requête des différents enregistrements présents dans la base de données
- La suppression d'un enregistrement précis :
- La modification des caractéristiques de la voix sur un enregistrement précis
- La requête d'un enregistrement précis

- La fermeture de la connexion

Pour faciliter la lecture de ce document, nous allons ici présenter seulement la méthode d'ajout et celle permettant d'avoir les différents enregistrements stockés car après cela nous aurons vu comment fonctionne le requêtage SQLite en Java. Le code des autres méthodes est disponible dans le projet dans la classe DBManager.

La méthode d'ajout se présente donc comme ceci :

```
/**
 * Add the record into dataBase.
 *
 * @param record the record to add
 * @return true if the record is added else return false
 */
public boolean add(Record record) {

    boolean isAdded;
    db.beginTransaction();
    try
    {
        // Create a new map of values, where column names are the keys
        ContentValues values = new ContentValues();
        values.put("Name", record.getName());
        values.put("Path", record.getPath());
        values.put("Jitter", record.getJitter());
        values.put("Shimmer", record.getShimmer());
        values.put("F0", record.getF0());

        db.insertOrThrow(TABLE_NAME, nullColumnHack: null, values);
        db.setTransactionSuccessful();
        isAdded = true;
    }
    catch(Exception e)
    {
        isAdded = false;
    }
    finally
    {
        db.endTransaction();
    }
    return isAdded;
}
```

Figure 29 - Méthode d'ajout d'un enregistrement dans la base de données

Le requêtage SQLite passe par des transactions que l'on doit commencer et finir. Ces transactions assurent d'avoir une seule connexion à la base de données pour chaque requête. Ici, la méthode d'ajout implique un SQL INSERT implémenté dans la librairie SQLite via la méthode insertOrThrow(). Cette méthode nécessite une structure assez particulière qui est le ContentValues qui est un type de Map où les clés sont en fait les noms des colonnes de la table.

Concernant maintenant la méthode permettant de récupérer les enregistrements, nous utiliserons un Cursor qui est en fait un moyen de parcourir un ensemble provenant d'une base de données. Cela se présente comme ceci :

```
/**
 * Get all the records in the dataBase
 *
 * @return the list of all the Records
 */
public List<Record> query() {
    ArrayList<Record> records = new ArrayList<>();
    Cursor c = queryTheCursor();
    while ( c.moveToNext() ) {
        Record record = new Record();
        record.setName( c.getString( c.getColumnIndex( columnName: "Name" ) ) );
        record.setPath( c.getString( c.getColumnIndex( columnName: "Path" ) ) );
        record.setJitter( c.getDouble( c.getColumnIndex( columnName: "Jitter" ) ) );
        record.setShimmer( c.getDouble( c.getColumnIndex( columnName: "Shimmer" ) ) );
        record.setF0( c.getDouble( c.getColumnIndex( columnName: "F0" ) ) );
        records.add( record );
    }
    c.close();
    return records;
}
```

Figure 30 - Méthode permettant de récupérer tous les enregistrements de la base de données

Comme nous pouvons constater, le Cursor permet d'obtenir un objet à un indice de colonne précis via la méthode getColumnIndex(). De plus, le Cursor implémente une méthode moveToNext() qui renvoie vrai si une entité est trouvée dans la base.

### 3 Interface

Avant de commencer cette partie, je tiens à préciser que ce sont les étudiants en 4<sup>e</sup> année en charge de ce projet dans le cadre du cours de génie logiciel qui ont fait l'interface. Des modifications mineures ont ensuite été ajoutées par moi-même.

Maintenant, passons à la description de l'interface. L'interface utilise les modules communs d'Android et une librairie externe qu'est MPAndroidChart en version 3.1.0.

Conformément aux maquettes proposées (voir Annexes), voici les interfaces créées :

- La page d'accueil :

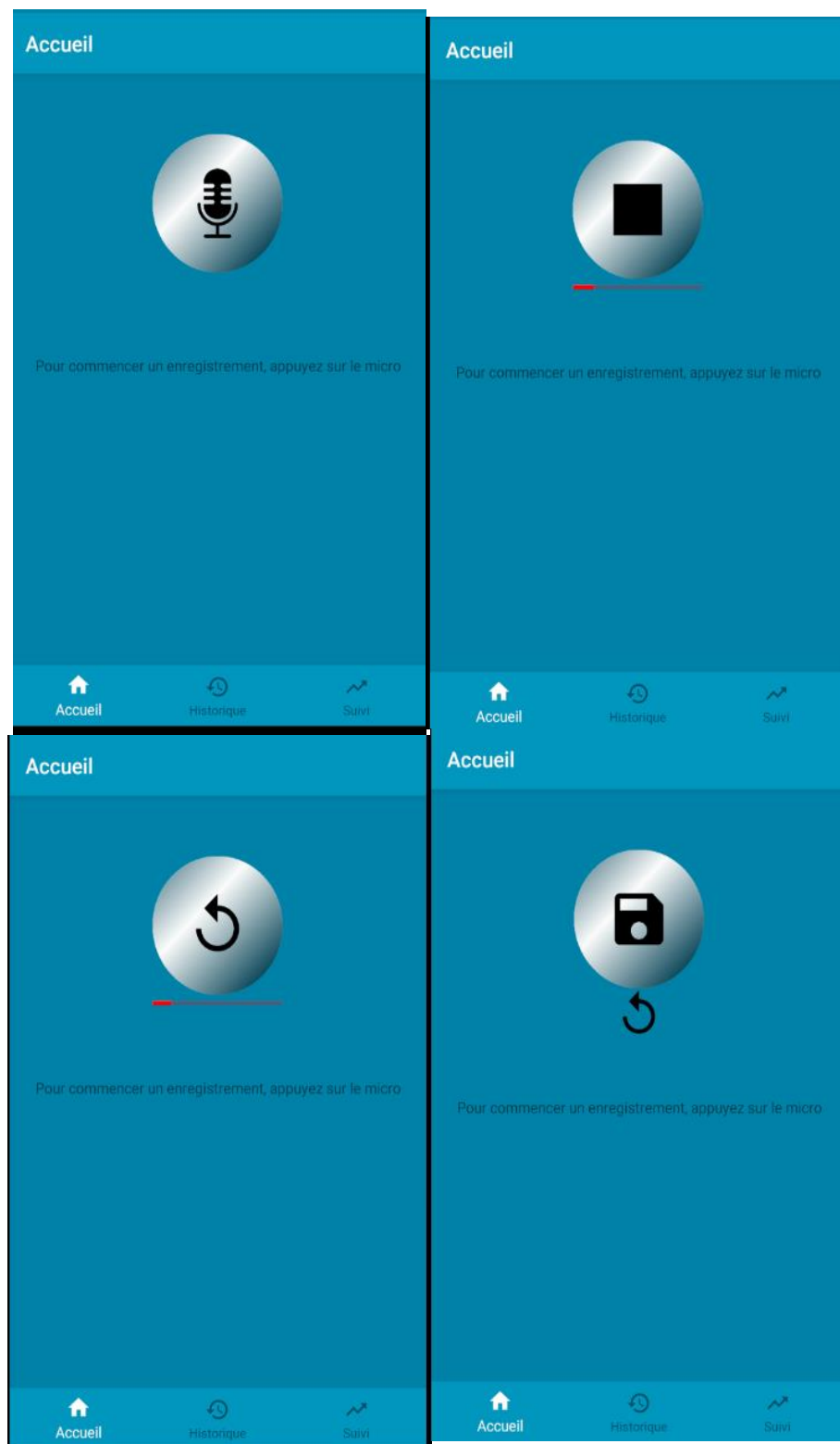


Figure 31 - Différentes formes de l'écran d'accueil

Simple, cette interface ne comporte qu'un champ texte, une barre de progression (invisible par défaut) et un bouton changeant de forme pour l'enregistrement.

Comme dit précédemment, le bouton affiché possède quatre formes : la forme par défaut, celle qui est visible en haut à gauche, la forme « Stop » lorsque l'enregistrement est en cours (en haut à droite), la forme « Reset » lorsque l'enregistrement est arrêté ou terminé (en bas à gauche) et enfin la forme « Enregistrer » avec un bouton de réinitialisation qui vient remplacer la barre de progression (en bas à droite).

Pour ce faire, nous avons dû créer une énumération appelée « Status\_mic » comme ceci :

```
/**
 * Status of the mic button
 */
public enum Status_mic {
    DEFAULT,
    RECORDING,
    CANCELED,
    FINISH;
}
```

Figure 32 - Enumération contenant les différents états du bouton micro de la page d'accueil

Cette énumération va permettre de connaître l'état dans lequel on est pour faire telle ou telle action. Cette page d'accueil étant destinée à l'enregistrement, lorsque le statut sera par défaut alors le bouton devient cliquable pour passer au statut « Recording », ce statut va initialiser la barre de progression et d'enregistrer la voix en temps réel. Si l'on clique sur le bouton Stop en plein enregistrement, on est en statut « Canceled », ce qui va arrêter l'enregistrement. Enfin, lorsque l'enregistrement est terminé, on arrive en statut « Finish » qui va permettre de sauvegarder l'enregistrement dans un fichier de format WAV, d'analyser ce même fichier audio créé dans la partie « Recording » pour extraire les caractéristiques de la voix et de créer une notification cliquable permettant de lire l'enregistrement directement sans aller chercher dans les fichiers.

Maintenant, passons à l'explication de l'implémentation de chacune de ces fonctionnalités :

- Pour le cas « Recording », la barre de progression progresse durant 5 secondes grâce à cela :

```
while (endTime - startTime < 5000000000L && status_mic_button == Status_mic.RECORDING) {
    progressBar.setProgress(Math.round((endTime - startTime) / 500000000f));
    try {
        Thread.sleep( millis: 10);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    endTime = System.nanoTime();
}
```

Figure 33 - Progression de la barre de progression via un Thread

Cette boucle while est incorporée dans un thread qui sera lancé dès lors que l'on appuie sur le bouton.

- Pour le cas « Recording », l'enregistrement de la voix sera expliqué dans une partie dédiée située plus loin dans ce document.
- Pour le cas « Finish », la sauvegarde de l'enregistrement dans un fichier WAV se fait comme ceci :

```

/**
 * Saves files.
 */
public void save() {
    DateFormat dateFormat = new SimpleDateFormat( pattern: "dd-MM-yyyy HH-mm-ss" );
    Date currentDate = new Date( System.currentTimeMillis() );
    fileName = dateFormat.format( currentDate );
    String newPath = FILE_PATH + File.separator + fileName + ".wav";

    if ( renameFile( finalPath, newPath ) ) {
        setFinalPath( newPath );
        addRecordDB( fileName, newPath );
    }
}

```

Figure 34 - Sauvegarde d'un enregistrement dans la base de données et sur le téléphone

Un fichier à la date du jour dans un format jour-mois-année heure-minute-seconde est créé et ce nom de fichier est ajouté à la base de données via la méthode addRecordDB() qui appelle la fonction add() vue précédemment dans le manager de la base.

- Pour le cas « Finish », l'analyse sera aussi expliquée dans une partie ultérieure de ce document.
- Pour le cas « Finish », la notification cliquable est effectuée à partir d'un NotificationManager propre à Android :

```

* Creates a notification about the recording's path
*/
public void createRecordingNotification()
{
    // Create the NotificationChannel, but only on API 26+ because
    // the NotificationChannel class is new and not in the support library
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        CharSequence name = "LarynxApp";
        int importance = NotificationManager.IMPORTANCE_DEFAULT;
        NotificationChannel channel = new NotificationChannel( "LarynxChannel", name, importance);
        // Register the channel with the system; you can't change the importance
        // or other notification behaviors after this
        NotificationManager notificationManager = getContext().getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channel);
    }

    String pathForNotification = finalPath.substring(finalPath.indexOf("voiceRecords/"));
    Intent intent = new Intent();
    File recordFile = new File(finalPath);
    intent.setAction(Intent.ACTION_VIEW);
    intent.setDataAndType(FileProvider.getUriForFile(this.getActivity(), authority: this.getActivity().getPackageName() + ".provider", recordFile), type: "audio/*");
    intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
    PendingIntent contentIntent = PendingIntent.getActivity(this.getActivity(), requestCode: 0, intent, PendingIntent.FLAG_CANCEL_CURRENT);
    Notification notification = new NotificationCompat.Builder(this.getActivity(), channelId: "LarynxChannel")
        .setSmallIcon(R.drawable.bouton_micro)
        .setContentTitle("Fichier enregistré sur : ")
        .setContentText(pathForNotification)
        .setContentIntent(contentIntent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setOngoing(false)
        .build();

    notification.flags |= Notification.FLAG_AUTO_CANCEL;

    mNotificationManager = (NotificationManager) getContext().getSystemService(Context.NOTIFICATION_SERVICE);
    mNotificationManager.notify(0, notification);
}

```

Figure 35 - Méthode permettant de créer une notification

Une notification en Android nécessite plusieurs choses : un channel correspondant à l'endroit où l'on veut envoyer la notification et un manager permettant de manipuler la notification. Une fois ceci créé, pour rendre la notification cliquable il faut créer un intent qui, en Android, correspond à une action effectuée par un module (qui sera donc ici la notification). Cet Intent est ici configuré de telle sorte à ce qu'il aille chercher le fichier de type audio dans le dossier où il est enregistré.

Une fois cet intent fait, il faut l'ajouter à la notification en cours de construction. Enfin, il suffit d'appeler le manager et de lancer un appel à la méthode notify() pour lancer la notification sur le téléphone.

Lorsque la notification est cliquée, le lecteur audio par défaut du téléphone s'ouvre et l'enregistrement est écoutable.

La notification se présente comme ceci :



Figure 36 - Affichage de la notification cliquable

**Remarque** : la notification est un bonus et n'est pas spécifié dans le cahier de spécifications, de plus, cette fonctionnalité est implémentée pour fonctionner sur les nouveaux appareils seulement.

- Concernant l'interface complète, nous remarquons en bas la présence d'onglets. Ces onglets sont ce que l'on appelle des Fragment en Android. Ces Fragment sont des sous-activités. Ici chaque page correspond donc à un Fragment.

- La page d'historique :

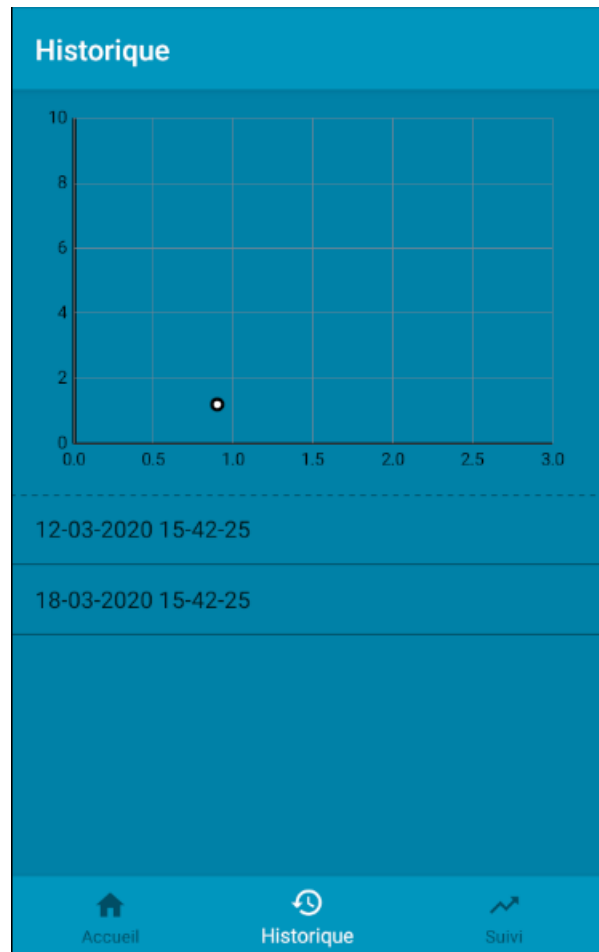


Figure 37 - Page d'historique

Cette page est constituée d'un graphique MPAndroidChart et d'une liste des différents enregistrements effectués. Cette liste est déroulante et triée par date. A chaque appui sur l'un des éléments de la liste, le point correspondant est affiché. Ce point désigne la valeur de shimmer en ordonnée et la valeur du jitter en abscisse. Ainsi, il est facile de trouver les différentes valeurs du shimmer et du jitter rapidement.

Avant de passer à la présentation et l'explication de l'utilisation de MPAndroidChart, nous allons voir comment a été implémentée la liste.

Pour personnaliser une liste, il faut passer par un adaptateur. Cet adaptateur, qui se comporte comme une liste d'objets, doit comprendre la liste des enregistrements présents dans la base de données. Ainsi, nous obtenons le code suivant :

```
final ArrayAdapter<Record> adapter = new ArrayAdapter<>(getActivity().getApplicationContext(),
    android.R.layout.simple_list_item_1, records);
listview.setAdapter(adapter);
```

Figure 38 - Adapter permettant de personnaliser la liste pour contenir des objets Record

Où records est la liste des enregistrements récupérés dans la base de données.

Passons maintenant à la présentation de MPAndroidChart. MPAndroidChart est une librairie open-source proposée par PhilJay qui permet de gérer différents types de graphes : des histogrammes, des graphes linéaires à points, des diagrammes circulaires, etc...

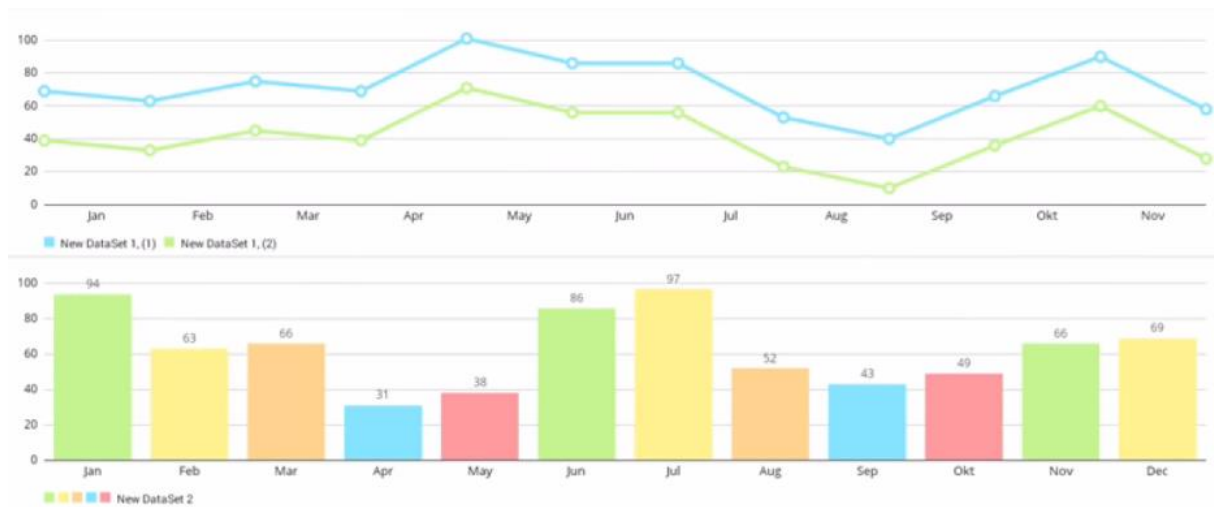


Figure 39 - Exemple de graphes créés via MPAndroidChart

Dans notre cas, nous allons utiliser des graphes linéaires aussi appelés « LineChart » dans la librairie. Ce LineChart est composé de différents dataset, chaque dataset représentant un point dans le graphe. A partir de ces datasets, il est possible d'ajouter des données au graphique. Tout ceci se résume comme ceci dans le code (initialisation du graphe en vérifiant s'il existe des enregistrements présents) :

```
if(!records.isEmpty() ){
    LineDataSet lineDataSet = new LineDataSet(dataValues(records.get(0)), records.get(0).getName());
    setLineData(lineDataSet);
    datasets.add((lineDataSet));
    final LineData data = new LineData(datasets);
    mpLineChart.setData(data);
}
```

Figure 40 - Ajout de données au graphique

Où dataValues est une fonction allant faire une requête pour récupérer les valeurs des caractéristiques de la voix pour l'enregistrement précis.

Une fois le graphique construit, il faut l'afficher. Pour ce faire, nous utilisons ces trois lignes suivantes :

```
setChart(mpLineChart);
mpLineChart.setDrawGridBackground(false);
mpLineChart.invalidate();
```

Figure 41 - Affichage du graphe

Où setChart est une fonction créée pour configurer le graphique en initialisant l'échelle des axes, leur position ainsi que leur taille :

```
/**
 * Set the graphic feature of the line chart
 * @param chart the chart to be set
 */
private void setChart(LineChart chart){

    YAxis yAxis = chart.getAxisLeft();
    XAxis xAxis = chart.getXAxis();

    chart.getAxisRight().setEnabled(false);

    //Set the y axis property
    yAxis.setAxisLineWidth(2.5f);
    yAxis.setAxisLineColor(Color.BLACK);
    yAxis.setAxisMinimum(0f);
    yAxis.setAxisMaximum(10f);
    yAxis.setTextSize(12f);

    //Set the x axis property
    xAxis.setAxisLineWidth(2f);
    xAxis.setAxisLineColor(Color.BLACK);
    xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
    xAxis.setAxisMinimum(0f);
    xAxis.setAxisMaximum(3f);
    xAxis.setTextSize(12f);

    chart.getLegend().setEnabled(false);
    chart.getDescription().setEnabled(false);
}
```

Figure 42 - Méthode permettant de configurer un graphique

- La page de suivi :

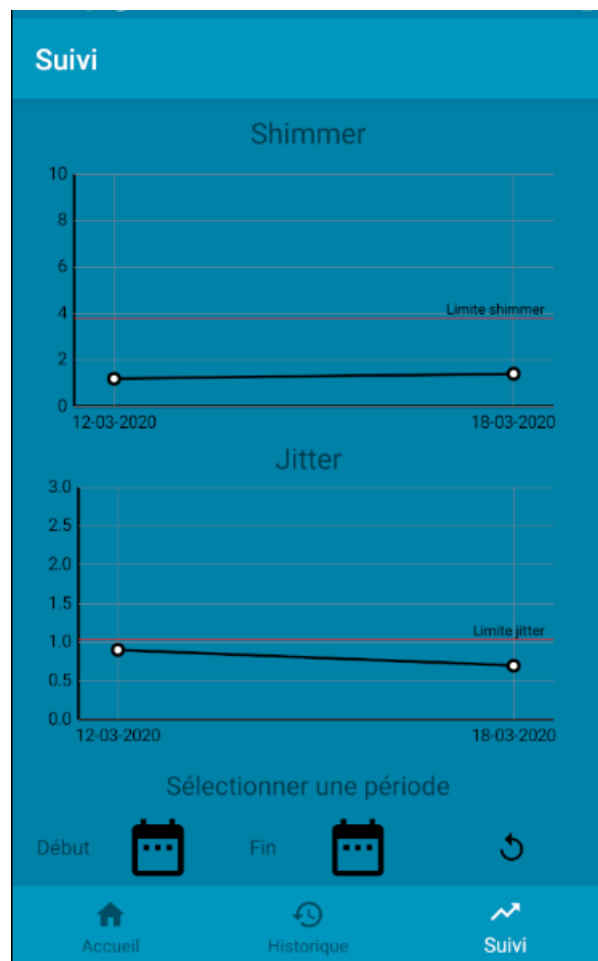


Figure 43 - Page de suivi

Cette page permet de suivre l'état de notre voix avec une indication en rouge représentant les limites à ne pas dépasser avant que cela devienne motif de consultation chez un ORL. Comme la page précédente, elle incorpore des graphiques réalisés avec MPAndroidChart qui ont ici deux spécificités : la ligne limite en rouge et les dates en abscisse.

Pour les lignes limites en rouge, il a suffi d'ajouter des LimitLine comme ceci :

```
LimitLine jitterLl = new LimitLine(1.04f);
jitterLl.setLabel("Limite jitter");
jitterLl.setLineColor(Color.RED);
jitterMpLineChart.getAxisLeft().addLimitLine(jitterLl);
```

Figure 44 - Ajout de ligne limites à ne pas franchir indiquant les seuils pathologiques

Ces objets LimitLine sont spécialement conçus pour l'utilisation qu'on en fait, c'est-à-dire indiquer une limite à ne pas dépasser. La limite ici a été placée au seuil pathologique de chaque caractéristique.

Pour afficher les dates en abscisse, il a fallu passer par un Formatter personnalisé sur l'axe X. Cela se présente comme ceci :

```
XAxis jitterXAxis = jitterMpLineChart.getXAxis();
jitterXAxis.setGranularity(1f);
jitterXAxis.setSpaceMax(0.1f);
jitterXAxis.setSpaceMin(0.1f);
jitterXAxis.setValueFormatter(new com.github.mikephil.charting.formatter.IndexAxisValueFormatter(dateValues()));
```

Figure 45 - Affichage des dates en abscisse

Où `dateValues()` est une fonction permettant de récupérer les noms de chaque enregistrement affiché (étant donné que les noms comportent la date) et de les modifier pour retirer seulement la date sans l'heure comme ceci :

```
/**
 * Initialisation of the dates arraylist
 * @return the dates arraylist
 */
private String[] dateValues(){
    ArrayList<String> dates = new ArrayList<>();
    for(int i = 0; i < records.size(); i++){
        {
            String strippedName = records.get(i).getName().replace( target: "-", replacement: " ");
            String[] dateTimes = strippedName.split( regex: " ");
            dates.add(i, element: dateTimes[0] + "-" + dateTimes[1] + "-" + dateTimes[2]);
        }
    }
    return dates.toArray(new String[0]);
}
```

Figure 46 - Méthode pour récupérer les dates à partir des noms des fichiers enregistrés

Etant donné que les noms sont enregistrés en jour-mois-année heure-minute-seconde, il a suffi de remplacer les « - » par des espaces et ensuite extraire chaque nombre séparé par un espace pour obtenir la date sans l'heure.

Revenons maintenant sur l'interface de suivi. On remarque qu'il existe des calendriers permettant de sélectionner une période dans le temps et de n'afficher que les points compris dans cette dite période. Le problème est que, par manque de temps, la fonctionnalité de filtrage par période n'a pas été implémentée.

Autre remarque, si le nombre d'enregistrements se trouve trop élevé il va y avoir des soucis d'affichage qui n'ont pas été réglé encore une fois par manque de temps.

## 4 Analyse de la voix et enregistrement

Dans cette partie, nous allons présenter la librairie utilisée pour récupérer les fréquences de la voix en temps réel qui permet aussi d'enregistrer une entrée audio dans un fichier sur le téléphone puis nous allons parler des différentes méthodes permettant de calculer les caractéristiques de la voix.

Passons donc dans un premier temps sur la présentation d'une librairie citée dans la Partie 4 Analyse et Conception qui est TarsosDSP.

TarsosDSP est une librairie open-source de traitement de signal (<https://github.com/JorenSix/TarsosDSP>). Elle permet notamment de rendre plus simple des tâches comme l'enregistrement ou la détection de « pitch » (fréquence de la voix) à un moment t. Ici, c'est pour ces deux tâches qu'elle sera utilisée.

La librairie est trop complète pour être expliquée en détails ici mais nous allons tout de même nous attarder sur les éléments utilisés.

TarsosDSP permet d'avoir un AudioDispatcher qui va regrouper plusieurs processus et qui va les lancer en même temps pour nous. Cet AudioDispatcher peut être créé à partir d'un AudioDispatcherFactory comprenant deux méthodes : fromDefaultMicrophone() et fromPipe(). La méthode fromDefaultMicrophone() permet de créer un dispatcher qui écouterait le microphone du téléphone et qui effectuerait les traitements en temps réel. La méthode fromPipe(), quant à elle, permet de créer un dispatcher qui effectuerait des traitements à partir d'un fichier pré-enregistré. À noter que cette dernière méthode nécessite l'installation d'un encodeur FFMpeg.

Dans notre cas, nous allons utiliser un dispatcher temps réel qui utilisera donc le microphone de l'appareil comme ceci :

```
dispatcher = AudioDispatcherFactory.fromDefaultMicrophone( f: 44100, i1: 2048, i2: 0 );
```

Figure 47 - Dispatcher utilisant le microphone par défaut du téléphone

Les paramètres correspondent respectivement à la fréquence d'échantillonnage, la taille du buffer et la partie « skipée » du buffer. Ici, nous avons donc une fréquence d'échantillonnage classique de 44.1kHz et une taille de buffer de 2048. Ce buffer va servir à garder les différents échantillons créés lors de l'enregistrement.

Maintenant que nous avons vu comment le dispatcher se comporte, allons voir du côté des processus que ce dispatcher permet de gérer. Ces processus se nomment des AudioProcessor.

Plusieurs AudioProcessor sont disponibles dans la librairie mais les deux principaux qui nous intéressent sont le WriterProcessor et le PitchProcessor.

Le WriterProcessor permet d'écrire un fichier contenant l'enregistrement audio. Il prend en paramètre le fichier de sortie et le format dans lequel nous souhaitons écrire.

```
AudioProcessor recordProcessor = new WriterProcessor(AUDIO_FORMAT, randomAccessFile);
dispatcher.addAudioProcessor(recordProcessor);
```

Figure 48 - Processus permettant d'écrire les données captées par le microphone (données audio) dans un fichier

Le format est un peu spécifique étant donné que c'est un TarsosDSPAudioFormat qui est en fait un AudioFormat classique d'Android mais qui permet d'indiquer respectivement la fréquence d'échantillonnage, le nombre de bits pour chaque échantillon, le nombre de canaux (1 pour mono, 2 pour stéréo), si les données sont signées ou non et dans quel ordre les bytes doivent être enregistrés.

```
AUDIO_FORMAT = new TarsosDSPAudioFormat(
    v: 44100,
    i: 16,
    il: 1,
    b: true,
    ByteOrder.LITTLE_ENDIAN.equals(ByteOrder.nativeOrder()));
```

Figure 49 - Format de l'audio

Le PitchProcessor permet de détecter la fréquence de la voix à chaque instant et est configurable de telle sorte à ce qu'il puisse utiliser différents algorithmes dont l'algorithme Yin qui est utilisé ici.

```
PitchDetectionHandler pitchDetectionHandler = (res, e) -> {
    float pitchInHz = res.getPitch();
    if (pitchInHz != -1 && pitchInHz < 400)
        pitches.add(pitchInHz);
};

AudioProcessor pitchProcessor = new PitchProcessor(new Yin(audioSampleRate: 44100, bufferSize: 2048),
    sampleRate: 44100, bufferSize: 2048, pitchDetectionHandler);
dispatcher.addAudioProcessor(pitchProcessor);
```

Figure 50 - Processus permettant de détecter les fréquences de la voix en temps réel en utilisant l'algorithme Yin

Comme on le voit ci-dessus, le PitchProcessor prend en compte un handler qui est la partie qui va détecter la fréquence à chaque instant. Ici, notre handler est configuré pour stocker les différentes fréquences dans une liste utilisable pour l'analyse. Comme dit précédemment, ce processor prend en paramètre un algorithme, la fréquence d'échantillonnage, la taille du buffer et un handler.

Pour lancer le dispatcher, il suffit ensuite de faire un run() comme pour toute classe implémentant Runnable.

**NB** : à chaque fois que l'enregistrement est terminé, le dispatcher est stop() et détruit pour éviter des conflits de thread.

## 4.2 Méthodes de calcul pour les caractéristiques de la voix

Pour réaliser les différents calculs du jitter et du shimmer, nous nous sommes basés sur les algorithmes déjà présents et produits par les anciens étudiants chargés du projet (voir Figure 72 - Algorithme de calcul de shimmer existant et Figure 73 - Algorithme de calcul de jitter existant). Quelques légères modifications ont été apportées sur le calcul du shimmer qui, de base, divisait par la taille de la liste contenant les amplitudes pics à pic et non pas par la taille de la liste contenant les périodes pour correspondre aux équations présentées dans la partie 1 de l'état de l'art.

## 4.3 Implémentation de l'algorithme Yin

Pour implémenter l'algorithme de Yin et correspondre aux besoins de TarsosDSP, nous avons créé une classe implémentant l'interface PitchDetector permettant de détecter les fréquences en temps réel.

```
public final class Yin implements PitchDetector {
```

Figure 51 - Classe Yin implémentant l'interface PitchDetector

Cette interface nécessite l'implémentation d'une méthode appelée `getPitch()`, cette méthode va retourner le résultat de la détection de fréquence (`PitchDetectionResult`). Dans cette méthode, l'algorithme de Yin est implémenté avec toutes ses étapes (voir Partie Analyse et Conception – Algorithme Yin).

```
public PitchDetectionResult getPitch(final float[] audioBuffer) {  
  
    final int tauEstimate;  
    final float pitchInHertz;  
  
    // step 2  
    difference(audioBuffer);  
  
    // step 3  
    cumulativeMeanNormalizedDifference();  
  
    // step 4  
    tauEstimate = absoluteThreshold();  
  
    // step 5  
    if (tauEstimate != -1) {  
        final float betterTau = parabolicInterpolation(tauEstimate);  
  
        // conversion to Hz  
        pitchInHertz = sampleRate / betterTau;  
    } else {  
        // no pitch found  
        pitchInHertz = -1;  
    }  
  
    result.setPitch(pitchInHertz);  
  
    return result;  
}
```

Figure 52 - Fonction `getPitch()` implémentant l'algorithme Yin

Maintenant, nous allons présenter chaque fonction utilisée dans ce `getPitch()`. Tout d'abord la fonction `difference`, qui correspond à l'étape 2 de l'algorithme de Yin :

```

/**
 * Implements the difference function as described in step 2 of the YIN
 * paper.
 */
private void difference(final float[] audioBuffer) {
    int index;
    int tau;
    float delta;
    for (tau = 0; tau < yinBuffer.length; tau++) {
        yinBuffer[tau] = 0;
    }
    for (tau = 1; tau < yinBuffer.length; tau++) {
        for (index = 0; index < yinBuffer.length; index++) {
            delta = audioBuffer[index] - audioBuffer[index + tau];
            yinBuffer[tau] += delta * delta;
        }
    }
}

```

Figure 53 - Fonction difference() équivalent à l'étape 2 de l'algorithme Yin

Un tableau de taille bufferSize/2 est initialisé à 0 et va ensuite contenir les delta entre les valeurs du signal et de lui-même décalé d'un temps tau.

**NB** : L'autocorrélation ici n'est pas réimplémentée car elle n'est pas nécessaire pour la suite du déroulement de l'algorithme. Dans l'algorithme présenté, l'autocorrélation correspond à l'étape 1 mais c'est en réalité pour montrer à quel point l'autocorrélation peut être amélioré mais elle n'est pas utilisée dans l'algorithme.

Puis, la fonction cumulativeMeanNormalizedDifference() qui réalise l'étape 3 de l'algorithme de Yin :

```

/**
 * The cumulative mean normalized difference function as described in step 3
 * of the YIN paper.
 */
private void cumulativeMeanNormalizedDifference() {
    int tau;
    yinBuffer[0] = 1;
    float runningSum = 0;
    for (tau = 1; tau < yinBuffer.length; tau++) {
        runningSum += yinBuffer[tau];
        yinBuffer[tau] *= tau / runningSum;
    }
}

```

Figure 54 - Fonction cumulativeMeanNormalizedDifference() équivalent à l'étape 3 de l'algorithme Yin

L'étape 4, le calcul du seuil absolu est implémenté comme ceci :

```

/**
 * Implements step 4 of the AUDIO_YIN paper.
 */
private int absoluteThreshold() {
    int tau;
    // first two positions in yinBuffer are always 1
    // So start at the third (index 2)
    for (tau = 2; tau < yinBuffer.length; tau++) {
        if (yinBuffer[tau] < threshold) {
            while (tau + 1 < yinBuffer.length && yinBuffer[tau + 1] < yinBuffer[tau]) {
                tau++;
            }
            // found tau, exit loop and return
            // store the probability
            // From the YIN paper: The threshold determines the list of
            // candidates admitted to the set, and can be interpreted as the
            // proportion of aperiodic power tolerated
            // within a periodic signal.
            //
            // Since we want the periodicity and not aperiodicity:
            // periodicity = 1 - aperiodicity
            result.setProbability(1 - yinBuffer[tau]);
            break;
        }
    }

    // if no pitch found, tau => -1
    if (tau == yinBuffer.length || yinBuffer[tau] >= threshold) {
        tau = -1;
        result.setProbability(0);
        result.setPitched(false);
    } else {
        result.setPitched(true);
    }

    return tau;
}

```

Figure 55 - Fonction `absoluteThreshold()` équivalant à l'étape 4 de l'algorithme Yin

Cette fonction parcourt le `yinBuffer` (buffer stockant les résultats des fonctions précédentes) et recherche le `tau` correspondant à la valeur de différence la plus faible.

Si aucune fréquence n'est trouvée, renvoie -1.

Maintenant, l'étape finale ici, l'interpolation parabolique qui correspond à l'étape 5 de l'algorithme.

```

/**
 * Implements step 5 of the AUDIO_YIN paper. It refines the estimated tau
 * value using parabolic interpolation. This is needed to detect higher
 * frequencies more precisely.
 *
 * @param tauEstimate The estimated tau value.
 * @return A better, more precise tau value.
 */
private float parabolicInterpolation(final int tauEstimate) {
    final float betterTau;
    final int x0;
    final int x2;

    if (tauEstimate < 1) {
        x0 = tauEstimate;
    } else {
        x0 = tauEstimate - 1;
    }

    if (tauEstimate + 1 < yinBuffer.length) {
        x2 = tauEstimate + 1;
    } else {
        x2 = tauEstimate;
    }

    if (x0 == tauEstimate) {
        if (yinBuffer[tauEstimate] <= yinBuffer[x2]) {
            betterTau = tauEstimate;
        } else {
            betterTau = x2;
        }
    } else if (x2 == tauEstimate) {
        if (yinBuffer[tauEstimate] <= yinBuffer[x0]) {
            betterTau = tauEstimate;
        } else {
            betterTau = x0;
        }
    } else {
        float s0;
        float s1;
        float s2;
        s0 = yinBuffer[x0];
        s1 = yinBuffer[tauEstimate];
        s2 = yinBuffer[x2];
        betterTau = tauEstimate + (s2 - s0) / (2 * (2 * s1 - s2 - s0));
    }

    return betterTau;
}

```

Figure 56 - Fonction `parabolicInterpolation` équivalant à l'étape 5 de l'algorithme Yin

Ce code permet, grâce aux estimations faites auparavant dans la fonction `seuil`, de trouver par interpolation parabolique d'ordre 2 le meilleur seuil possible.

Maintenant passons aux performances de l'application vis-à-vis des résultats sortis par Praat.

Pour illustrer ici je n'ai malheureusement que mon cas étant donné les circonstances de confinement, je n'ai pas pu tester sur plusieurs personnes et voir si les valeurs coïncidaient.

Néanmoins voici tout de même un exemple des résultats qu'on peut obtenir avec l'application :

```
Jitter : 0.7854860681114552 Shimmer : 3.222886063616004 F0 : 110.22064208984375
```

Et maintenant, en prenant le même fichier audio, voici les résultats sortis par Praat :

```
Shimmer (dda): 3.987% Jitter (ddp): 0.805% Mean pitch: 110.180 Hz
```

On remarque une nette ressemblance dans les résultats obtenus, seul le shimmer varie un peu dépendamment de l'algorithme utilisé.

## 5 Permissions

Les différentes permissions utilisées par l'application sont les suivantes :

- Microphone, évidemment, pour accéder au micro du téléphone pour enregistrer la voix de l'utilisateur
- Stockage pour que les fichiers puissent être sauvegardés sur le téléphone de l'utilisateur

## 6 Tests

Des tests utilisant JUnit ont été effectués sur quasiment toutes les classes du modèle. La classe `FeaturesCalculatorTest` (servant à tester la classe `FeaturesCalculator`) doit créer un signal sinusoïdal pur pour ensuite vérifier les valeurs des caractéristiques calculées (F0, jitter et shimmer). Pour ces deux dernières caractéristiques, leurs valeurs doivent être nulles car il n'y a aucune variation en fréquence ou en amplitude car c'est un signal pur. Cette classe est créée mais les tests ne sont pas fonctionnels pour des problèmes de taille d'entrée.

Un test fonctionnel de l'application dans sa globalité a été fait avec Espresso qui est un plugin préinstallé sur Android Studio pour faire des tests fonctionnels à la manière d'un enregistrement de macro (on entre une suite de commande ou de clics et on demande ce qu'il devrait s'afficher ou se passer ensuite pour vérifier le fonctionnement de l'application).

Ce projet utilise un gestionnaire de version décentralisé qui est GitHub, le lien vers le projet est le suivant : <https://github.com/StevenMartin00/PRD-Voix>

Un système d'intégration continue a été implémenté via GitHub Actions, un service d'intégration continue fourni par GitHub. Ce service utilise un fichier YAML (.yml) pour gérer ses workflows de test. A partir de GitHub Actions, nous pouvons créer un fichier YAML de base pour Android ne contenant rien.

Ainsi, il a fallu définir plusieurs jobs dans ce YAML qui est le suivant :

---

```

name: Android CI

on: [push]

jobs:
  build:

    runs-on: macOS-latest

    steps:
      - name: checkout
        uses: actions/checkout@v2
      - name: set up JDK 1.8
        uses: actions/setup-java@v1
        with:
          java-version: 1.8
      - name: Build with Gradle
        run: chmod +x gradlew && ./gradlew build
      - name: Run tests
        run: ./gradlew test -PdisablePreDex --stacktrace
      - name: Run tests on Android emulator
        uses: reactivecircus/android-emulator-runner@v2
        with:
          api-level: 29
          script: ./gradlew connectedCheck
      #- name: Run code coverage (reports available in app/build/reports/coverage)
      # run: ./gradlew createDebugAndroidTestCoverageReport

```

---

Figure 57 - Fichier de configuration pour l'intégration continue via GitHub Actions

Ce fichier permet de définir la suite de jobs qui doit être lancée à chaque nouveau push sur le dépôt Git. Ici, pour l'environnement, nous allons utiliser macOS car c'est le seul environnement ayant les packages nécessaires pour faire tourner un émulateur Android.

Maintenant, passons aux différentes étapes :

- Set up JDK 1.8 : préparation du compilateur Java pour le prochain build
- Build with Gradle : permet de lancer le build du projet avec Gradle
- Run tests : permet de lancer les tests unitaires à partir de Gradle
- Run tests on Android emulator : permet de lancer les tests fonctionnels réalisés par Espresso sur un émulateur Android. Cet émulateur est le reactivecircus qui n'est compatible qu'avec macOS et utilise la même version d'Android qu'est utilisée pour le développement soit la version 29.
- Run code coverage, en commentaire ici, est un job permettant de créer un document décrivant la couverture de code de l'application. Cependant, ce job ne fonctionne pas car je n'arrive pas à initialiser JaCoCo (bibliothèque de couverture de code en Java) pour les tests unitaires sur Android (dossier test et non pas androidTest qui sont les tests fonctionnels).

## 1 Fait et Reste à faire

Les tâches que j'ai déjà faites sont :

- La recherche sur les différentes bibliothèques Java / Android pour du traitement de signal
- La recherche sur l'implémentation de réseaux de neurones pour fiabiliser l'application
- L'analyse de l'implémentation des algorithmes dans l'application existante et dans Praat
- La correction de quelques bugs minimes sur l'enregistrement
- La modélisation du projet existant
- La refonte de l'application sur un pattern MVC
- La conception d'une base de données optimisée pour des prises régulières de données datées
- La gestion de projet concernant ce projet : comment répartir les tâches et les diviser pour que les étudiants en 4<sup>e</sup> année aient aussi du travail
- La conception d'un meilleur algorithme pour calculer la période d'un signal
- La réalisation de ce document
- L'implémentation de l'algorithme créé
- L'implémentation de la base de données créée
- La réalisation des interfaces (cette tâche a été effectuée par les étudiants en 4<sup>e</sup> année)
- 80% des tests unitaires
- L'intégration continue
- Les tests fonctionnels (en fonction de la version d'Android)
- Des comparaisons avant / après implémentation du nouvel algorithme

Les tâches qu'il reste à faire sont :

- L'implémentation d'un tri d'enregistrement par période pour l'affichage sur graphes
- Tester sur différentes personnes (impossible à cause de la pandémie)
- Génération du document JaCoCo permettant d'avoir la couverture de code par les tests unitaires
- Les 20% de tests unitaires restants
- Faire valider toute l'implémentation par le médecin

Les tâches qui ont pris du retard sont principalement les tâches de rédaction de documents demandés et les tâches d'analyse et de compréhension d'algorithmes connus permettant de gérer des problèmes de calcul de période. En effet, côté documents à rendre, j'ai pris beaucoup de temps à écrire le cahier de spécifications car je devais mettre au clair les points principaux du projet avec le client qui n'est pas très souvent disponible. Puis, côté analyse et compréhension d'algorithmes connus, j'ai beaucoup trop perdu de temps sur la compréhension d'algorithmes peu fiables ou non adaptés au problème que nous avons ici. J'ai aussi perdu pas mal de temps sur la partie conception de l'interface réalisée par les étudiants en 4<sup>e</sup> année qui étaient à mes côtés et qui ont peiné à faire le travail en temps et en heure. Cette dernière partie a engendré un retard sur l'implémentation d'une liaison entre l'interface et la base de données pour tout ce qu'était enregistrement audio.

## 2 Organisation prévisionnelle

Le tableau Trello, disponible en annexe (voir Figure 62 - Trello (Tableau organisationnel des tâches)), liste les différentes tâches qu'il reste à faire. Certaines vont encore s'ajouter au fil du temps.

Le diagramme de Gantt, disponible en annexe (voir Figure 67 - Export Gantt prévisionnel, Page 5 (Diag. *Gantt*)), permet de voir que la partie concernant l'implémentation du nouvel algorithme et de la base de données ainsi que la création des interfaces a été planifiée pour le semestre prochain.

## 3 Organisation finale

Le tableau Trello est mis à jour et est toujours disponible via le même lien trouvable en annexe.

Le diagramme de Gantt final quant à lui est disponible en annexe (voir Annexe – Gantt final). Nous pouvons voir que des retards ont été engendrés par différentes tâches comparé à ce qui était prévu.

## 4 Bilan qualité

Côté qualité de la mise en œuvre, j'ai réalisé les documents attendus : la documentation développeur, la documentation utilisateur et le cahier de tests. J'ai mis en place un dépôt Git pour versionner mon projet (disponible ici : <https://github.com/StevenMartin00/PRD-Voix>) sur lequel a été installé GitHub Actions permettant de faire de l'intégration continue.

Ainsi, un fichier de configuration permettant de créer un workflow qui lance automatiquement les tests unitaires, les tests fonctionnels grâce à Espresso (qui est un plug-in d'Android Studio permettant d'enregistrer et de vérifier la sortie d'une séquence d'entrées produits sur l'interface de l'application) : ce fichier se nomme « android.yml » dans le dossier « .github/workflows ». Seul problème ici est que je n'ai pas réussi à créer le document JaCoCo concernant la couverture de code pour les tests unitaires sur Android.

Toute la documentation Javadoc a été générée et est fournie avec le projet. Enfin, pour le déploiement, étant donné que c'est un projet Android, il suffira d'installer un APK pour obtenir l'application. Cette APK n'est pas encore disponible via Google Play.

## 5 Bilan auto-critique sur la gestion de projet

Pour mon projet, j'ai été fier de pouvoir gagner la confiance du client au départ en restant à l'écoute de ses besoins tout en proposant des solutions pour son problème. De plus, j'ai su analyser, comprendre et implémenter des algorithmes complexes de recherche de période tel que l'algorithme Yin. Même lorsqu'une supposée solution qui se présentait à moi pour résoudre les problèmes impliqués par le projet ne fonctionnait pas j'ai su trouver des alternatives. Sur ce point, en revanche, j'ai aussi perdu énormément de temps à essayer d'implémenter des algorithmes non adaptés au problème. De plus, j'ai aussi perdu beaucoup de temps à comprendre les algorithmes même si finalement j'ai réussi à les comprendre. Enfin, je trouve que je n'ai pas été assez strict avec mes collègues étudiants quant au niveau des livrables attendus et des deadlines.

1. **Rairán, J. D.** Two algorithms for estimating the period of a discrete signal. [Online] [http://www.scielo.org.co/scielo.php?script=sci\\_arttext&pid=S0120-56092014000300010#t1](http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0120-56092014000300010#t1).
2. **Linda M. Carroll.** *The Voice Lab: Is it just numbers?*
3. *A Signal Period Detection Algorithm Based on.* **Wang, Zhao Han and Xiaoli.**
4. **Praat.** Voice. [Online] <http://www.fon.hum.uva.nl/praat/manual/Voice.html>.
5. **Mireia Farrús, Javier Hernando, Pascual Ejarque.** Jitter and Shimmer Measurements for Speaker Recognition. [Online] [https://www.cs.upc.edu/~nlp/papers/far\\_jit\\_07.pdf](https://www.cs.upc.edu/~nlp/papers/far_jit_07.pdf).
6. **VAUTHIER, Aurelien.** *Rapport stage 3e année.*
7. **Meisam Khalil Arjmandi, Mohammad Pooyan.** *An optimum algorithm in pathological voice quality assessment using.*
8. **Tianxue WANG, Wenli YAN.** *Rapport PRD.*
9. **Anas MONCEF, Lucien LE GUELLEC.** *Rapport Projet Genie Log.*
10. **Miltiadis Vasilakis, Yannis Stylianou.** *Spectral jitter modeling and estimation.*
11. **Stefan Hadjitodorov, Petar Mitev.** *A computer system for acoustic analysis of pathological voices and.*
12. **Meisam Khalil Arjmandi, Mohammad Pooyan, Mohammad Mikaili, Mansour Vali, Alireza Moqarehzadeh.** *Identification of Voice Disorders Using Long-Time.*
13. **De Cheveigné, Alain and Kawahara, Hideki.** YIN, a fundamental frequency estimator for speech and music. [Online] [http://audition.ens.fr/adc/pdf/2002\\_JASA\\_YIN.pdf](http://audition.ens.fr/adc/pdf/2002_JASA_YIN.pdf).

## 1 Description des interfaces externes du logiciel

### 1.1 Interfaces matériel/logiciel

Le matériel nécessaire est simplement un appareil Android. Aucune liaison au réseau n'est obligatoire. L'appareil Android doit disposer d'un microphone fonctionnel et d'assez d'espace pour sauvegarder les enregistrements. De plus, l'appareil Android doit être sous une version 9 ou supérieure car les autres versions n'ont pas encore été testées.

### 1.2 Interfaces hommes/machine

Les interfaces suivantes ont été réalisées par les étudiants en 4<sup>e</sup> année et ont été validées par le client :

- L'écran d'accueil correspond à l'écran affiché au lancement de l'application. Cet écran facilement modifiable (au niveau des couleurs par exemple) permet d'enregistrer sa voix en appuyant sur l'icône de microphone ou d'accéder aux différentes pages.



Figure 58 - Nouvel écran d'accueil

- L'écran « Historique » correspond à un écran où seront affichés les différentes mesures de shimmer et de jitter sur un enregistrement précis. En effet, grâce à une liste déroulante, on peut sélectionner un enregistrement et voir sur le graphe le point correspondant aux mesures prises pour cet enregistrement. Cet écran est ici utilisé principalement pour l'analyse de résultats.

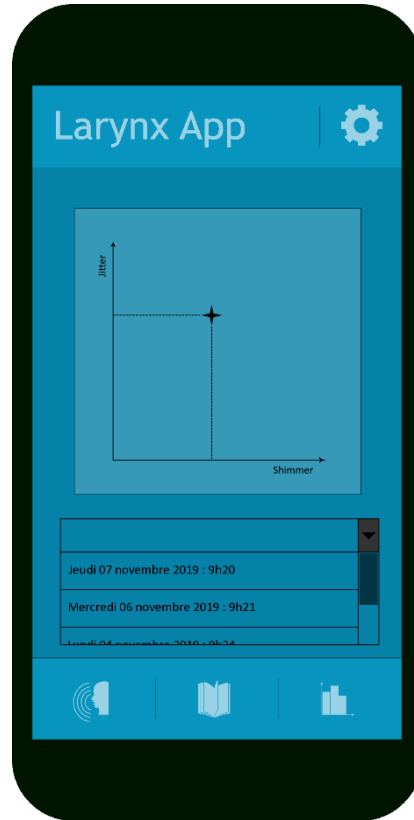


Figure 59 - Ecran d'historique des enregistrements

- L'écran « Suivi » correspond à un écran où seront affichés deux graphes : le jitter en fonction du temps et le shimmer en fonction du temps. Cela permettra de suivre l'évolution de ces mesures et voir s'il y a une dégradation. Sur cet écran, on peut choisir d'afficher le graphe sur une certaine période donnée en entrant une date de début dans le champ Debut et une date de fin dans le champ Fin. Ces deux champs seront en fait des calendriers sélectionnables.

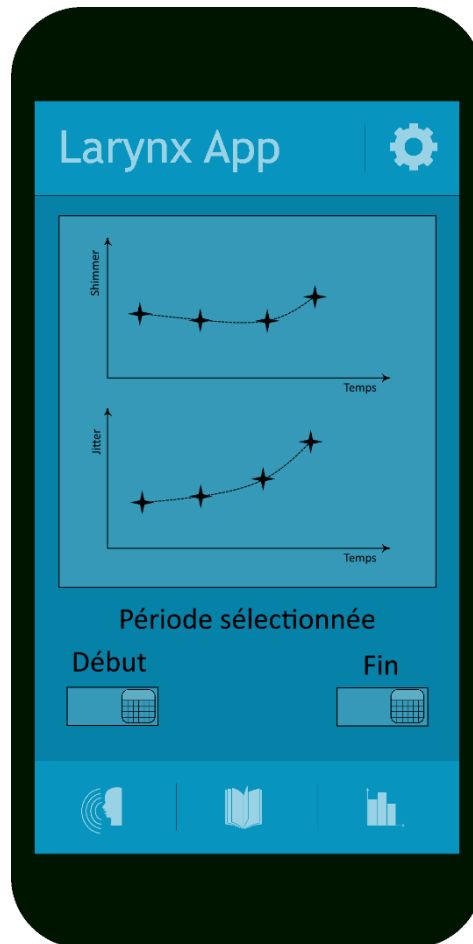


Figure 60 - Ecran de suivi de la voix

Sur la barre d'icônes existent :

1. L'icône "Voix" (Visage avec des ondes partant de sa bouche) permet de revenir à l'écran d'accueil.
2. L'icône "Historique" (Livre) permet d'afficher Historique.
3. L'icône "Suivi" (Graphe) permet d'afficher Suivi.

### 1.3 Interfaces logiciel/logiciel

La base de données est créée et stockée dans la mémoire interne du téléphone. On utilise pour cela une base de données SQLite auquel on accède avec des objets comme SQLiteDatabase ou des classes comme SQLiteOpenHelper.

L'application accédera aussi aux différentes bibliothèques d'Android pour créer l'interface et les actions des déclencheurs associés.

## 2 Spécifications fonctionnelles

Dans cette partie, nous allons décrire chacune des fonctions présentes dans ce diagramme d'activités :

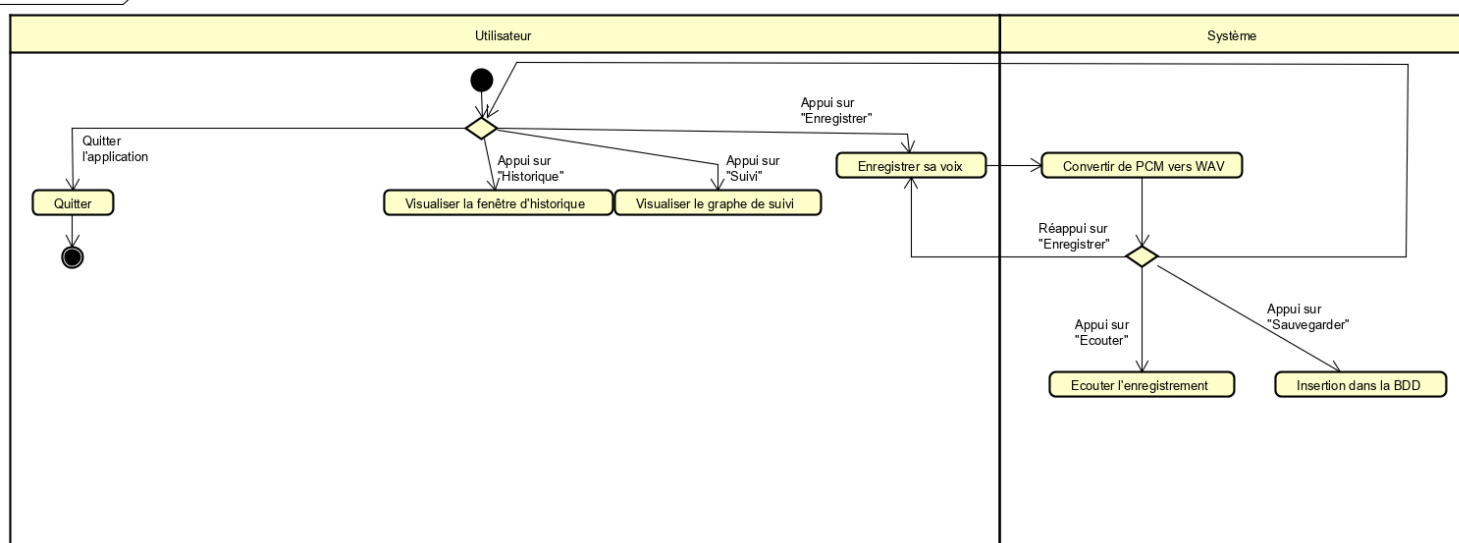


Figure 61 - Diagramme d'activités de l'application

### 2.1 Définition de la fonction « Enregistrer sa voix »

Identification de la fonction « Enregistrer sa voix » :

- Nom : « Enregistrer sa voix »
- Rôle : cette fonction permet de capturer la voix à partir du microphone de l'appareil et va sauvegarder cet enregistrement dans le système de fichiers local à l'appareil
- Priorité : primordiale

Description de la fonction « Enregistrer sa voix » :

- Entrée : la voix du patient
- Sortie : fichier .WAV correspondant à l'enregistrement
- Préconditions : le microphone de l'appareil est activé
- Interactions : avec la visualisation de l'état de la voix (graphe de suivi), avec un convertisseur PCM vers WAV et avec le lecteur audio de l'appareil

## 2.2 Définition de la fonction « Visualiser le graphe de suivi »

Identification de la fonction « Visualiser le graphe de suivi » :

- Nom : « Visualiser le graphe de suivi »
- Rôle : cette fonction permet de visualiser le graphe représentant l'évolution de différents critères pathologiques de la voix en fonction du temps
- Priorité : primordiale

Description de la fonction « Visualiser le graphe de suivi » :

- Entrée : les enregistrements précédemment effectués
- Sortie : un graphique représentant les différents enregistrements à des instants t
- Préconditions : au moins un enregistrement a été effectué, sinon le graphe sera vide
- Interactions : avec la base de données pour consulter quel enregistrement a été fait à telle date et son résultat en jitter et en shimmer et avec la bibliothèque pour créer des graphiques

## 2.3 Définition de la fonction « Visualiser l'historique »

Identification de la fonction « Visualiser l'historique » :

- Nom : « Visualiser l'historique »
- Rôle : permet d'afficher l'historique des enregistrements
- Priorité : primordiale

Description de la fonction « Visualiser l'historique » :

- Entrée : les enregistrements précédemment effectués
- Sortie : une liste des enregistrements et un graphe représentant chaque point (enregistrement)
- Interactions : avec la bibliothèque pour créer des graphiques, avec la base de données

## 3 Spécifications non fonctionnelles

### 3.1 Contraintes de développement et conception

Le développement logiciel est restreint à la programmation en Java Android pour la reprise de l'existant et aussi car le langage natif d'Android est Java. Il est donc nécessaire de disposer d'un émulateur Android ou d'un appareil Android. Le développement est effectué sur l'IDE Android Studio qui est un IDE privilégié par de grosses entreprises telles que Google pour le développement Android.

### 3.2 Contraintes de fonctionnement et d'exploitation

#### 3.2.1 *Performances*

Du point de vue de l'utilisateur, il faudrait qu'en un temps de réponse très court (quelques secondes au maximum) le résultat de l'analyse de son enregistrement s'ajoute au graphe de suivi. L'utilisateur ayant une utilisation fréquente de l'application, cette dernière se soit d'être fonctionnelle la plupart du temps (si le problème vient de l'appareil ou d'Android lui-même, l'application peut ne pas être disponible).

Du point de vue de l'environnement, la fréquence moyenne d'acquisition de mesures dépendra de l'utilisateur, il se peut que ce soit une fois par jour comme une fois par mois. Il faudrait tout de même que l'appareil possède un minimum de mémoire pour stocker ces enregistrements (un enregistrement fait environ 430Ko, ce qui est peu).

Les capacités dépendent des appareils Android utilisés. Ainsi, il est difficile de spécifier une capacité maximale de stockage. Néanmoins, on peut toujours spécifier la taille maximale des données traitées (enregistrements audio). Celle-ci vaut 430Ko par enregistrement, là encore cela dépend des appareils Android (qualité d'enregistrement, qualité du microphone).

Aucun compte utilisateur n'est utilisé ici car l'utilisateur ira sur l'application via son propre téléphone. L'utilisateur a donc accès à toutes les fonctionnalités spécifiées plus haut.

Etant donné que tout est sauvegardé localement, si un reformatage de l'appareil est fait toutes les mesures prises auparavant ainsi que l'application en elle-même auront disparu.

## 4 Gestion de projet

### 4.1 Tâches Kanban : Trello

Pour accéder au tableau Trello, il suffit de cliquer sur ce lien : [Trello - PRD Voix](#). Une capture d'écran est disponible ci-dessous :

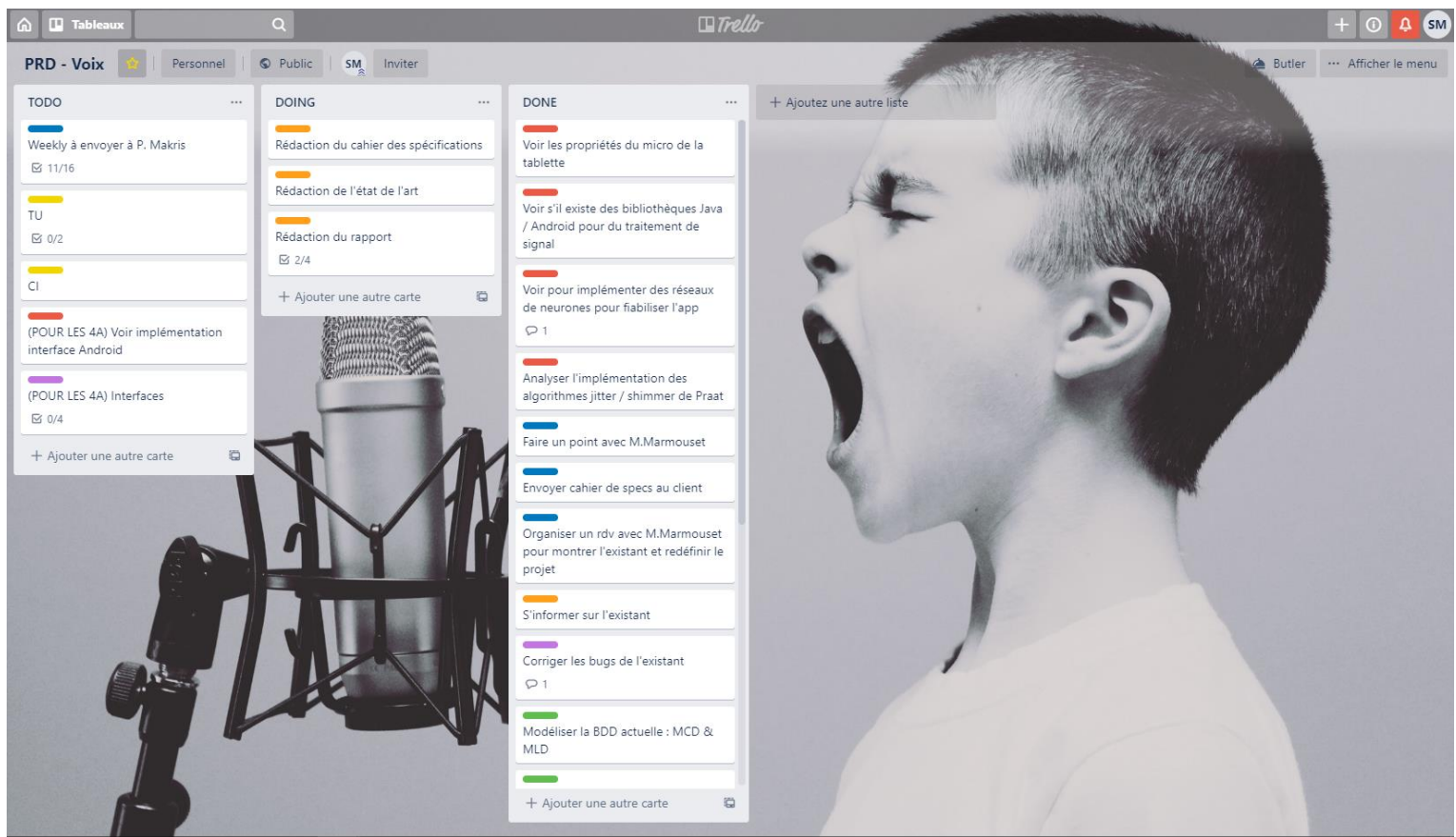


Figure 62 - Trello (Tableau organisationnel des tâches)

Un export PDF du diagramme de Gantt effectué sur GanttProject est disponible ci-dessous. Il regroupe le diagramme des tâches ainsi que le diagramme des ressources (avec les étudiants en 4<sup>e</sup> année comme ressources).

PRD - Voix

Polytech

30 nov. 2019

http://

Chef de projet

Dates du projet

Avancée

Tâches

Ressources

MARTIN Steven

18 sept. 2019 - 27 mars 2020

59%

37

3

Diagramme de Gantt correspondant à la gestion de projet du PRD sur la caractérisation de la voix

Figure 63 - Export Gantt prévisionnel, Page 1 (Intro)

<b>PRD - Voix</b>		30 nov. 2019
<b>Tâches</b>		2
Nom	Date de début	Date de fin
Découverte du projet	18/09/19	17/10/19
S'informer sur l'application existante	18/09/19	03/10/19
S'informer sur les principes liés à la voix	18/09/19	03/10/19
<i>Jitter, Shimmer &amp; FO</i>		
S'informer sur le développement Android	26/09/19	03/10/19
Découverte d'Android Studio	18/09/19	19/09/19
Recherche	09/10/19	17/10/19
Implémentation de réseaux de neurones ?	16/10/19	17/10/19
Existence de bibliothèques permettant le calcul des paramètres ?	09/10/19	10/10/19
Livrables	18/10/19	09/12/19
Rédaction du cahier des spécifications	18/10/19	13/11/19
<i>V1</i>		
Rédaction de l'état de l'art	18/10/19	04/12/19
Rédaction de l'analyse des besoins	18/10/19	04/12/19
Rédaction du rapport	05/12/19	09/12/19
Remise des livrables	09/12/19	09/12/19
Réunions	18/09/19	20/01/20
Réunion d'introduction au sujet	18/09/19	18/09/19
Réunion avec le responsable du projet au CHU	26/09/19	26/09/19
Réunion validation mockup et avancement avec le client	21/11/19	21/11/19
Troisième réunion avec le client	21/01/20	21/01/20
Génie logiciel	17/10/19	07/11/19
Modélisation du projet	17/10/19	07/11/19
Revoir le pattern MVC de l'application	06/11/19	07/11/19
Tests préliminaires	23/10/19	24/10/19
Créer des vraies données de test	23/10/19	24/10/19

Figure 64 - Export Gantt prévisionnel, Page 2 (Tâches 1)

## Tâches

3

Nom	Date de début	Date de fin
Prise en main avec injection de signaux purs <i>Pour détecter s'il y a un problème avec l'algorithme et les différents seuils associés</i>	24/10/19	24/10/19
Conception	06/11/19	07/11/19
Concevoir une base de données efficace pour la prise de données en fonction du temps	06/11/19	07/11/19
Développement	25/10/19	26/03/20
Correction des bugs de l'existant	25/10/19	25/10/19
Ajustement de l'algorithme et des seuils associés	25/10/19	11/12/19
Création d'interface (4A)	25/10/19	21/01/20
Mockups d'interface	25/10/19	13/11/19
Développement des interfaces	21/11/19	21/01/20
Développement des contrôleurs des interfaces	14/01/20	18/03/20
Qualité logiciel	19/03/20	26/03/20
Tests unitaires	19/03/20	26/03/20
Intégration continue (CI)	19/03/20	26/03/20

Figure 65 - Export Gantt prévisionnel, Page 3 (Tâches 2)

## Ressources

4

Nom	Rôle par défaut
MARTIN Steven	Chef de projet
PUPETTO Fabien	Développeur
GAURIAT Yohan	Développeur

Figure 66 - Export Gantt prévisionnel, Page 4 (Ressources)

# PRD - Voix

30 nov. 2019

5

## Diagramme de Gantt

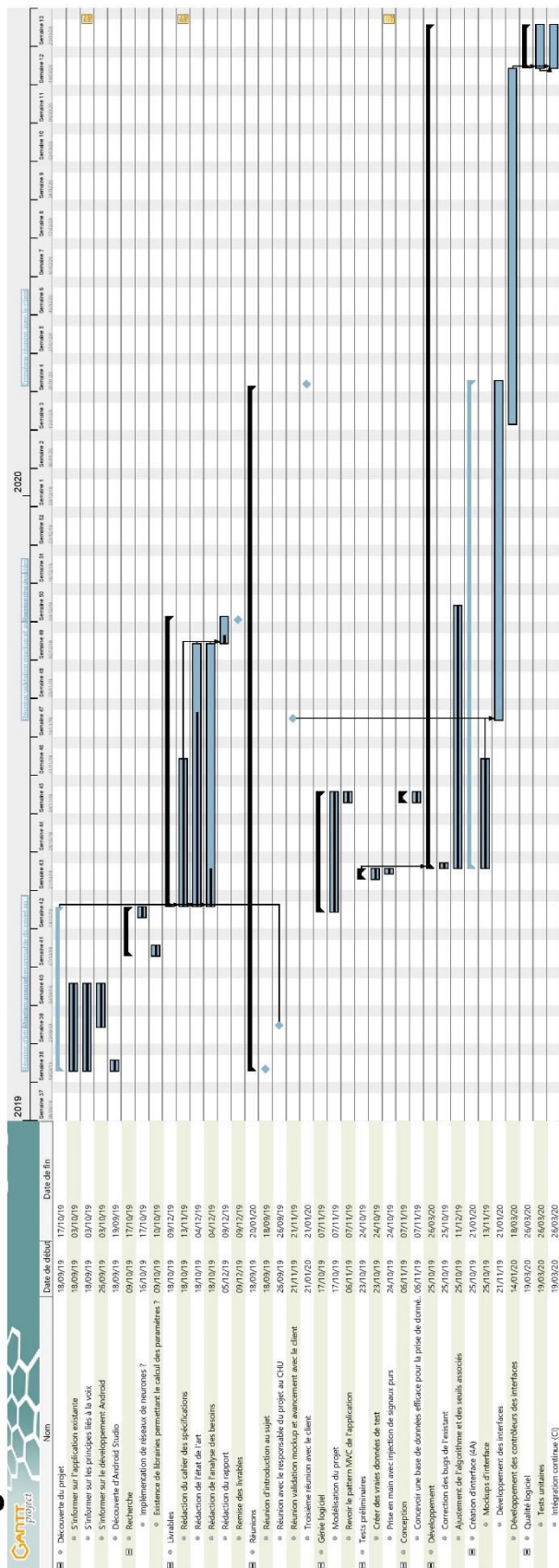


Figure 67 - Export Gantt prévisionnel, Page 5 (Diag. Gantt)

## Diagramme des Ressources



Figure 68 - Export Gantt prévisionnel, Page 6 (Diag. Ressources)

Vous pourrez trouver ici l'export PDF du diagramme de Gantt final.

PRD - Voix

12 avr. 2020

Polytech

<http://>

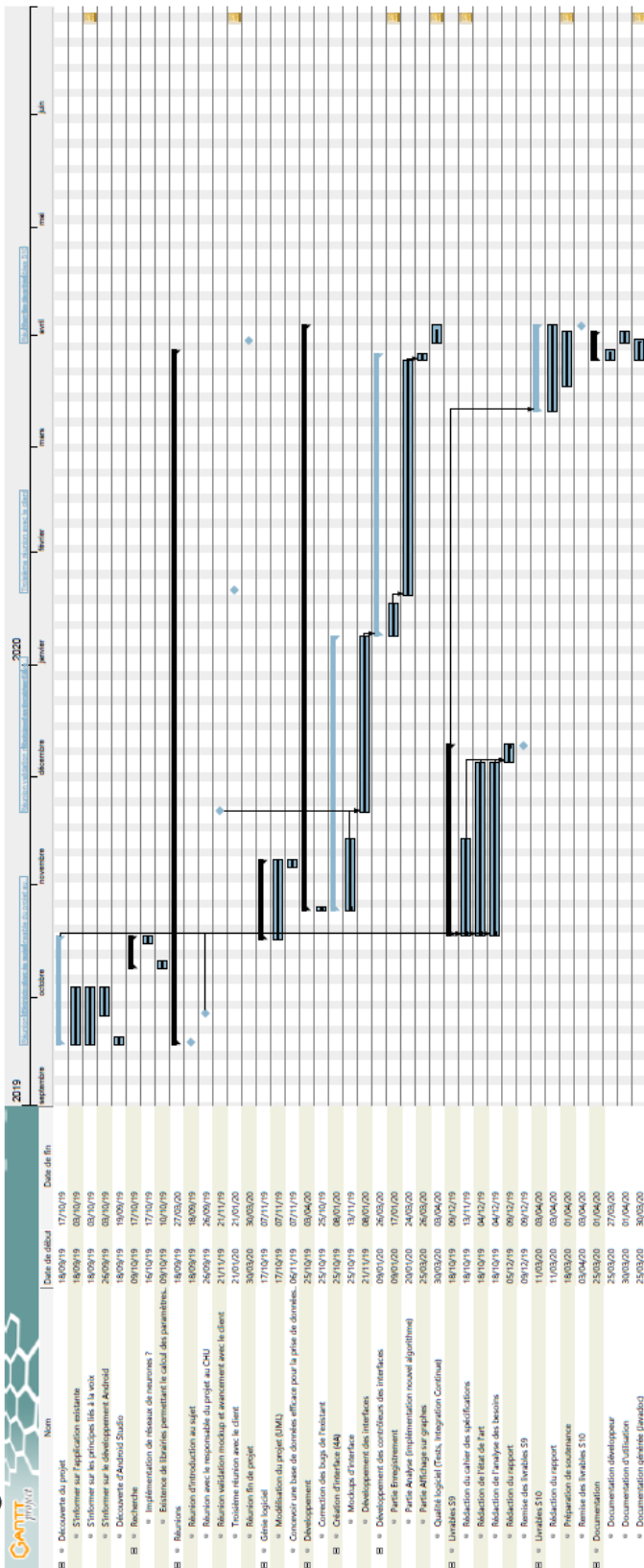
Chef de projet	MARTIN Steven
Dates du projet	18 sept. 2019 - 4 avr. 2020
Avancée	83%
Tâches	41
Ressources	3

Diagramme de Gantt correspondant à la gestion de projet du PRD sur la caractérisation de la voix

Figure 69 - Export Gantt final - Général

# PRD - Voix

## Diagramme de Gantt



12 avr. 2020

5

Figure 70 - Export Gantt final

Dans cette annexe, nous allons mettre les algorithmes utilisés dans l'application existante.

### 5.1 Algorithme de calcul de période existant

```
/**
 * The method calculating the base fragment
 */
public void calculatePositionsV1() {

    //check if data is valid
    if ( data.size() < 5000 )
        return;

    // get the first 5000 points for calculating the periods
    List<Short> dataTmp = new ArrayList<>();
    int maxTmp = 0;
    for ( int i = 0; i < 5000; i++ ) {
        dataTmp.add( data.get( i ) );
        if ( maxTmp < data.get( i ) ) {
            maxTmp = data.get( i );
        }
    }

    // get the pitch position temporary and the difference between two pitch position
    // set the first condition of select the pitch position : value > 0.9 * maxTmp
    List<Integer> listPositions = new ArrayList<>();
    List<Short> listPositionValue = new ArrayList<>();
    List<Integer> diffPositions = new ArrayList<>();
    double thresholdTop = maxTmp * 1.1;
    double thresholdUnder = maxTmp * 0.90;
    for ( int i = 0; i < 5000; i++ ) {
        if ( dataTmp.get( i ) > thresholdUnder ) {
            listPositions.add( i );
            listPositionValue.add( dataTmp.get( i ) );
        }
    }

    //set the second condition of select the pitch position :
    // the difference between two pitch point must in (40,400)
    // because the frequency of the man normal is 100Hz to 1000Hz
    // the frequency of the sampling is 44100Hz
    int diffThresholdUnder = 40;
    int diffThresholdTop = 400;
    for ( int i = 0; i < listPositions.size() - 1; i++ ) {
        int diff = listPositions.get( i + 1 ) - listPositions.get( i );

        if ( diff > diffThresholdUnder && diff < diffThresholdTop ) {
            diffPositions.add( diff );
        }
    }

    if ( diffPositions.isEmpty() ) {
        Log.d( "calculatePositions", "The record seems to be just noise" );
        return;
    }

    int mean = 0;
    for ( int i = 0; i < diffPositions.size(); i++ ) {
        mean += diffPositions.get( i );
    }

    // get the length of the period temporary
    mean = mean / diffPositions.size();

    // the length of base_fragment must be greater than a period and less than two period
    this.baseFragment = (int) ( mean * 1.5 );
    this.offset = this.baseFragment / 2;
}
```

```

/**
 * The method calculating the pitch periods
 */
private void calculatePeriods() {
    int size      = data.size();
    int maxAmp    = 0;
    int startPos  = 0;

    // get the first pitch in the basic period
    for ( int i = 0; i < baseFragment; i++ ) {
        if ( maxAmp < data.get( i ) ) {
            maxAmp = data.get( i );
            // set this position as the start position
            startPos = i;
        }
    }

    // find every pitch in all the fragments
    int pos = startPos + offset; // set current position
    int posAmpMax;
    while ( startPos < size - baseFragment ) {
        if ( data.get( pos ) > 0 ) { // only read the positive data
            posAmpMax = 0;
            maxAmp = 0;
            // access to all the data in this fragment
            while ( pos < startPos + baseFragment ) {
                // find the pitch and mark this position
                if ( maxAmp < data.get( pos ) ) {
                    maxAmp = data.get( pos );
                    posAmpMax = pos;
                }
                pos++;
            }
            // add pitch position into the list
            pitchPositions.add( posAmpMax );
            // update the start position and the current position
            startPos = posAmpMax;
            pos = startPos + offset;
        }
        else {
            pos++;
        }
    }

    // calculate all periods and add them into list
    for ( int i = 0; i < pitchPositions.size() - 1; i++ ) {
        periodsLength.add( pitchPositions.get( i + 1 ) - pitchPositions.get( i ) );
    }
}

```

Figure 71 - Algorithme de calcul de période existant

```

// FEATURE NUMBER 1 : SHIMMER

/**
 * The method calculating the Shimmer
 *
 * @return the Shimmer
 */
public double getShimmer() {
    int minAmp = 0;
    int maxAmp;
    Long A_diff_sum = 0; // sum of difference between every two peak-to-peak amplitudes
    Long A_sum = 0; // sum of all the peak-to-peak amplitudes
    List<Integer> ampPk2Pk = new ArrayList<>(); // this list contains all the peak-to-peak amplitudes

    for ( int i = 0; i < pitchPositions.size() - 1; i++ ) {
        // get each pitch
        maxAmp = data.get( pitchPositions.get( i ) );
        for ( int j = pitchPositions.get( i ); j < pitchPositions.get( i + 1 ); j++ ) {
            if ( minAmp > data.get( j ) ) {
                minAmp = data.get( j );
            }
        }
        // add peak-to-peak amplitude into the list
        ampPk2Pk.add( maxAmp - minAmp );
        // reset the min amplitude
        minAmp = 0;
    }

    // SHIMMER FORMULA (RELATIVE)
    for ( int i = 0; i < ampPk2Pk.size() - 1; i++ ) {
        A_diff_sum += Math.abs( ampPk2Pk.get( i ) - ampPk2Pk.get( i + 1 ) );
        A_sum += ampPk2Pk.get( i );
    }
    // add the last peak-to-peak amplitude into sum
    if ( ampPk2Pk.size() > 0 ) {
        A_sum += ampPk2Pk.get( ampPk2Pk.size() - 1 );
    }
    // calculate shimmer (relative)
    return ( (double) A_diff_sum / (double) ( ampPk2Pk.size() - 1 ) ) / ( (double) A_sum / (double) ampPk2Pk.size() );
}

```

Figure 72 - Algorithme de calcul de shimmer existant

```
// FEATURE NUMBER 2 : JITTER

/**
 * The method calculating the Jitter
 *
 * @return the Jitter
 */
public double getJitter() {
    double sumOfDifferenceOfPeriods = 0.0; // sum of difference between every two periods
    double sumOfPeriods = 0.0; // sum of all periods
    double numberOfPeriods = periodsLength.size(); //set as double for double division

    // JITTER FORMULA (RELATIVE)
    for ( int i = 0; i < periodsLength.size() - 1; i++ ) {
        sumOfDifferenceOfPeriods += Math.abs( periodsLength.get( i ) - periodsLength.get( i + 1 ) );
        sumOfPeriods += periodsLength.get( i );
    }

    // add the last period into sum
    if ( periodsLength.size() > 0 ) {
        sumOfPeriods += periodsLength.get( periodsLength.size() - 1 );
    }

    double meanPeriod = sumOfPeriods / numberOfPeriods;

    // calculate jitter (relative)
    return ( sumOfDifferenceOfPeriods / ( numberOfPeriods - 1 ) ) / meanPeriod;
}
```

Figure 73 - Algorithme de calcul de jitter existant

```
/**
 * The method finding the fundamental frequency of the data.
 *
 * To increase efficiency, this method only test the frequencies between 40Hz to 400Hz.
 */
private void calculateF0() {
    final double newMaxThreshold = 1.03;

    nextPeriodSearchingAreaEnd = data.size() - HzToPeriod( 40 );
    Long autoCorrelationMax = 0;
    int periodRef = 0;
    Long autoCorrelation;

    for ( int i = HzToPeriod( 400 ); i < HzToPeriod( 40 ); i++ ) {
        autoCorrelation = autoCorrelation( i );
        if ( autoCorrelation > autoCorrelationMax * newMaxThreshold ) {
            autoCorrelationMax = autoCorrelation;
            periodRef = i;
        }
    }

    f0 = periodToHz( periodRef );
}
```

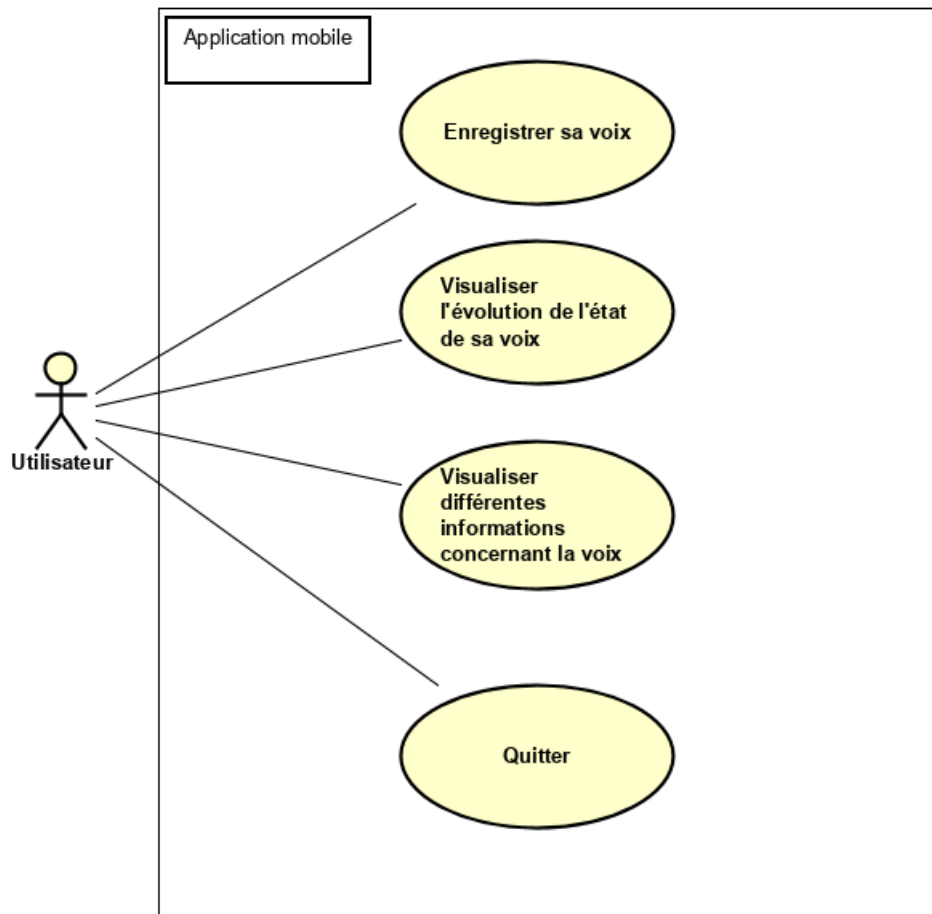
Figure 74 - Algorithme de calcul de F0 existant

## I. Modélisation du projet

Dans cette partie, nous indiquerons comment a été conçu le projet grâce à une modélisation UML du projet en lui-même puis nous parlerons de la base de données implémentée.

### 1. Modélisation UML du projet

Le projet a été conçu pour répondre au diagramme de cas d'utilisation suivant :



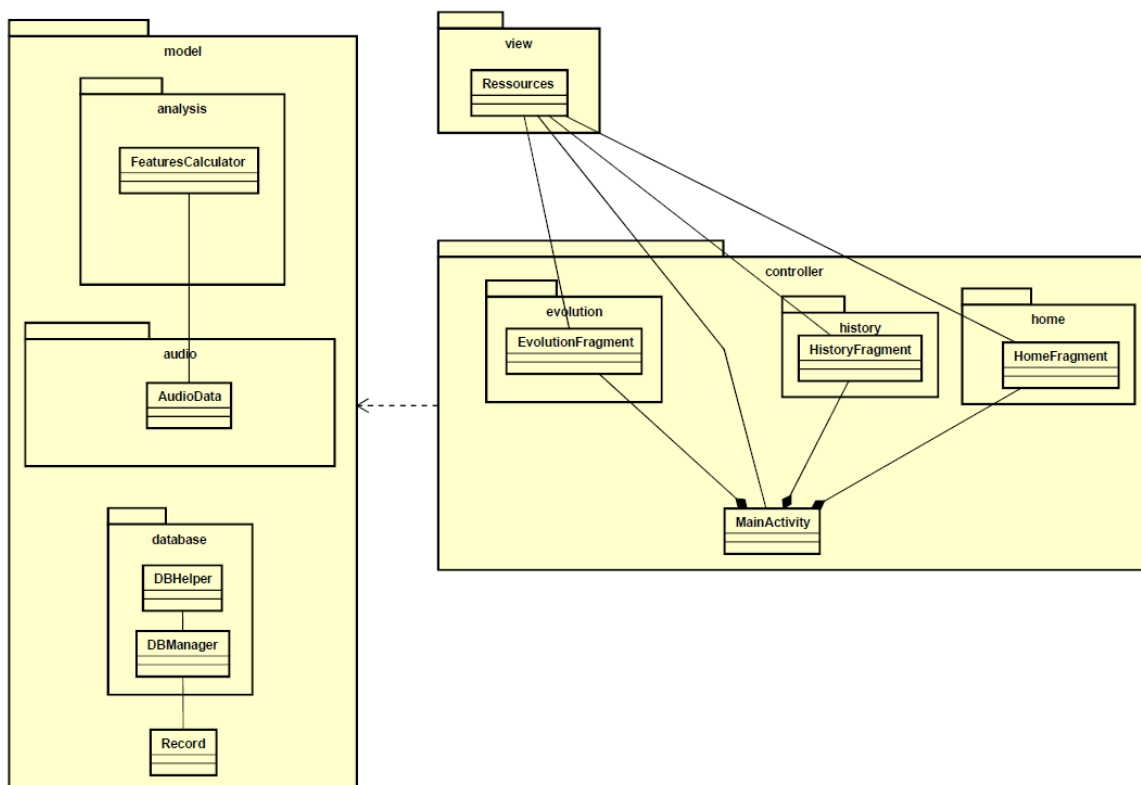
En effet, ce projet a pour but d'enregistrer sa voix pendant 5 secondes puis d'analyser cette dernière pour en ressortir les différentes caractéristiques qui pourront être affichées dans les différents onglets de l'application.



Pour ce faire, nous avons décidé de partir sur une modélisation MVC où le modèle serait les composantes de la base de données et tous les modèles de calcul utilisés pour trouver les différentes caractéristiques de la voix, où le contrôleur serait composé des différents fragments (Android Fragment : un onglet de l'application) et de l'activité principale qui appellerait ces fragments et où la vue serait l'interface.

**NB** : Les activités Android correspondent au moyen de gérer des « fenêtres » tandis que les fragments permettent de gérer des « onglets » appartenant à une fenêtre.

Ainsi, nous nous retrouvons avec le diagramme de classe suivant :



## 2. Base de données

Maintenant, nous allons parler de la base de données implémentée. Cette base de données est une base SQLite qui est le plus commun pour Android pour des questions de légèreté et de rapidité. Cette base, ici très simple, est composée d'une seule et unique table : la table Record (ou Enregistrement). En effet, pour ce projet nous n'avons pas besoin de plus que ça car dans cette table nous allons stocker le nom du fichier créé après l'enregistrement, le chemin vers ce fichier et les valeurs des différentes caractéristiques de la voix (Shimmer, Jitter et fréquence fondamentale). Chaque fichier sera créé de telle sorte à indiquer la date dans son nom, pour trier par date il faudra donc simplement parser les différents noms enregistrés dans la base de données.

La table Record est donc comme ceci :

Record	
PK	<u>name</u>
	path
	jitter
	shimmer
	f0

## II. Environnement de développement et langage

Pour développer l'application, nous avons utilisé Android Studio version 3.5.3 avec pour langage de programmation le Java.

## III. Versions et librairies utilisées

Dans cette partie, nous aborderons les différentes versions utilisées et supportées puis nous parlerons des différentes librairies utilisées.

### 1. Versions utilisées et supportées

L'application a été conçue pour fonctionner sur quasiment tous les appareils étant donné qu'elle est compatible avec toutes les versions antérieures à la version 15 d'Android (actuellement nous sommes à la 29). Cette version minimale est la version proposée directement par Android Studio.

Concernant maintenant les versions de différents packages, nous avons la liste suivante :

- Androidx.navigation : v2.1.0
- JUnit : v4.13
- Robolectric : v4.2.1
- Androidx.test.core : v1.2.0
- Espresso (tests fonctionnels) : v3.2.0
- Material design : v1.0.0
- ConstraintLayout : v1.1.3
- Androidx.appcompat : v1.1.0
- Android Studio : v3.5.3

A partir de cette liste, nous pouvons voir que les nouvelles librairies AndroidX sont utilisées pour une meilleure fiabilité de l'application.

## 2. Librairies utilisées

Pour ce projet, deux librairies externes ont été utilisées : TarsosDSP et MPAndroidChart.

Dans un premier temps, nous allons présenter TarsosDSP et son intégration dans le projet puis dans un second temps nous parlerons de MPAndroidChart.

### 2.1. TarsosDSP

TarsosDSP est une librairie open-source de traitement de signal (<https://github.com/JorenSix/TarsosDSP>). Elle permet notamment de rendre plus simple des tâches comme l'enregistrement ou la détection de « pitch » (fréquence de la voix) à un moment t. Ici, c'est pour ces deux tâches qu'elle sera utilisée.

La librairie est trop complète pour être expliquée en détails ici mais nous allons tout de même nous attarder sur les éléments utilisés.

TarsosDSP permet d'avoir un AudioDispatcher qui va regrouper plusieurs processus et qui va les lancer en même temps pour nous. Cet AudioDispatcher peut être créé à partir d'un AudioDispatcherFactory comprenant deux méthodes : fromDefaultMicrophone() et fromPipe(). La méthode fromDefaultMicrophone() permet de créer un dispatcher qui écoutera le microphone du téléphone et qui effectuera les traitements en temps réel. La méthode fromPipe(), quant à elle, permet de créer un dispatcher qui effectuera des traitements à partir d'un fichier pré-enregistré. A noter que cette dernière méthode nécessite l'installation d'un encodeur FFMpeg.

Dans notre cas, nous allons utiliser un dispatcher temps réel qui utilisera donc le microphone de l'appareil comme ceci :

```
dispatcher = AudioDispatcherFactory.fromDefaultMicrophone( 44100, 2048, 0);
```

Les paramètres correspondent respectivement à la fréquence d'échantillonnage, la taille du buffer et la partie « skipée » du buffer. Ici, nous avons donc une fréquence d'échantillonnage classique de 44.1kHz et une taille de buffer de 2048. Ce buffer va servir à garder les différents échantillons créés lors de l'enregistrement.

Maintenant que nous avons vu comment le dispatcher se comporte, allons voir du côté des processus que ce dispatcher permet de gérer. Ces processus se nomment des AudioProcessor.

Plusieurs AudioProcessor sont disponibles dans la librairie mais les deux principaux qui nous intéressent sont le WriterProcessor et le PitchProcessor.

Le WriterProcessor permet d'écrire un fichier contenant l'enregistrement audio. Il prend en paramètre le fichier de sortie et le format dans lequel nous souhaitons écrire.

```
AudioProcessor recordProcessor = new WriterProcessor(AUDIO_FORMAT, randomAccessFile);
dispatcher.addAudioProcessor(recordProcessor);
```

Le format est un peu spécifique étant donné que c'est un TarsosDSPAudioFormat qui est en fait un AudioFormat classique d'Android mais qui permet d'indiquer respectivement la fréquence d'échantillonnage, le nombre de bits pour chaque échantillon, le nombre de canaux (1 pour mono, 2 pour stéréo), si les données sont signées ou non et dans quel ordre les bytes doivent être enregistrés.

```
AUDIO_FORMAT = new TarsosDSPAudioFormat(
    v: 44100,
    b: 16,
    ch: 1,
    s: true,
    ByteOrder.LITTLE_ENDIAN.equals(ByteOrder.nativeOrder()));
```

Le PitchProcessor permet de détecter la fréquence de la voix à chaque instant et est configurable de telle sorte à ce qu'il puisse utiliser différents algorithmes dont l'algorithme Yin qui est utilisé ici.

```
PitchDetectionHandler pitchDetectionHandler = (res, e) -> {
    float pitchInHz = res.getPitch();
    if (pitchInHz != -1 && pitchInHz < 400)
        pitches.add(pitchInHz);
};

AudioProcessor pitchProcessor = new PitchProcessor(new Yin(audioSampleRate: 44100, bufferSize: 2048),
    (sampleRate: 44100, bufferSize: 2048, pitchDetectionHandler));
dispatcher.addAudioProcessor(pitchProcessor);
```

Comme on le voit ci-dessus, le PitchProcessor prend en compte un handler qui est la partie qui va détecter la fréquence à chaque instant. Ici, notre handler est configuré pour stocker les différentes fréquences dans une liste utilisable pour l'analyse. Comme dit précédemment, ce processor prend en paramètre un algorithme, la fréquence d'échantillonnage, la taille du buffer et un handler.

Pour plus d'explications sur pourquoi l'algorithme Yin a été choisi, nous vous invitons à vous référer au rapport de PRD.

Pour lancer le dispatcher, il suffit ensuite de faire un run() comme pour toute classe implémentant Runnable.

**NB** : à chaque fois que l'enregistrement est terminé, le dispatcher est stop() et détruit pour éviter des conflits de thread.

Côté version, TarsosDSP est implémenté avec la dernière version en date.

## 2.2. MPAndroidChart

MPAndroidChart est une librairie de création de graphes pour Android (<https://github.com/PhilJay/MPAndroidChart>).

La version utilisée dans le projet est la version 3.1.0.

## IV. Permissions

Des permissions à fournir sont obligatoires pour que l'application tourne. Ces permissions sont les suivantes :

- L'accès au microphone de l'appareil
- L'accès au stockage de l'appareil

## V. Tests

Des tests utilisant JUnit ont été effectués sur chaque classe du modèle.

**NB** : FeaturesCalculatorTest crée un signal sinusoïdal pur pour ensuite vérifier les valeurs des caractéristiques calculées. En effet, le Jitter et le Shimmer doivent, pour un signal sinusoïdal pur, valoir 0 étant donné qu'il n'y a pas de variation en fréquence ou en amplitude au cours du temps. Ce test n'est pour le moment pas fonctionnel.

Le test fonctionnel de l'activité principale (MainActivity) a été réalisé avec Espresso qui est un plugin préinstallé sur Android Studio pour faire des tests fonctionnels facilement.

## VI. Intégration continue et gestion de version

Ce projet utilise un gestionnaire de version décentralisé qui est GitHub, le lien vers le projet est le suivant : <https://github.com/StevenMartin00/PRD-Voix>.

Un système d'intégration continue a été implémenté via GitHub Actions. GitHub Actions utilise un fichier yml pour gérer ses workflows.

A partir de GitHub Actions, il est possible de créer un fichier yml de base pour Android. Ce fichier ne comporte pas de jobs prédéfinis.

Ainsi, il a fallu définir plusieurs jobs dans ce yml qui est le suivant :

---

```
name: Android CI

on: [push]

jobs:
  build:

    runs-on: macOS-latest

    steps:
      - name: checkout
        uses: actions/checkout@v2
      - name: set up JDK 1.8
        uses: actions/setup-java@v1
        with:
          java-version: 1.8
      - name: Build with Gradle
        run: chmod +x gradlew && ./gradlew build
      - name: Run tests
        run: ./gradlew test -PdisablePreDex --stacktrace
      - name: Run tests on Android emulator
        uses: reactivecircus/android-emulator-runner@v2
        with:
          api-level: 29
          script: ./gradlew connectedCheck
      #- name: Run code coverage (reports available in app/build/reports/coverage)
      # run: ./gradlew createDebugAndroidTestCoverageReport
```

---

---

```

name: Android CI

on: [push]

jobs:
  build:

    runs-on: macOS-latest

    steps:
      - name: checkout
        uses: actions/checkout@v2
      - name: set up JDK 1.8
        uses: actions/setup-java@v1
        with:
          java-version: 1.8
      - name: Build with Gradle
        run: chmod +x gradlew && ./gradlew build
      - name: Run tests
        run: ./gradlew test -PdisablePreDex --stacktrace
      - name: Run tests on Android emulator
        uses: reactivecircus/android-emulator-runner@v2
        with:
          api-level: 29
          script: ./gradlew connectedCheck
      #- name: Run code coverage (reports available in app/build/reports/coverage)
      # run: ./gradlew createDebugAndroidTestCoverageReport

```

---

Ce fichier permet de configurer le fait qu'une suite de jobs sera lancée à chaque nouveau push sur le dépôt.

Ici, pour l'environnement nous allons utiliser macOS pour une raison particulière : cet environnement a les packages nécessaires pour pouvoir faire tourner un émulateur Android.

Maintenant, détaillons les étapes :

- Set up JDK 1.8 : préparation du JDK 1.8 en vue du prochain build
- Build with Gradle : permet de build le projet avec Gradle
- Runs tests : permet de lancer les tests unitaires à partir de Gradle
- Runs tests on Android emulator : permet de lancer les tests fonctionnels sur l'interface d'un émulateur Android. L'émulateur Android utilisé est le reactivecircus qui n'est compatible qu'avec macOS. Cet émulateur utilise la même version qu'est utilisée pour le développement (version de l'api : 29).
- Run code coverage, en commentaire ici, est un job permettant de créer un document décrivant la couverture de code de l'application. Cependant, ce job ne fonctionne pas car je n'arrive pas à initialiser JaCoCo (bibliothèque de couverture de code en Java) pour les tests unitaires sur Android (dossier test et non pas androidTest qui sont les tests fonctionnels).

## VII. Installation

L'APK est distribué avec tous les autres livrables. Pour l'installer, il suffit d'aller sur votre téléphone dans Paramètres > Sécurité et d'activer les sources inconnues. Ensuite, il suffit de cliquer sur l'apk et l'application s'installe et se lance.

### 7 Documentation utilisateur

#### I. Installation

L'APK est distribué avec tous les autres livrables. Pour l'installer, il suffit d'aller sur votre téléphone dans Paramètres > Sécurité et d'activer les sources inconnues. Ensuite, il suffit de cliquer sur l'apk et l'application s'installe et se lance.

#### II. Lancement de l'application

Au lancement de l'application, deux permissions sont demandées : l'accès au micro et au stockage. Il faut à tout prix les accepter. Si elles ne sont pas acceptées, l'application ne peut pas fonctionner.

Toutefois, si l'une des permissions est refusée, il est possible d'aller sur votre téléphone dans la liste des applications puis de sélectionner l'application « Larynx App ». Une fois sélectionné, il suffit d'aller dans « Autorisations » et accepter les différentes autorisations demandées.

#### III. Fenêtre principale

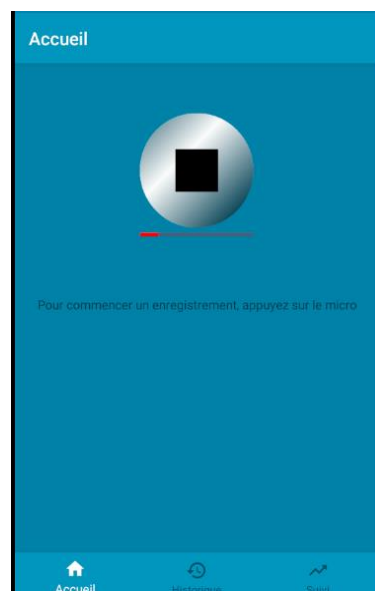
Dans cette partie, nous parlerons des fonctionnalités disponibles sur la fenêtre principale.

Tout d'abord, voici un aperçu de ce à quoi ressemble la fenêtre principale :



Trois onglets sont cliquables en bas pour accéder aux différentes fenêtres décrites dans ce document.

Comme indiqué via le texte présent, il suffit d'appuyer sur le bouton micro pour lancer un enregistrement. Une fois l'enregistrement lancé, le bouton devient comme ceci et une barre de progression de 5 secondes apparaît et défile.

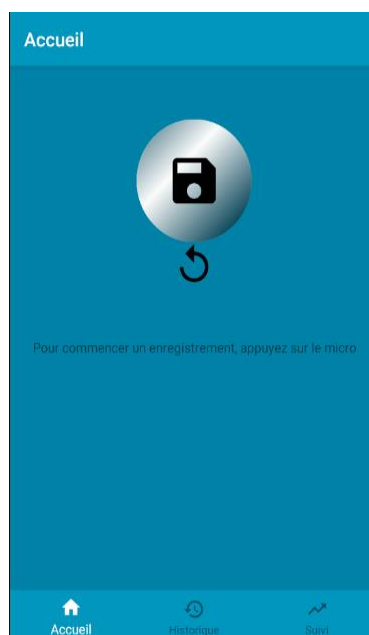


Lors d'un appui sur le bouton Micro devenu Stop, l'enregistrement peut être arrêté et annulé.

Lorsque le bouton Stop est appuyé, il est possible de réessayer via le même bouton devenu un bouton Relancer. Ce bouton permet de revenir à l'état par défaut et de pouvoir enregistrer de nouveau.



Lorsque l'enregistrement est relancé et terminé, cet écran devient comme ceci :



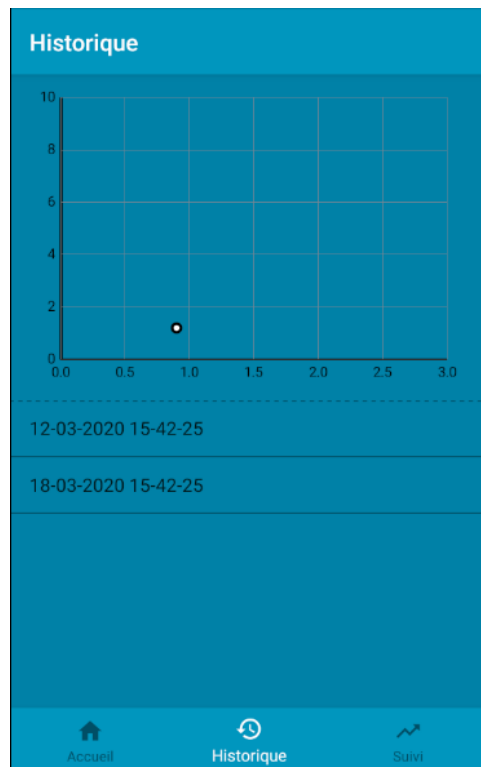
Lors d'un appui sur le bouton Micro devenu Enregistrer, une notification cliquable arrive sur le téléphone indiquant que l'enregistrement a été sauvegardé sur le téléphone. Lorsque la notification est cliquée, l'enregistrement est lu grâce au lecteur par défaut du téléphone.

Voilà donc tout pour cette page. Passons à la fenêtre suivante.

## IV. Fenêtre d'historique

Dans cette partie, nous parlerons des fonctionnalités disponibles sur la fenêtre d'historique.

Maintenant, voici un aperçu de ce à quoi ressemble la fenêtre d'historique :



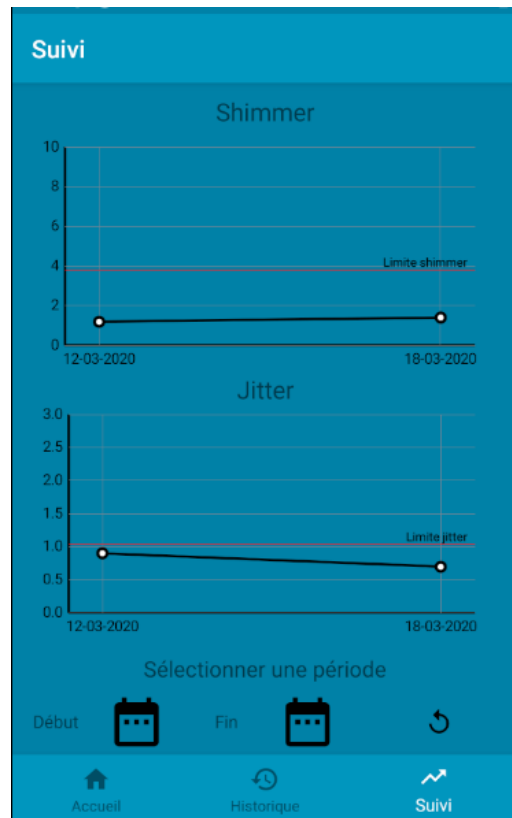
Un graphique représentant les deux caractéristiques calculées en abscisse et en ordonnée est affiché. Ce graphique est présent pour que le médecin ait un rapide coup d'œil sur les valeurs importantes de la voix.

Une liste répertoriant les noms des différents fichiers enregistrés est affichée. Dans cette liste, il est possible de cliquer sur chaque nom pour changer le point correspondant sur le graphique.

## V. Fenêtre de suivi

Dans cette partie, nous parlerons des fonctionnalités disponibles sur la fenêtre de suivi.

Enfin, pour cette dernière fenêtre, voici un aperçu de ce à quoi ressemble la fenêtre de suivi :



Ici deux graphes sont affichés, l'un représentant l'évolution du shimmer au cours du temps et l'autre représentant l'évolution du jitter au cours du temps. Shimmer et Jitter sont les deux caractéristiques représentatives de la voix. Deux limites sont décrites dans ces graphes, il ne vaut mieux pas les dépasser.

Enfin, une option encore indisponible est présente sur cette fenêtre : le filtrage par période de temps.

N°	ACTION	ATTENDU	RESULTAT
1	Ajouter des données à AudioData	Données ajoutées	OK
2	Process les données dans AudioData	Données coupées de manière à enlever les premiers 20% et les derniers 30%	OK
3	Enregistrer sa voix	Fichier crée contenant l'enregistrement pouvant être lu et analysé	OK
4	Analyser la voix	Récupère les fréquences et calcule Jitter/Shimmer	OK
5	Afficher les enregistrements sur un graphe	Un point = un enregistrement trié par date	KO
6	Filtrer les enregistrements par date	Seuls les points dont la date est comprise entre début/fin du filtre sont affichés	KO
7	Réinitialiser l'enregistrement	Fichier non créé, redémarrage du processus d'enregistrement	OK
8	Stopper l'enregistrement en cours	Fichier non créé	OK
9	Ajouter un enregistrement à la BDD	Enregistrement créé dans la BDD	OK
10	Enlever un enregistrement de la BDD	Enregistrement supprimé de la BDD	OK
11	Récupérer les enregistrements de la BDD	Liste d'enregistrements présents dans la BDD retournée	OK
12	Trier les enregistrements par date	Liste d'enregistrements triés par date présents dans la BDD retournée	OK
13	Changer d'onglet	Onglet change	OK
14	Changer d'onglet en plein enregistrement	Enregistrement se coupe, fichier non créé et onglet qui change	OK
15	Tester avec un signal pur	Jitter = 0, Shimmer = 0, F0 = fréquence du signal pur	KO