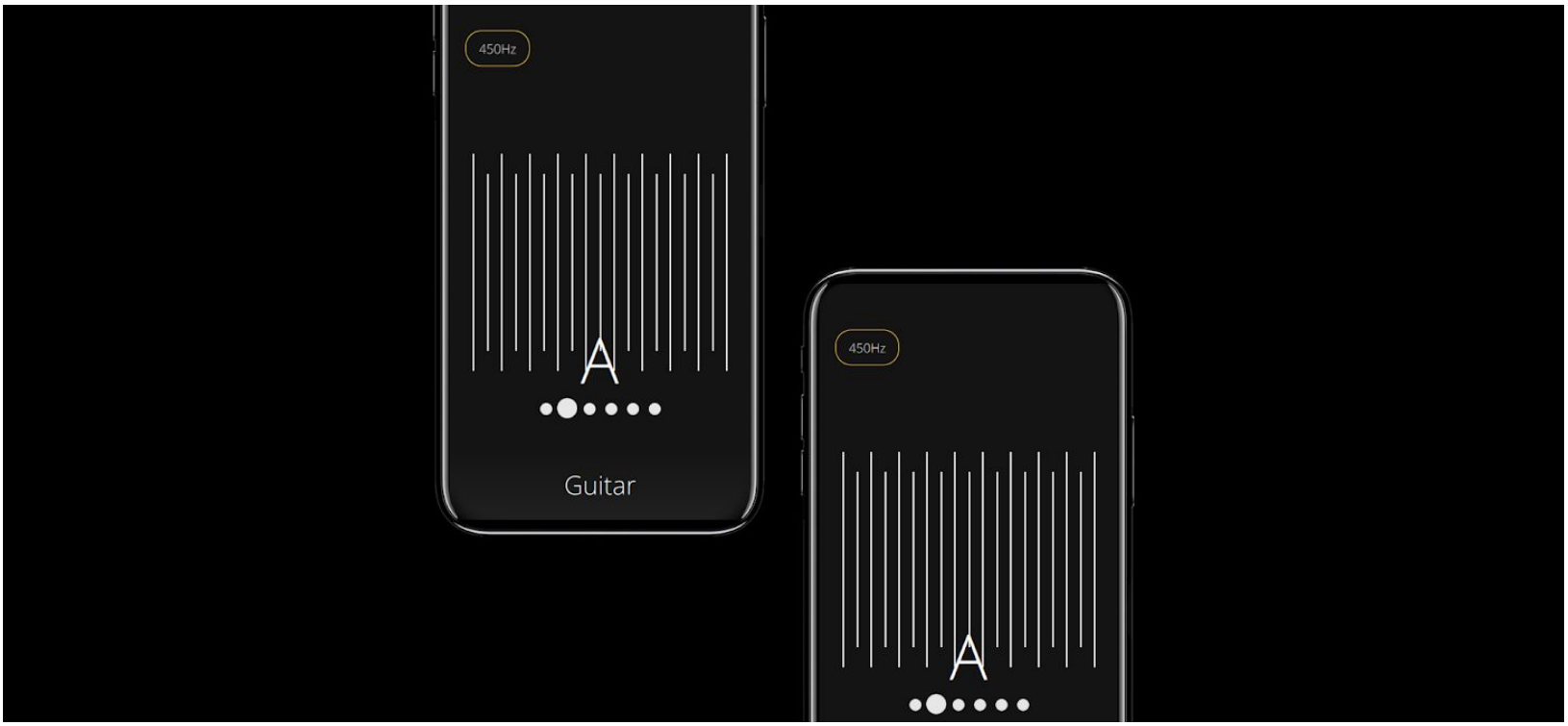
Tune Spot

Minimal tuner for all instruments

Developers:

Front-end: Γεώργιος Μελισσουργός (2771)

Back-end: Σπυρίδων Τσαλίκης (2812), Θεόδωρος Μποζίνης (2493)



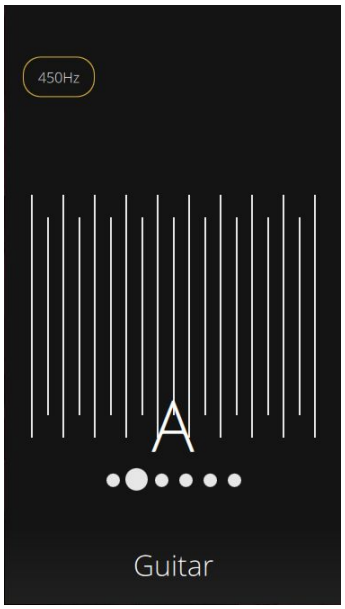
iphone representation

Στόχοι εφαρμογής

Η μουσική είναι ένας τομέας απασχόλησης με πολύ μεγάλο κοινό. Ένα από τα πιο βασικά στοιχεία στην (ανα)παραγωγή ενός κομματιού είναι το σωστό κούρδισμα των οργάνων που χρησιμοποιούνται. Προκειμένου, λοιπόν, ένα όργανο να αποδώσει στο μέγιστο, πρέπει να έχει κουρδιστεί με πολύ μεγάλη ακρίβεια και ακριβώς στα μέτρα τόσο του κομματιού αλλά και του ίδιου του μουσικού. Εμείς οι ίδιοι σαν μουσικοί έχουμε δοκιμάσει ένα μεγάλο μέρος των εφαρμογών που επιλύουν αυτόν τον σκοπό, καμία όμως δεν έχει αρκετή ακρίβεια, αλλά σημαντικότερα δεν περιέχουν τρόπους προσαρμογής του τρόπου κουρδίσματος. Θέλοντας να μιμηθούμε την λειτουργικότητα και την ακρίβεια ενός

παραδοσιακού κουρδιτηριού δημιουργήσαμε αυτή την εφαρμογή.

> It's not always that we need to do more but rather that we need to focus on less.
>
> -Nathan W. Morris



Μινιμαλιστική σχεδίαση - περισσότερες δυνατότητες

----χ

Θέλοντας να φτιάξουμε ένα κουρδιστήρι που να μπορεί να χρησιμοποιηθεί από όλους τους μουσικούς,

ανεξαρτήτως το επίπεδο γνώσης της τεχνολογίας που έχουν, αλλά και ανεξαρτήτως του οργάνου που χρησιμοποιούν προσεγγίσαμε όσο πιο μινιμαλιστικά την γραφική διεπαφή, βάζοντας όμως όσο το δυνατόν μεγαλύτερη λειτουργικότητα. Αρχίσαμε με την λογική της υποστήριξης όλων των οργάνων. Οι περισσότερες εφαρμογές αντιμετωπίζουν το κούρδισμα μόνο σε έγχορδα όργανα, πράγμα που αφήνει όλα τα υπόλοιπα όργανα χωρίς την βοήθεια της τεχνολογίας. Έτσι, δομείσαμε μία εφαρμογή με την ιδέα της συμπερίληψης μοντέλων όλων των οργάνων -ακόμη και της φωνής. Άλλο ένα χαρακτηριστικό που δεν θα μπορούσε να λείπει από την εφαρμογή είναι η επιλογή της βασικής Λα. Ορίζουμε μία συχνότητα η οποία αντιστοιχεί στην νότα Λα (συνήθως μεταξύ 400Hz με 480Hz), προσθέτοντας και αφαιρώντας σε αυτή τη βασική συχνότητα συγκεκριμένα ποσά μπορούμε να πάρουμε όλες τις πιθανές νότες. Η συχνότητα της βασικής Λα διαφέρει, αναλόγως με τον τύπο του σχήματος αλλά ακόμη και του ίδιου του κομματιού. Για παράδειγμα μια κλασική κιθάρα θα κουρδιστεί με βασική Λα 440Hz, ενώ τα όργανα μιας συμφωνικής ορχήστρας θα χρησιμοποιήσουν τα 442Hz. Έτσι η αλλαγή της βασικής Λα αποτελεί κύριο κομμάτι στην γραφική διεπαφή.

Η εφαρμογή διατίθεται δωρεάν ως λογισμικό ανοικτού κώδικα υπό την BSD-3-Clause, περισσότερες πληροφορίες στο αποθετήριο της εφαρμογής:

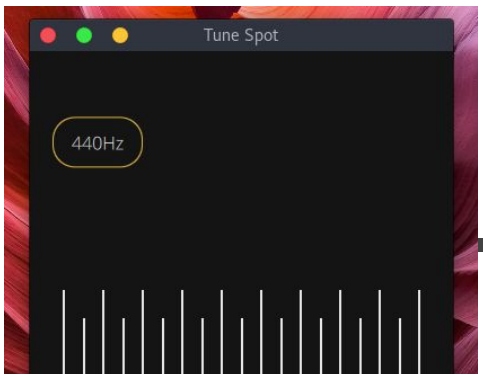
https://github.com/DrMerfy/tune_spot

Σχεδίαση/Ανάπτυξη της γραφικής διεπαφής - frontend documentation -

Γεώργιος Μελισσουργός

Abstract

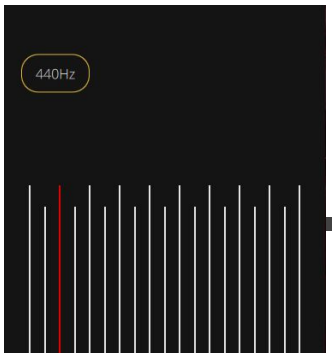
Η ανάπτυξη της γραφικής διεπαφής έγινε κατά κόρων σε qml και Javascript, φυσικά ήταν αναπόφευκτο να αγνοηθεί εντελώς η C++ καθώς είναι αναγκαία για κάποια κομμάτια της λειτουργικότητας (κυρίως για την διασύνδεση με το backend). Δημιουργώντας ένα όσο το δυνατόν πιο μινιμαλιστικό περιβάλλον ο χρήστης με το που ανοίγει την εφαρμογή βλέπει την αρχική σελίδα (εικ.1) όπου

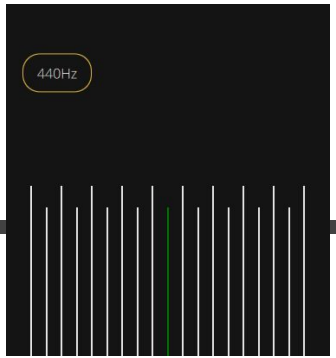


αποτελεί και το κύριο στοιχείο της εφαρμογής. Από εδώ έχει πρόσβαση σε όλες τις δυνατότητες της εφαρμογής.

Ανάλυση στοιχείων

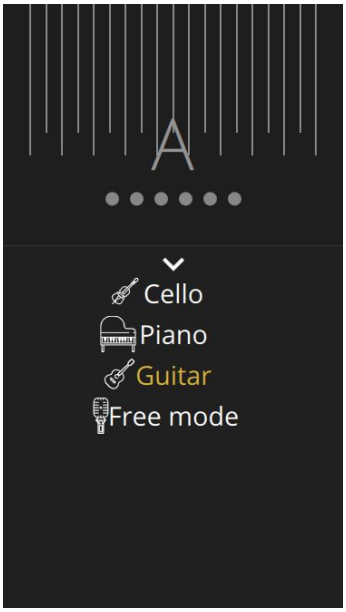
Κεντρικά (οι κατακόρυφες γραμμές) βρίσκεται η **οπτική αναπαράσταση του κουρδιστηριού**, η οποία είναι αναδραστική. Όταν η συχνότητα που εντοπίζεται διαφέρει από την συχνότητα της νότας στόχου (out of tune) μια γραμμή της διεπαφής γίνεται κόκκινη, αναλόγως με πόσο μεγάλη είναι η διαφορά, αλλά και προς τα ποια φορά. Για παράδειγμα, αν ο χρήστης θέλει να κουρδίσει στην Μι (σε μία κιθάρα η πρώτη χορδή) που αντιστοιχεί σε 82.41Hz αλλά η συχνότητα που παράγει η χορδή του είναι 80Hz (η χορδή είναι πιο λάσκα) η τρίτη γραμμή από αριστερά θα γίνει κόκκινη (εικ.2), υποδηλώνοντας στον χρήστη να “πάρει” την χορδή ώσπου η συχνότητα που θα βγάζει να είναι περίπου ίδια με την Μι. Καθόλη τη διάρκεια της διαδικασίας αυτής η διεπαφή αναδράει αλλάζοντας την κόκκινη γραμμή πρός το κέντρο, μέχρι η συχνότητα να γίνει περίπου ίδια με αυτή του στόχου, όπου η κεντρική γραμμή θα γίνει πράσινη (εικ.3). Εδώ αξίζει να σημειωθεί πως η διαχείριση των συχνοτήτων (καταγραφή, υπολογισμός κτλ.) γίνεται από το backend της εφαρμογής. Υπομέρος της οπτικής αναπαράστασης του κουρδιστηριού αποτελεί η **απεικόνιση της νότας στόχου**, η οποία δείχνει κάθε φορά την νότα που κουρδίζει ο χρήστης.





Σημαντικό μέρος του UI αποτελεί η επιλογή του οργάνου, αυτό

γίνεται μέσα από το **drawer** στο κάτω μέρος της οθόνης. Ο χρήστης μπορεί να το ανοίξει είτε πατώντας επάνω του, είτε κάνοντας edge swipe. Το εσωτερικό του drawer αποτελείται από μία λίστα η οποία φορτώνει δυναμικά τα ονόματα των οργάνων καθώς και κάποια εικονίδια.



Επιλέγοντας ένα όργανο, αμά το συγκεκριμένο έχει κάποιο τυποποιημένο κούρδισμα (π.χ. οι χορδές του τσέλου) εμφανίζεται ο **επιλογέας νοτών**, οι κύκλοι κάτω από το όνομα της νότας, ο χρήστης επιλέγοντας τον κατάλληλο κύκλο διαλέγει και την νότα που θέλει να κουρδίσει. Για παράδειγμα αν διαλέξει τον δεύτερο κύκλο, έχοντας επιλεγμένο το τσέλο, θα κουρδίσει την δεύτερη χορδή του τσέλου.

Τέλος, πάνω αριστερά βρίσκεται ο **επιλογέας βασικής συχνότητας**, όπου ο χρήστης σέρνοντας είτε προς τα επάνω, είτε προς τα κάτω διαλέγει την επιθυμητή βασική συχνότητα.

Γενικός σχεδιασμός

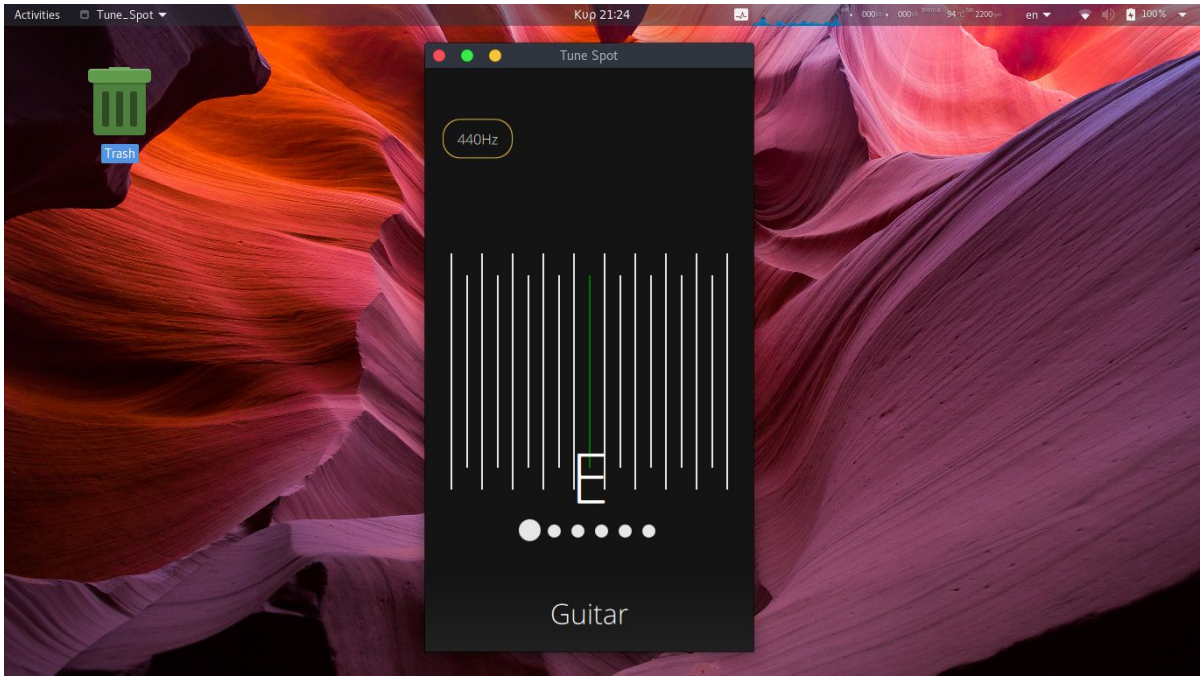
Ακολουθώντας τα πρότυπα καλής σχεδίασης εφαρμογών, η δόμηση του front-end γίνεται ως εξής:

- Το πρόγραμμα αρχίζει στην main.cpp, όπου λειτουργεί σαν controller της main.qml. Σε αυτό το c++ αρχείο ορίζεται η επικοινωνία με το backend

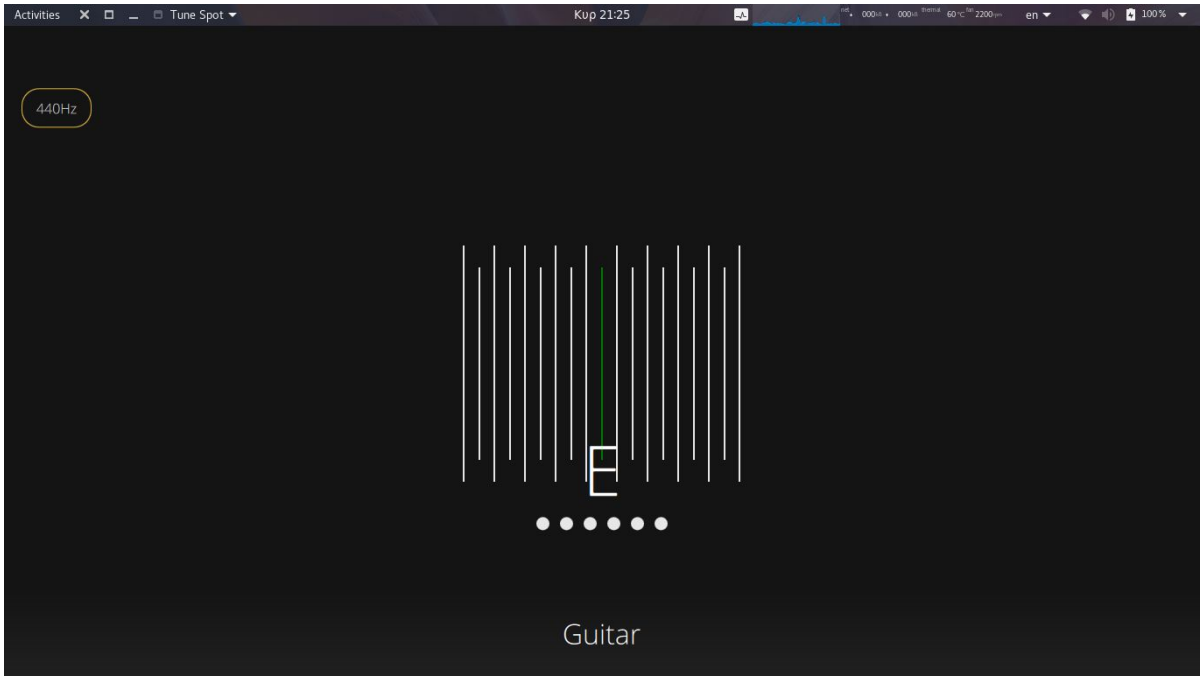
μετατρέποντας με ένα adapter τα αρχεία του backend σε qml types.

- Η main.qml ορίζει την βασική δομή του προγράμματος, κατατάσοντας τα υπό-στοιχεία (components) και ορίζοντας τις σχέσεις μεταξύ τους.
- Κάθε qml component έχει το αντίστοιχο του javascript context. Το qml αναλαμβάνει την εμφάνιση και την κατάταξη των αντικειμένων, ενώ το αντίστοιχο context περιέχει την λογική του component. Για παράδειγμα, το FrequencySelector.qml έχει το context FrequencySelectorContext.js.
- Τα περισσότερα context υλοποιούν μια συνάρτηση onCreate(), όπου καλείται όταν το qml component ολοκληρώσει την δημιουργία του. Σε αυτή τη συνάρτηση δίνονται ορίσματα τα διάφορα αντικείμενα που το context θα μεταλλάξει, καθώς η qt δεν έχει κάποιον άλλο τρόπο πρόσβασης των αντικειμένων από javascript.
- Όσα components χρειάζεται να αλλάζουν κατά την διάρκεια εκτέλεσης της εφαρμογής υλοποιούν το μοντέλο σχεδίασης observer-observable. Με το interface observable να ορίζεται στο Observable.js -το οποίο δεν αποτελεί context κανενός.
- Για τα περισσότερα μεγέθοι των γραφικών στοιχείων χρησιμοποιείται relative sizing στο parent του, που ακολουθώντας αναδρομικά το parent κάθε αντικειμένου φτάνουμε στο μέγεθος της οθόνης. Έτσι η εφαρμογή φαίνεται το ίδιο σε όλες τις οθόνες, ανεξαρτήτως της ανάλυσής τους. Μερικά παραδείγματα θα δούμε το επόμενο σκέλος.

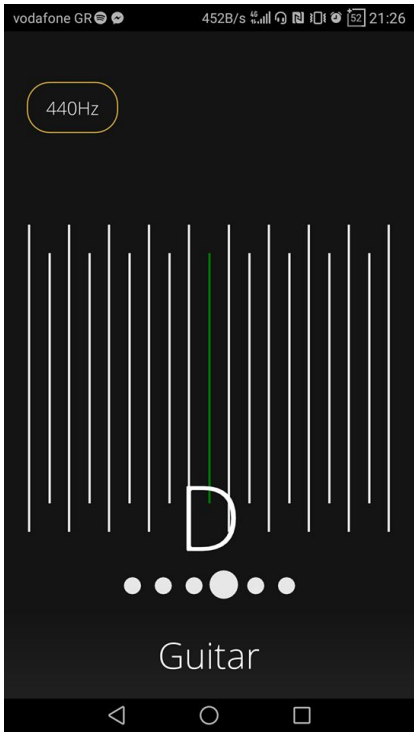
Παραδείγματα σε διάφορες οθόνες/λειτουργικά



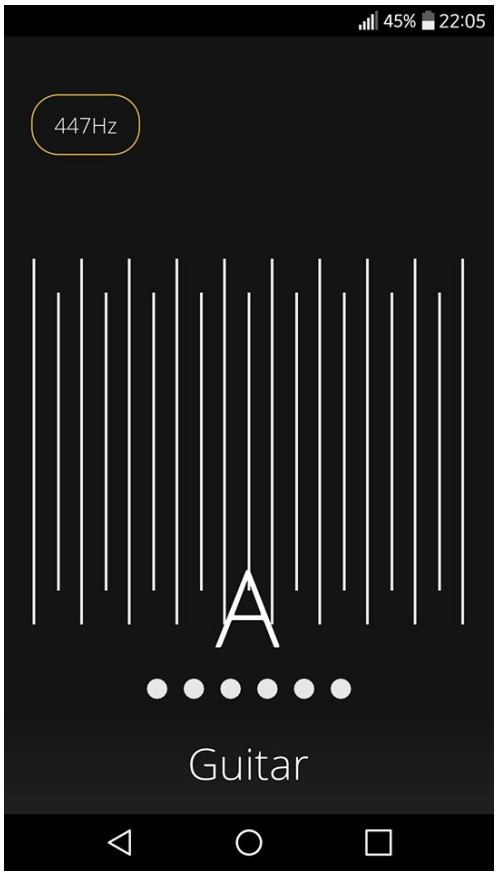
Gnome - normal size



Gnome - full size



Huawei P9 lite - FHD ~ 442 ppi



Lg G4 - 2K ~ 534 ppi

Σχεδίαση/Ανάπτυξη της λειτουργικότητας - backend documentation - Σπυρίδων Τσαλίκης 2812

Το κομμάτι της υλοποίησής μου αποτελείται από 4 κλάσεις.

Η πρώτη κλάση είναι η **NotesController**. Αυτή είναι υπεύθυνη για για τη διαχείριση των νοτών. Αρχικά, διαβάσει τα ονόματα των νοτών και έπειτα είτε διαβάξει το υπάρχον αρχείο των συχνοτήτων είτε δημιουργεί ένα καινούριο με default base frequency = 440. Με την κλάση αυτή μπορεί να αλλαχτεί η base frequency κάνοντας τις απαραίτητες αλλαγές στο αρχείο των συχνοτήτων. Και επίσης να ξητηθούν τα ονόματα των νοτών ώς μεταβλητές, οι συχνοτητες των νοτών και η base frequency τα οποία όλα αυτά είναι αποθηκευμένα και ως μεταβλητές.

Η δεύτερη κλάση είναι η **Recorder**. Αυτή είναι υπεύθυνη για τη δημιουργία ενός αρχείου το οποίο θα αναλυθεί για να βρεθεί η συχνότητα. Το αρχείο αυτό δημιουργείται με 44100 sample rate, 8 sample size, 1 channel, little Endian byte order και unSigned Int sample type και με codec pcm. Έπειτα, η default device του κάθε συστήματος ηχογραφεί για 400ms και αποθηκεύει τα αποτελέσματά της σε ένα αρχείο σε raw μορφη στο οποίο εν συνεχεία με τη χρήση μιας συνάρτησης προστίθεται ο wavHeader της.

Η τρίτη κλάση είναι η **audioFile** ή οποία πάρθηκε από αυτό το repository https://github.com/adamstark/AudioFile και είναι υπεύθυνη για την ανάγνωσή του wav αρχείου.

Η τέταρτη κλάση είναι η **Configurator** η οποία λειτουργεί ως ένας wrapper για τις 3 προηγούμενες και επιτελεί έξτρα διεργασίες που χρειάζονται από το front. Οι διεργασίες είναι οι εξής:

1) η recordSample παράγει ένα αρχείο test. wav
2) η analizeSample φορτώνει το αρχείο με τη χρήση της κλάσης audioFile, μηδενίζει κάποιο θόρυβο στην αρχή και έπειτα βρίσκει την συχνότητα που "άκουσε" στο αρχείο με τη χρήση του αλγορίθμου yin (ο οποίος ευρέθη απο τον 2ο back-end developer).
3) Οι 3 setters setCelloXString, setGuitarXString, setFreeMode είναι υπεύθυνοι για τον ορισμό 2 μεταβλητών, closest Note και percentage of distance from the closest note ώστε το front να μπορεί να απεικονίσει γραφικά το ποια είναι η πλησιέστερη νότα και πόσο απέχουμε από αυτήν με τιμές που περιέχονται στο διάστημα [-100, 100].

Σχεδίαση/Ανάπτυξη της λειτουργικότητας - backend documentation - Θεόδωρος Μπόζίνης 2493

Το κομμάτι της υλοποίησής μου αποτελείται από 3 σετ αρχείων.

Το πρώτο σετ είναι οι συναρτήσεις της Yin η οποία πάρθηκε απο το repository https://github.com/ashokfernandez/Yin-Pitch-Tracking/ και είναι υπεύθυνση για την εύρεση της συχνότητας που "ακούει" από το αρχείο που της ζητάμε.

Το τρίτο σετ περιέχει την κλάση Core που λειτουργεί ώς ένας wrapper για να λειτουργεί το front-end.

Η βασική ιδέα είναι ότι η core χρησιμοποιεί ένα αντικείμενο της κλάσης worker που κληρονομεί από την Thread ώστε η διαδικασία της ηχογράφησης αλλά και της ανάλυσης να γίνονται στο background. Αφού η

ανάλυση γίνει τότε γίνεται emit ένα signal το οποίο έχει όλες τις απαραίτητες πληροφορίες ώστε να ανανεωθεί το front end.