Giorgio Gamba
Matricola 833313
giorgio.gamba@edu.unito.it
Università degli Studi di Torino
Dipartimento di Informatica
Anno Accademico 2020/21
Corso “Programmazione per Dispositivi Mobili”

RELAZIONE PROGETTO “PROGRAMMAZIONE DISPOSITIVI MOBILI” APPLICAZIONE “SIMPLE SAMPLE”

INDICE

1. Idea

Simple Sample nasce dalla volontà di creare un'astrazione di un Akai MPC, strumento per il campionamento di suoni sviluppato durante gli anni '80, che ha portato alla nascita di vari generi musicali, quali Elettronica e Hip Hop. Il suo funzionamento si basa sulla registrazione di frammenti audio in ingresso tramite microfono o tagliando tracce, la loro eventuale modifica e infine la possibilità di comporre nuova musica e ritmiche a partire da questi campioni, detti "Samples", creati precedentemente.

Figura 1: Akai MPC 1000

Simple Sample permette, quindi, di registrare file audio associandoli ai tasti del campionatore (disposti in griglia 4x4) e riprodurli interagendo con gli stessi.

L'applicazione, inoltre, definisce un S*equencer*, anch'esso ispirato al funzionamento dei sequencer usati per controllare Drum Machines e Tastiere negli studi di registrazione, al fine di poter costruire semplici ritmiche con i suoni raccolti precedentemente. Lo scopo di Simple Sample è quello di svolgere il ruolo di "raccoglitore di suoni", permettendo ai suoi utilizzatori di riutilizzare i Samples raccolti per strada all'interno di studi di registrazione (o Home Recording Studios di molti produttori amatoriali) durante i processi di produzione musicale. Spesso, infatti, si deve fare fronte a problemi di trasferimento o reperimento di nuovi suoni da usare per comporre musica a causa di mancata compatibilità di formati di codifica o problemi di connessione alla rete. Simple Sample si pone l'obiettivo di rimuovere questo ostacolo.

2. Deviazioni rispetto alla scheda di presentazione

A causa di problemi con le librerie Firebase, non è stato possibile sviluppare un sistema di notifica per il proprietario di un campione quando un utente decide di scaricarlo o salvarlo nei Preferiti. Al tempo stesso, però, è stato posto in aggiunta a quanto dichiarato nella scheda di presentazione un sistema di caricamento di file audio dal servizio Cloud di default da poter utilizzare all'interno dell'applicazione. Esso permette di poter caricare elementi magari generati tramite l'utilizzo di altre applicazioni, come il registratore posto di serie in ogni dispositivo. Inoltre, all'interno del sistema di condivisione è stato aggiunto un elenco di tag che associa a ogni campione condiviso sulla piattaforma una serie di parole chiave tramite cui facilitare la ricerca agli altri utenti all'interno della schermata "Explorer". In aggiunta viene infine fornito un set di suoni di una Drum Machine Roland 909 preinstallati per creare semplici beat e un supporto di rinominazione dei campioni creati

3. Metodologia di sviluppo utilizzata

La metodologia di sviluppo scelta è la "Test Driven Programming", la quale si basa su un test incrementale dei metodi e delle funzionalità man mano che queste vengono sviluppate. Alla sezione "Test" sono elencate le tipologie di test effettuati durante lo sviluppo.

4. Funzionalità dell’applicazione

L’applicazione fornisce diverse funzionalità:

#		Funzionalità
1		**Registrazione**: Possibilità di registrare tramite il microfono del dispositivo suoni dell’ambiente e associarli automaticamente a un tasto dell’applicazione. L’interazione consiste nel mantenere premuto il testo a cui voler associare il suono che si andrà a registrare, attivando così il registratore. Dopo aver rilasciato il tasto, il file audio (in formato “.wav”) creato viene automaticamente salvato all’interno del filesystem del dispositivo nella cartella predefinita. Il tasto utilizzato per la registrazione cambia colore (da verde a rosa) e mostra il nome del file, generato automaticamente. Ripremendo lo stesso tasto è possibile riprodurre il suono associato ed è possibile riprodurre contemporaneamente più suoni ma registrarne solo uno per volta
2		**Rinominazione dei campioni**: nella schermata “Sampler” è presente il tasto “Rename” che, dato un tasto di colore viola, a cui quindi è stato associato un file audio, permette di rinominare tale file associato (e quindi anche ciò che viene mostrato sul tasto)
3		**Condivisione dei campioni**: Se l’utente è registrato alla piattaforma, può decidere di condividere i propri campioni con gli altri utenti iscritti a Simple Sample, associando a tale file audio un nome personalizzato e una serie di tag tramite cui filtrare la ricerca all’interno dell’explorer
4		**Upload su Google Drive**: se l’utente è iscritto alla piattaforma tramite account Google, gli è concessa la possibilità di caricare i propri file registrati su Drive. Questo risulta essere molto utile per tutti i produttori musicali che, collezionando i campioni tramite Simple Sample, vogliano poterli avere facilmente a disposizione sulla propria stazione di lavoro dello studio di produzione, scaricandoli.
5		**Caricamento di file dal filesystem**: è possibile associare ai bottoni del Sampler file prelevati dal filesystem. Sono inoltre forniti di default dei campioni di una batterie Roland TR-909 (classica drum machine della musica House).
6		**Sequencer**: una delle schermate principali dell’applicazione, fornisce agli utenti una griglia per la programmazione della riproduzione automatica dei suoni associati ai bottoni. E’ possibile comandare tale sistema tramite tasti di Play, Stop e Pausa. E’ possibile resettare tutta la programmazione effettuata tramite il pulsante “Reset” e modificare i BPM (battiti per minuto), ovvero la velocità di riproduzione dei suoni.
7		**Schermata Utente**: Una volta iscritto alla piattaforma, l’utente ha accesso a una schermata contenente alcune informazioni, sia condivise con gli altri utenti che private. In tale schermata sono mostrati, oltre all’immagine del profilo, due elenchi: i samples condivisi con gli altri utenti, con la possibilità di riprodurli immediatamente, e i samples invece segnati come Preferiti, ovvero registrazioni audio proprie o di altri utenti etichettate come “preferite”. In questa schermata è possibile sia riprodurli immediatamente che rimuoverli dai preferiti. È inoltre visibile il “numero di downloads”, ovvero il numero totale di volte che i campioni condivisi dall’utente sono stati scaricati.
8		**Immagine del profilo**: Possibilità di caricare una immagine del profilo tramite uso della fotocamera oppure prelevandola dal filesystem
9		**Explorer**: Una delle schermate principali, formata da una lista di tutti i campioni condivisi sulla piattaforma dai vari utenti. Per ogni campione, ne viene visualizzato il nome, la lista di tag associata, il numero di downloads effettuati. L’utente può svolgere su ogni campione le seguenti operazioni; riprodurlo per ascoltarlo, scaricarlo nel proprio filesystem (e quindi successivamente associarlo a un bottone del Sampler caricandolo dallo stesso) o segnarlo tra i proprio preferiti. E’ possibile accedervi solo se l’utente è registrato

10		**Sistema di filtraggio**: Nell'Explorer è possibile selezionare quali campioni visualizzare in base al nome oppure ai tag associati
11		**Iscrizione tramite servizi Google**: l'utente può decidere di iscriversi alla piattaforma tramite il proprio account Google e poter quindi usufruire dell'upload su Drive
12		**Iscrizione tramite indirizzo email**: L'utente può iscriversi a Simple Sample facendo utilizzo di un indirizzo email e una password. Una volta effettuata la registrazione, egli potrà accedere alla pagina utente solo dopo aver verificato l'indirizzo email. Viene infatti inviato un messaggio di posta elettronica contente un link di verifica con cui interagire
13		**Username**: L'utente può decidere di inserire un proprio username personale

Tabella 1: Elenco delle funzionalità dell'applicazione

5. Architettura dell'applicazione

L'applicazione è stata sviluppata tramite architettura MVC (Model - View - Controller), la quale permette di poter separare in modo netto i vari interessi dell'applicazione. Il pattern MVC consiste nel suddividere il codice dell'applicazione nei seguenti elementi:

1. **Model**: Classe univoca che ha l'obiettivo di mantenere la persistenza dei dati che governano l'app. Alcuni degli obiettivi sono, nel caso di Simple Sample, di controllare la scrittura su filesystem dei dati registrati o mantenere informazioni sull'utente correntemente connesso.
2. **View**: Sviluppa la vista dell'app, il cui unico obiettivo è quello di presentarne i dati. Il codice di una vista avrà come compito quello di richiamare determinati metodi dei controller associati per il prelievo dei dati e la gestione delle operazioni nate dall'interazione dell'utente con la schermata. La vista si modificherà nel caso di cambiamento dei dati presentati
3. **Controller**: Classe che collega Vista e Modello e che, insieme a quest'ultimo, contiene la business logic dell'app, determinando quindi come i dati devono essere maneggiati e come deve comportarsi l'app. Inoltre, i controller hanno il compito di eseguire operazioni in risposta alle interazioni dell'utente con la vista del progetto, andandola a modificare in base ai nuovi dati generati e creando, quindi, la logica di funzionamento dell'app stessa.

Nella mia implementazione, i controller sono tutti stati sviluppati come *Singleton*, ovvero come un oggetto che può essere istanziato solo una volta. Questo mi permette di evitare di avere più istanze della stessa classe che possono controllare l'interfaccia utente e il modello, delegando quindi un solo oggetto a gestire un determinato aspetto del sistema. Inoltre, seguendo questa logica, non si corre il rischio di inizializzare più volte uno stesso oggetto, perdendo quindi la consistenza dei dati generati. Singleton porta anche il vantaggio di avere un avvio più rapido e un funzionamento generalmente più leggero sulle risorse del dispositivo qualora l'utente non facesse uso di tutte le funzionalità offerte dall'applicazione, poiché determinati controller vengono inizializzati solo quando è richiesto il loro intervento per specifici compiti.

Questo pattern architetturale porta con sé il vantaggio di scrivere codice più pulito e di separare molto bene i compiti delle varie istanze che costituiscono l'applicazione. In questo modo, evitiamo che gli oggetti siano troppo accoppiati (ovvero che per il funzionamento di uno sia necessario lo stretto intervento di un altro). Il voler creare codice tramite questo pattern architetturale aiuta lo sviluppatore e progettare classi in un giusto numero che svolgano ruoli ben distaccati e differenziati. Altro vantaggio, inoltre, è la gestione della persistenza e consistenza dei dati da parte di un singolo oggetto (il Modello), evitando quindi che vi siano problemi con la manipolazione degli stessi o inconsistenze durante l'uso dell'app.

6. Strutture di salvataggio dei dati online

La parte di salvataggio dei dati sui server online di Google è eseguita tramite il supporto delle librerie di Firebase. Esse, dopo aver effettuato la registrazione delle applicazione Android e iOS, permettono di attivare diversi servizi, quali Firebase Storage o Autenticazione. Al fine di poter

salvare i dati da condividere e rendere disponibili a tutti gli utenti, sono state definite due strutture di organizzazione dei dati mostrate attraverso i due schemi qui di seguito:

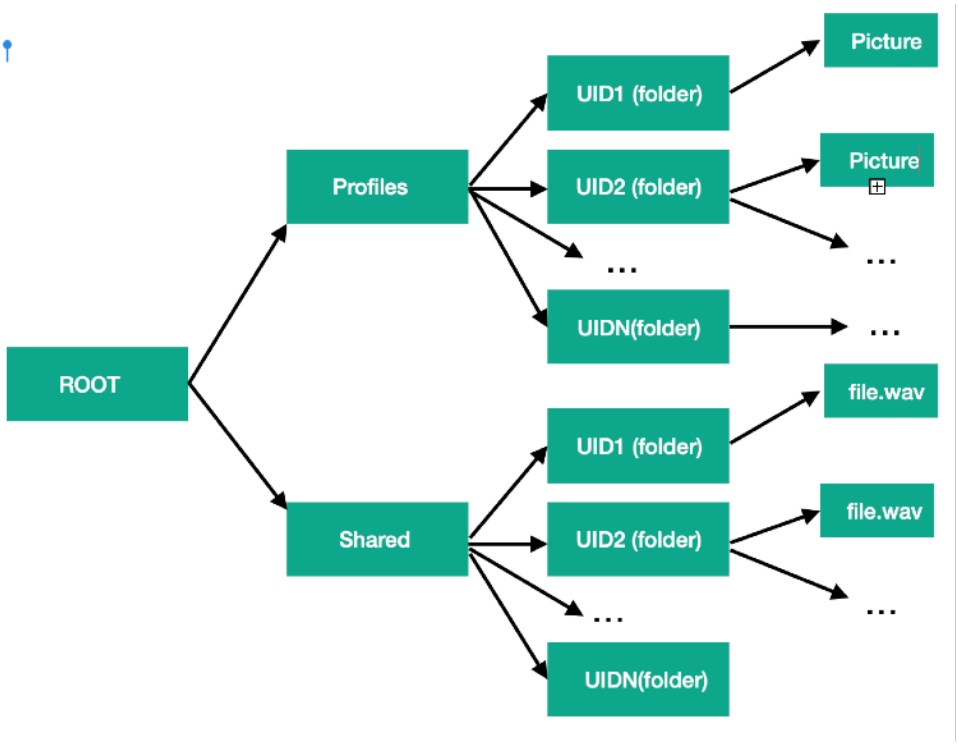

Figura 2: Struttura organizzazione dei dati di Firebase Storage

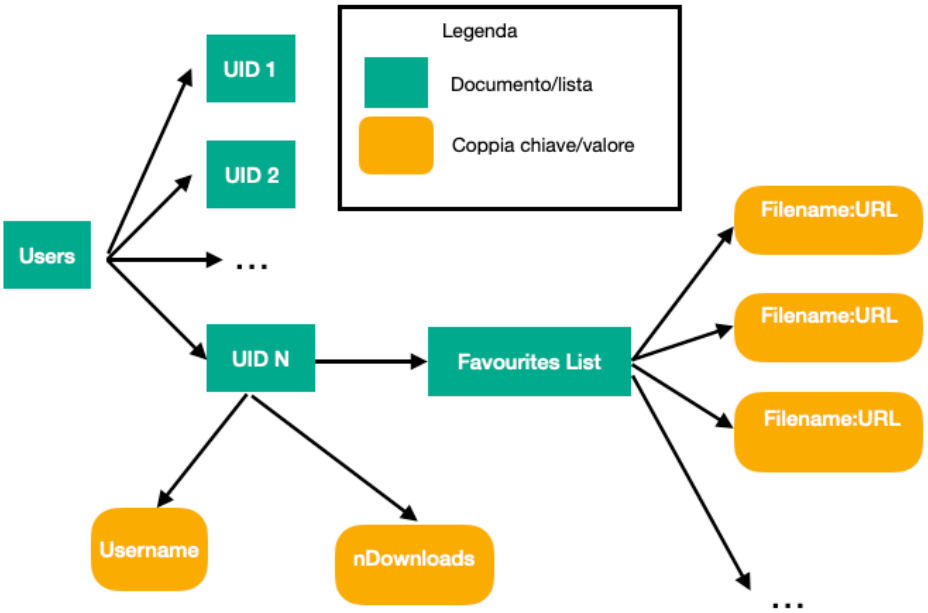

Figura 3: Schema organizzazione dei dati di Firestore Storage

Le strutture dati create si basano sul modo in cui i vari servizi Firebase gestiscono i dati. Per quanto riguarda Firebase Storage, l'organizzazione è formata da una struttura del tipo chiave/valore, contenente anche collezioni e documenti. Data questa organizzazione, ho deciso di

salvare al suo interno dati di tipo primitivo, come lo username o il numero totale di downloads di un utente, e i preferiti.

Per quanto riguarda invece il salvataggio Cloud, esso è basato sulla definizione di un filesystem di cartelle contenenti al proprio interno documenti e file. Proprio grazie a questa struttura è possibile salvare in esso i file audio registrati dai vari utenti, come anche le immagini del profilo scelte dagli stessi. Ho deciso di utilizzare due basi di dati differenti al fine di sfruttare al meglio tutte le possibilità che Firestore può offrire attraverso i suoi servizi.

7. Librerie utilizzate

Simple Sample fa uso di diverse librerie, elencate all'interno del file *pubspec.yaml*, il quale contiene molte delle informazioni di base dell'applicazione. Una volta inserito il codice di un pacchetto, deve essere eseguito il comando "*flutter get*", che si occupa di scaricare i file appositi, ma soprattutto di controllare le dipendenze tra le varie librerie. E' in seguito a questo comando che si capisce se è possibile utilizzare un determinato insieme di librerie, e i controlli delle dipendenze hanno determinato un ruolo fondamentale nella scelta di quali di esse utilizzare.

I codici importati sono:

#	Nome	Descrizione
1	**AudioPlayers**	Libreria per la riproduzione di file audio sia da remoto che dentro dal filesystem del dispositivo. Simple Sample fa uso di entrambe queste funzionalità. La riproduzione è impostata su "LOW LATENCY", in modo da poter gestire sia la pressione ripetuta velocemente di un tasto del Sampler sia un valore di BPM alto nel Sequencer. L'applicazione fa uso di 16 istanze di Audioplayers (una per tasto/canale). Vantaggi dell'utilizzo di questa libreria sono sia il piccolo consumo di risorse in presenza di istanze multiple sia il rilascio automatico a fine riproduzione. E' la libreria più popolare tra gli sviluppatori Flutter nell'ambito della riproduzione di file audio dato il suo poco dispendio di risorse e facilità di utilizzo
2	**Flutter_sound**	Libreria per la gestione audio che fornisce sia strumenti per la registrazione che per la riproduzione. Simple Sample fa uso della prima funzionalità. Viene infatti stanziato un oggetto *Recorder* quando aperto il Sampler e rilasciato quando chiuso. Tramite questo oggetto ogni tasto può registrare un file audio (infatti si può registrare un solo tasto per volta) che viene automaticamente salvato nel filesystem del dispositivo. Difetti di questa libreria sono molteplici, come il notevole uso di risorse per gli oggetti istanziati, o la mancanza della possibilità di registrare da input che non siano il microfono. Prima di arrivare alla scelta di utilizzare questa libreria sono state testate diverse sue concorrenti. Tutte quante, compresa Flutter_sound, presentano molte limitazioni, ma quest'ultima è stata scelta tra tutte per il minor dispendio di risorse e le codifiche (".mp3" e ".wav") permesse
3	**Firebase_Auth**	Pacchetto di codici che permette di effettuare la registrazione di un utente ai sistemi Firebase, su cui Simple Sample si appoggia. Questa libreria offre metodi molto semplici per gestire iterazioni complesse come possono essere quelle di iscrizione a un portale o accesso allo stesso.

4	**Google_Apis**	Libreria che permette sia l'iscrizione a un portale tramite il proprio account Google sia la creazione di un collegamento tra un'applicazione e i servizi Drive. Al momento del "sign in", Simple Sample richiama l'applicazione di Google di default del sistema per effettuare il login. Una volta completata, tale app ausiliaria viene rilasciata ritornando a Simple Sample
5	**File_Picker**	Libreria estremamente utile per il reperimento di file. Essa richiama automaticamente il filepicker proprio del sistema operativo in uso, con anche la possibilità di reperire i dati stessi da Google Drive, nel caso di Android, o iCloud nel caso di iOS, ottenendo quindi due diverse funzionalità tramite un solo codice sviluppato. Questo pacchetto offre inoltre la possibilità di porre dei filtri di selezione basati sull'estensione dei file. Nel caso di Simple Sample, infatti, gli unici file che è possibile selezionare per l'importazione sono i ".wav", ".mp3", ".m4a" e ".aac", che sono anche le uniche tipologie di file gestibili dalle istanze di Audioplayers. E' la miglior libreria per l'import di file, data il vantaggio di dover utilizzare solamente un metodo e la possibilità di applicare filtri per la selezione
6	**Firebase_storage**	Pacchetto sempre sviluppato da Firebase per la gestione degli upload e dei download dal "Firebase Storage", base di dati usata da Simple Sample per il salvataggio dei file audio condivisi dagli utenti. Permette un interfacciamento rapido e una semplice gestione della gerarchia di file creata all'interno del database
7	**Cloud_firestore**	Libreria per la gestione del database "Firestore", base di dati che organizza i dati secondo un sistema "chiave-valore" sui cui Simple Sample salva le informazioni dell'utente, come username o preferiti.
8	**Firebase_core**	Libreria che implementa le funzionalità di base di Firebase senza la quale non sarebbe possibile utilizzare tutte le librerie Firebase sopra citate. Necessita solamente di essere presente tra le dipendenze del progetto e di essere attivata all'avvio dell'applicazione.

Tabella 2: Elenco delle librerie utilizzate

8. Supporto per device multipli

L'interfaccia grafica di un'applicazione in Flutter è definita secondo una gerarchia di oggetti Widget, i quali formano un'albero n-ario. In Simple Sample, tutti i widget hanno dimensioni definite a partire dalle dimensioni dello schermo, reperibili all'interno del metodo "build" di ogni widget attraverso il comando

```
MediaQuery.of(context).size.width //restituisce la larghezza dello schermo del dispositivo in uso
MediaQuery.of(context).size.height //restituisce l'altezza dello schermo del dispositivo in uso
```

Date queste due grandezze, le dimensioni dei Widget vengono calcolate secondo il rapporto tra esse e alcuni valori assoluti definiti empiricamente, creando quindi l'interfaccia grafica in modo relativo e rendendo invariati i rapporti di dimensioni e distanze tra i vari Widget su schermi di dimensioni differenti.

Data l'idea che sta alla base dell'applicazione, l'interfaccia grafica di Simple Sample è stata definita solo per smartphone. Qualora si volesse implementarla anche per tablet, è possibile costruirla semplicemente all'interno di un Widget definito appositamente per tale tipologia di

dispositivo. La scelta di quale schermata utilizzare verrà poi effettuata tramite le informazioni prelevate nel metodo “build" attraverso l’oggetto MediaQuery.

9. Supporto per lingue diverse

Simple Sample supporta la lingua Inglese e quella Italiana. Data la struttura del codice sviluppato per l’assistenza multilinguistica, risulta facile implementare il sorgente per altre lingue qualora lo si desiderasse.

Il servizio per le lingue è sviluppato a partire dalla dipendenza “flutter_localizations”, la quale fondamentalmente è in grado di utilizzare la localizzazione dello smartphone per decidere quale linguaggio utilizzare. In ogni caso, è stato comunque sviluppato un piccolo menu per la scelta manuale, accessibile tramite l’interfaccia “Settings” della Pagina Utente.

Alla base, il supporto multilinguistico parte da “Languages”, classe astratta che deve essere implementata da tutte le classi rappresentanti una determinata lingua. Infatti, per ognuno dei linguaggi supportati è stata sviluppata una classe concreta chiamata “Languages[SiglaLingua]” che sviluppa tutti i metodi di “Languages”. I metodi contenuti in quest’ultima sono semplicemente dei getter che vengono richiamati dalle interfacce grafiche e che, a seconda del contesto utilizzato (e quindi della lingua correntemente in uso), utilizzeranno l’implementazione corretta che restituirà una stringa con la traduzione corretta. Le interfacce grafiche possono richiamare queste informazioni tramite il comando

```
Languages.of(context).[nome del getter]
```

Al fine di poter fornire un servizio di traduzione responsive per le esigenze dell’utente, il quale può così cambiare in qualunque momento la lingua utilizzata, il widget principale dell’applicazione “MyApp” è stato sviluppato come “Stateful Widget”, ovvero un widget che può contenere al proprio interno uno stato e in base al valore da esso assunto comportarsi in modo differente. Una volta cambiata lingua, infatti, viene richiamato il metodo “setLocale” che modifica lo stato di tale widget tramite il metodo “setState”. A seguito di questa operazione, tutti i widget figli di MyApp (quindi tutta l’applicazione) verranno ricostruiti facendo uso delle stringhe di testo corrette.

10. Gestione audio

Tutti i file registrati tramite Simple Sample sono codificati in “.wav”, il quale è il formato audio universalmente utilizzato nell’ambito musicale, poiché esso non porta a compressioni di alcun tipo. Quindi, sebbene esso occupi più spazio di un file mp3 classicamente usato nei dispositivi mobili, risulta essere il formato più adatto, ricordando lo scopo per cui Simple Sample è stato creato. Per quanto riguarda, invece, la riproduzione, le codifiche audio che possono essere utilizzate sono “.wav”, “.mp3”, “.m4a” e “.aac” al fine di supportare i file registrati anche con le applicazioni di note vocali fornite di default sui dispositivi.

11. Struttura del codice

Tutta l’applicazione è gestita tramite un widget MyBottomNavigationBar, che implementa la barra di navigazione della parte bassa dello schermo. Tramite questo widget vengono istanziate le 4 interfacce grafiche principali che formano l’applicazione.

Il codice, scritto in Dart, definisce widget per la creazione dell’interfaccia grafica e classi Singleton per la gestione della Business Logic. Esso è organizzato e strutturato secondo i criteri del Pattern Model-View-Controller.

Qui di seguito è presentato uno schema delle classi principali e di come queste collaborano tra loro:

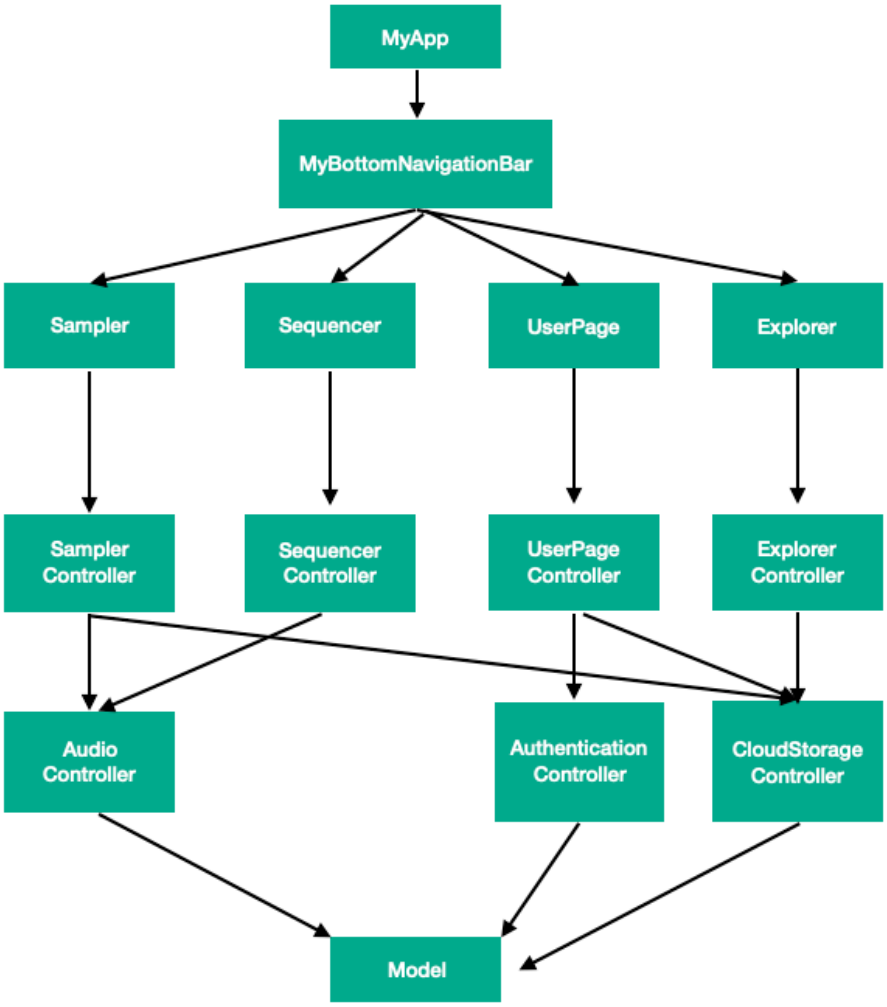

Figura 4: Classi principali e relazioni tra esse

11.1 Modelli

Simple Sample, per la gestione dei dati, fa utilizzo di due classi:

11.1.1 Model

Costituisce il modello dell'applicazione ed è stato sviluppato secondo il pattern Singleton. Inizializzato all'avvio dell'applicazione, tra le altre cose ha il compito di tenere traccia delle registrazioni "attive", ovvero quelle correntemente associate a un tasto, tramite una lista di 16 elementi Record, una per ogni bottone. Questa classe, inoltre, mantiene al proprio interno un riferimento a un oggetto "User" (della libreria *firebase_auth*) rappresentante un utente registrato e collegato alla piattaforma Firebase. E' tramite questo oggetto che Simple Sample riesce a gestire le operazioni di autenticazione e aggiornare le pagine di conseguenza. Compito del Modello è anche quello di generare i path da associare ai vari campioni creati durante l'utilizzo dell'applicazione.

Al fine di mantenere la consistenza tra le registrazioni eseguite in avvii diversi dell'applicazione, il Model tiene traccia di una variabile int "contatore", inizializzata a ogni avvio tramite la lettura di un valore prelevato dal documento "Shared Preferences", in modo che venga mantenuta traccia di questo valore anche in seguito all'arresto di Simple Sample. Questo numero intero viene usato per la creazione dei path creati automaticamente in fase di registrazione di suoni. A ogni avvio del registratore, quindi, verrà creato un file con indirizzo "*[path_cartella_di_salvataggio]/[valore_contatore].wav*". In questo modo, avremo sempre path differenti, ed eviteremo sovrascritture di file date dall'assegnamento dello stesso path.

11.1.2 Record

Rappresenta un “Sample” di Simple Sample. Contiene informazioni di gestione del campione, quali: nome, indirizzo (che può essere sia locale che remoto), proprietario (identificativo univoco dell’utente fornito da Firebase) e tag associati. Il reperimento delle informazioni per istanziare questi Record, come il file path, avviene tramite il Model.

11.2 Views

Sono presenti 4 Widget principali, che formano le viste dell’applicazione:

11.2.1 Sampler:

Stateful Widget (widget dotato di stati e che modifica il proprio comportamento in base a essi) che tramite il metodo “build” definisce un elemento “Column” con 4 elementi figli “Row”, ognuno contenente i 4 bottoni che compongono una riga del Sampler. Viene inoltre aggiunta, alla fine della griglia, una riga contenente tutti i tasti ausiliari. Tutte le operazioni richiamate dai bottoni sono delegate al “SamplerController", il quale si occupa della logica di tutte le operazioni effettuabili tramite il Sampler.

Ogni bottone del Sampler è un oggetto “Stack” che effettua azioni diverse a seconda dello stato in cui si trova il Sampler in un dato momento, e quindi dell’operazione in corso di esecuzione. Nel caso di rinomina in corso, ad esempio, se al bottone è associata una registrazione, allora viene mostrata la spunta di selezione per l’oggetto che, se scelto, apre un dialog contenente gli strumenti per la rinomina dell’oggetto.

11.2.2 Sequencer:

La struttura del Sequencer è strettamente collegata al Sampler, infatti ogni sua riga è associata a un tasto del Sampler, e attivare il bottone alla riga *i* e colonna *j* significa che l’utente vuole che all’istante *j* il bottone *i* suoni il campione associato. E’ ovviamente possibile suonare solamente quei bottoni che hanno una registrazione associata. La costruzione dell’interfaccia grafica lavora in modo analogo al Sequencer, per cui essa è costituita da un oggetto “Column” che possiede come figli, tra gli altri, tanti oggetti Row quante sono le righe che compongono il sequencer stesso. Ogni riga contiene un widget Text (di colore rosa se a tale tasto è associata una registrazione Record) e 8 widget ElevatedButton che costituiscono i tasti di una riga del Sequencer. Inoltre, il Sequencer possiede in cima un oggetto Row “puntatore” che indica all’utente la colonna riprodotta in un certo istanze e uno al fondo contenente i tasti per il controllo del Sequencer. Il puntatore cambia posizione in modo ciclico all’infinito, ripartendo da 0 dopo aver passato la posizione 7.

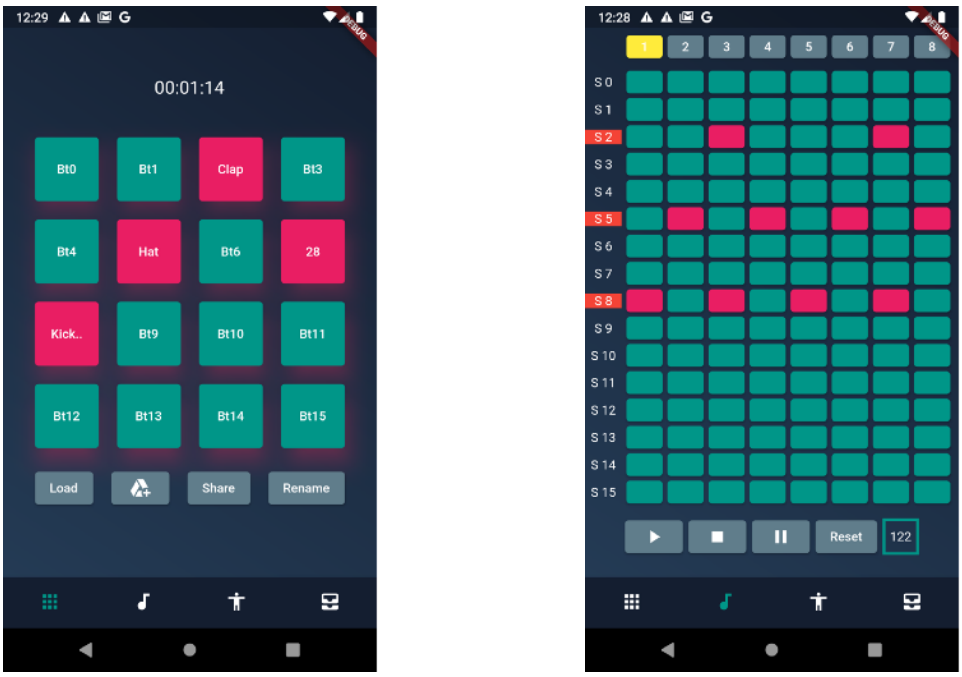

Figura 5: Schermata “Sampler” e schermata “Sequencer”

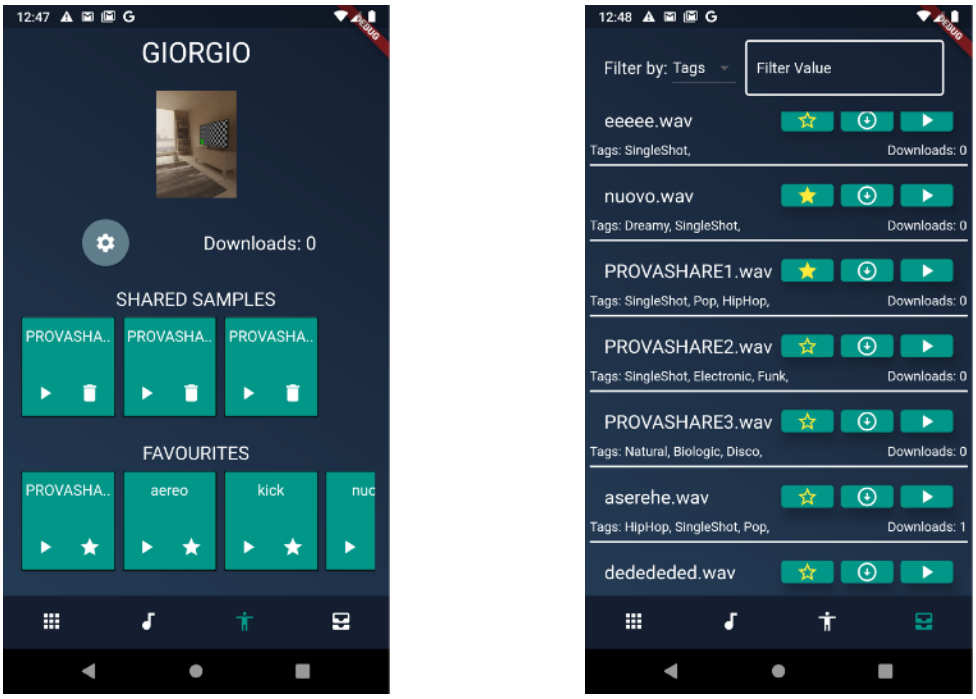

Figura 6: Schermata “User Page” e schermata “Explorer”

11.2.3 User Page

Widget usato per mostrare le informazioni utente. In tale pagina sono resi visibili: lo username, la sua immagine del profilo, il numero totale di downloads, un bottone per accedere al menu impostazioni e due liste orizzontali. Le due liste hanno un aspetto grafico identico ma due funzioni differenti: la prima mostra l’insieme dei Record condivisi dall’utente, mentre la seconda raffigura tutti quelli che l’utente ha salvato nei Preferiti. Entrambe le liste contengono al proprio interno elementi “SquareListItem” che mostrano il nome del Sample e due tasti funzione.

11.2.4 Explorer

Widget Column contenente come figli una barra di ricerca, che permette il filtraggio degli elementi tramite ricerca per tag o ricerca per nome, e una lista di istanze definite dalla classe “ExplorerItem”. Ognuno di questi widget facenti parte della lista contiene al proprio interno il nome del record, la lista di tag associata, il numero di downloads (tutti espressi tramite widget “Text”) e un oggetto “Row” contenente al proprio interno 3 tasti, che implementano le operazioni di salvataggio tra i preferiti, download e riproduzione.

11.3 Controllers

All’interno di Simple Sample sono presenti vari controllers, ognuno dedito a eseguire operazioni in un singolo ambito, al fine di poter ridurre al minimo le responsabilità di ogni oggetto e modulare il codice a dovere. Ogni controller è stato progettato come Singleton, secondo un pattern di questo tipo:

```
static final AudioController _instance = AudioController._internal();

Factory AudioController() {
        return _instance;
}

AudioController._internal() {
        /*Codice di inizializzazione della classe AudioController*/
}
```

Il quale mostra come ogni oggetto contenga al proprio interno un riferimento a una sua istanza che viene inizializzato solo alla prima invocazione e restituito in tutte le successive.

Possiamo suddividere i controller in due gruppi: classi che eseguono servizi di background, e classi invece dedite alla gestione delle operazioni attivate dalle interfacce grafiche.

Per quanto riguarda i servizi in background, le classi create sono:

11.3.1 CloudStorageController:

Classe Singleton il cui compito è quello di gestire le iterazioni con Firebase Storage. In questo codice sono infatti presenti diversi metodi per l’upload e il download di istanze di Record, salvate su Firebase Storage come file audio, con ognuna associata un pacchetto di metadati personalizzati contenenti: codice del proprietario, numero di downloads, lista dei tag associati.

11.3.2 AuthenticationController:

Codice dedito alla gestione dell’autenticazione dell’utente, sia tramite Google sia tramite le credenziali email e password. Questa classe, quando effettuato il primo accesso alla pagina “User Page”, esegue automaticamente un login a Firebase. Se andato a buon fine, restituisce la pagina contenente le informazioni utente, altrimenti mostra il Widget contenente gli elementi per effettuare il login o la registrazione al portale.

11.3.3 AudioController:

Classe dedita alla completa gestione del motore audio, il quale è composto da 1 oggetto FlutterSoundRecorder e 16 AudioPlayers, uno per ogni bottone del Sampler. Quest’ultima lista di oggetti viene utilizzata sia dal Sampler che dal Sequencer. Ho scelto di istanziare 16 oggetti identici al fine di permettere la riproduzione simultanea di più suoni, essendo che ogni AudioPlayer può riprodurre un solo file audio a un determinato URL per volta. Ho deciso di mantenere un solo FlutterSoundRecorder perchè logicamente è necessario registrare un solo file audio per volta. Grazie a AudioPlayers è possibile riprodurre sia file audio contenuti localmente che presenti in remoto.

I controllers invece dediti alla gestione delle interfacce grafiche sono:

- SamplerController
- SequencerController
- ExplorerController
- UserPageController

Tutti i codici dei controllers sono classi Singleton che dichiarano e implementano metodi per la gestione delle operazioni eseguibili tramite l'interazione con il dispositivo. Questi possono a loro volta fare utilizzo di metodi di altri controller (come ad esempio quelli per le operazioni in background) oppure comunicare con il Model.

Simple Sample fa anche utilizzo di alcune istanze di AlertDialog, le quali implementano le interfacce grafiche per lo svolgimento di alcune funzionalità "specifiche", come la condivisione di un campione, o la sua rinominazione.

12. Test

Lo sviluppo dell'applicazione ha seguito un approccio TDD (Test Development Driven), basato sull'implementazione dei singoli metodi e il subito successivo testing degli stessi.
Il testing è basato su 4 diversi livelli:

1. Unit Test: test di singoli metodi non banali
2. Widget Test: test dei widget di Flutter
3. Integration Test: verifiche che includono l'azione combinata di più metodi che lavorano al fine di raggiungere un obiettivo comune
4. End-to-End test: test che simula l'interazione dell'utente con l'applicazione

Al fine di poter eseguire queste procedure, sono state utilizzate le librerie proprietarie di Flutter come *flutter_test*. A causa di problemi di compatibilità con tutte le librerie utilizzate, non è stato possibile effettuare gli End-to-End test data l'impossibilità di poter utilizzare le librerie di Flutter (*integration_test)* per la simulazione.

L'applicazione è stata sviluppata tramite simulatori Android Google Pixel 2 con sistema operativo Android 9.0 e livello di API 28, mentre per la parte iOS è stato utilizzato il simulatore di iPhone 12 con sistema operativo iOS 14.

13. Conclusioni

Simple Sample è un'applicazione che permette la raccolta e condivisione di campioni con utenti collegati attraverso la rete. Grazie ad essa, è possibile creare una community di appassionati di produzione musicale, che possono trovare in Simple Sample un luogo di condivisione di idee e magari un'ispirazione per creare nuovi modi di fare musica.

E' facile pensare a nuovi sviluppi di questo progetto, come la ricerca degli utenti collegati alla piattaforma, l'applicazione di effetti audio ai campioni registrati e la registrazione delle composizioni create tramite il Sequencer, per citarne alcune.