SŠPU
STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

FretFly: Kytarová aplikace ve frameworku Flutter

Autor: Jakub Halama

Obor: 18-20-M/01 INFORMAČNÍ TECHNOLOGIE
se zaměřením na počítačové sítě a programování

Třída: IT4

Školní rok: 2025/26

Prohlášení

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým a prezentačním účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 5. 1. 2026

. . . . . .

Podpis autora

Abstrakt

Práce se zabývá tvorbou mobilní aplikace pro kytaristy s nástroji pro učení a cvičení. Součástí práce byla implementace ladičky s detekcí frekvence, metronomu s nastavitelným tempem a databáze kytarových akordů s vizualizací hmatníku. Výsledkem je funkční mobilní aplikace vytvořená ve Flutteru, která umožňuje kytaristům ladit nástroj pomocí mikrofonu, cvičit rytmus s metronomem a procházet rozsáhlou databázi akordů s diagramy hmatníku. Aplikace obsahuje autentizační systém s podporou přihlášení přes email, uživatelský profil se statistikami pokroku včetně sledování naučených akordů a počtu dní v řadě. Mezi hlavní funkce patří ladička s analýzou frekvence pomocí zero-crossing detection, metronom s nastavitelným BPM (40–240) a taktem, databáze akordů s vyhledáváním a filtrováním podle základního tónu, a systém pro označování naučených akordů. Aplikace využívá Firebase pro autentizaci a ukládání dat v Cloud Firestore, podporuje dark mode a light mode.

Klíčová slova

mobilní aplikace, Flutter, Firebase, kytara

Abstract

This thesis focuses on the development of a mobile application for guitarists that provides tools for learning and practice. The work included the implementation of a tuner with frequency detection, a metronome with adjustable tempo, and a database of guitar chords with fretboard visualization. The result is a fully functional mobile application developed in Flutter that allows guitarists to tune their instrument using the device microphone, practice rhythm with a metronome, and browse an extensive chord database with fretboard diagrams.

The application includes an authentication system with support for email sign-in, a user profile with progress statistics, including tracking learned chords and consecutive practice days. The main features include a tuner with frequency analysis using zero-crossing detection, a metronome with adjustable BPM (40–240) and time signature, a chord database with search and filtering by root note, and a system for marking learned chords. The application uses Firebase for authentication and data storage in Cloud Firestore and supports both dark and light modes.

Keywords

mobile application, Flutter, Firebase, guitar

Obsah

Úvod

Mobilní aplikace určené pro výuku hry na hudební nástroje jsou dnes běžně využívány začínajícími i pokročilými hudebníky. Ani já nejsem výjimkou, a proto bylo cílem této práce vytvořit mobilní aplikaci, která nabídne základní nástroje, jež jsou v současnosti nedílnou součástí hry na kytaru.

Téma bylo zvoleno z důvodu osobního zájmu o hru na kytaru a zároveň snahy vytvořit praktickou aplikaci s reálným využitím. Projekt byl realizován ve frameworku Flutter a využívá platformu Firebase pro správu uživatelských dat a službu Cloud Firestore pro ukládání databáze kytarových akordů.

Hlavním cílem práce bylo vytvořit mobilní aplikaci FretFly, která nabídne základní nástroje pro efektivní cvičení na kytaru. Mezi tyto nástroje patří ladička využívající mikrofon mobilního zařízení, metronom s nastavitelným tempem a databáze kytarových akordů s přehledným zobrazením prstokladu na hmatníku. Součástí aplikace je také uživatelský účet, který umožňuje ukládat informace o pokroku, například naučené akordy nebo počet dní cvičení v řadě.

1 Flutter a mobilní vývoj

1.1 Co je to Flutter

Flutter je open-source framework od společnosti Google určený pro vývoj multiplatformních aplikací. Umožňuje vytvářet mobilní aplikace pro operační systémy Android a iOS z jedné společné kódové základny, což zjednodušuje vývoj i následnou údržbu aplikace. Framework využívá programovací jazyk Dart a je založen na vlastním vykreslovacím enginu, který zajišťuje vysoký výkon a plynulé uživatelské rozhraní.

Flutter poskytuje rozsáhlou sadu předpřipravených grafických komponent (widgetů), díky nimž je možné rychle vytvářet moderní a responzivní uživatelská rozhraní. Výhodou je také funkce hot reload, která umožňuje okamžitě zobrazit změny v kódu bez nutnosti znovu spouštět celou aplikaci, což výrazně urychluje vývoj.

1.2 Výhody Flutter frameworku

- vývoj aplikací pro Android i iOS z jedné společné kódové základny,
- vysoký výkon díky vlastnímu vykreslovacímu enginu,
- moderní a konzistentní uživatelské rozhraní napříč platformami,
- otevřený (open-source) framework s aktivní komunitou,
- dobrá integrace s dalšími službami, například Firebase.

1.3 Programovací jazyk Dart

1.3.1 Co je to Dart

Dart je moderní, objektově orientovaný programovací jazyk vyvinutý společností Google. Používá se především pro vývoj aplikací ve frameworku Flutter. Dart je navržen tak, aby byl jednoduchý na čtení a psaní, podporuje statické typování, asynchronní programování a nabízí vysoký výkon díky kompilaci do nativního kódu pro mobilní i webové aplikace.

Výhody

- Snadná integrace s Flutterem pro vývoj multiplatformních aplikací,
- Podpora statického typování a objektově orientovaného programování, což zvyšuje bezpečnost a přehlednost kódu,
- Možnost kompilace do nativního kódu pro vysoký výkon aplikací.

Nevýhody

- Menší komunita a méně dostupných knihoven než u populárnějších jazyků, například JavaScriptu,
- Relativně méně zkušených vývojářů, což může komplikovat hledání odborné podpory,
- Méně rozšířený mimo Flutter, takže se hodí spíše pro specifické projekty než pro obecný vývoj softwaru.

1.3.2 Použití Dart ve Flutteru

Dart je primárním programovacím jazykem frameworku Flutter a je klíčový pro tvorbu aplikací v tomto prostředí. Ve Flutteru slouží Dart k definici widgetů, což jsou základní stavební bloky uživatelského rozhraní. Každý vizuální prvek aplikace, jako jsou tlačítka, texty nebo obrázky, je reprezentován widgetem, jehož chování a vzhled je definován v Dartu.

Dart rovněž umožňuje efektivní práci s asynchronními úlohami, což je při vývoji mobilních aplikací důležité například při načítání dat z databáze nebo z internetu. Díky hot reload ve Flutteru lze změny v Dart kódu okamžitě zobrazit v aplikaci, což výrazně urychluje vývoj a testování.

Použití Dartu ve Flutteru přináší jednotnou a přehlednou strukturu kódu, snadnější údržbu a možnost vytvářet multiplatformní aplikace pro Android i iOS z jedné kódové základny. Jazyk také podporuje moderní programovací principy, jako je objektově orientované programování, statické typování a modularita, které usnadňují tvorbu složitějších a dobře udržovatelných aplikací.

2 FUNKCE MOBILNÍ APLIKACE FRETFLY

2.1 METRONOM

Metronom je nástroj pro udržování konstantního tempa při cvičení na kytaru. Pomáhá držet správné tempo, a zároveň je to skvělý pomocník pro trénink rytmu. Při vytváření jsem měl trošku problém s implementací zvukového "tiku", který měl znázorňovat jednu dobu. Nakonec ale metronom funguje vcelku slušně, a ikdyž to není ještě úplně stoprocentní, svůj účel držení tempa splňuje.

2.1.1 Účel a funkce metronomu

Metronom v aplikaci **FretFly** poskytuje následující funkce:

- **Nastavení tempa v rozsahu 40–240 BPM**
  Uživatel si může zvolit tempo podle náročnosti cvičení, od pomalého procvičování až po rychlá rytmická cvičení.
- **Výběr taktu (2/4, 3/4, 4/4, 6/4)**
  Metronom podporuje více taktových označení, což umožňuje přizpůsobení různým hudebním stylům a skladbám.
- **Zvukový výstup s odlišením první doby**
  První doba v taktu je zvýrazněna odlišným zvukem, což usnadňuje orientaci v rytmu.
- **Vizuální indikátor aktuální doby**
  Grafický prvek zobrazuje průběh taktu a aktuální dobu, což je užitečné při cvičení bez zvuku nebo v hlučném prostředí.
- **Tap tempo pro nastavení BPM poklepáním**
  Uživatel může nastavit tempo opakovaným poklepáním na obrazovku podle rytmu konkrétní skladby.

2.1.2 Technická implementace

Metronom je v aplikaci **FretFly** implementován jako `StatefulWidget` s názvem `MetronomePage`, který spravuje svůj stav pomocí několika klíčových proměnných. Tyto proměnné určují chování metronomu, jeho aktuální nastavení a průběh přehrávání.

- **_isPlaying**
  Logická proměnná indikující, zda je metronom aktuálně spuštěn, nebo zastaven.
- **_bpm**
  Proměnná uchovávající aktuální tempo metronomu v úderech za minutu (BPM), přičemž podporovaný rozsah je 40 až 240 BPM.
- **_beatsPerBar**
  Určuje počet dob v jednom taktu. Hodnoty odpovídají podporovaným taktům (2, 3, 4 nebo 6).
- **_currentBeat**
  Sleduje aktuální dobu v rámci taktu, přičemž nabývá hodnot od 1 do hodnoty `_beatsPerBar`.
- **_timer**
  Periodický časovač, který zajišťuje pravidelné spouštění jednotlivých dob metronomu na základě nastaveného tempa.

2.1.3 Výpočet intervalu mezi dobami

Interval mezi jednotlivými dobami metronomu je odvozen z aktuálně nastaveného tempa, které je udáváno v jednotkách BPM (beats per minute). Jelikož jedna minuta odpovídá 60 000 milisekundám, lze interval mezi jednotlivými dobami vypočítat podle následujícího vzorce:

$$\text{interval} = \frac{60000}{\text{BPM}} \text{ ms}$$

Například při nastaveném tempu 120 BPM vychází interval mezi jednotlivými dobami na 500 ms, tedy 0,5 sekundy. Tento interval určuje, jak často je spuštěna další doba metronomu.

V aplikaci je výpočet intervalu a spuštění metronomu realizováno v metodě `_startMetronome()`, která nejprve okamžitě přehraje první dobu a následně nastaví periodický časovač pro další doby. Implementace této metody je uvedena níže:

```
1 void _startMetronome() {
2   _playBeat();
3
```

```
4    final interval = Duration(
5      milliseconds: (60000 / _bpm).round(),
6    );
7
8    _timer = Timer.periodic(interval, (timer) {
9      _playBeat();
10   });
11 }
```

Kód 2.1: Ukázka kódu pro spuštění metronomu.

Metoda `_startMetronome()` nejprve zavolá funkci `_playBeat()`, čímž zajistí okamžité přehrání první doby bez zpoždění. Následně je vypočítán časový interval na základě aktuální hodnoty BPM a vytvořen objekt třídy `Duration`. Tento interval je následně použit při vytvoření periodického časovače `Timer.periodic`, který v pravidelných časových intervalech spouští další doby metronomu.

Tímto způsobem je zajištěno přesné a stabilní časování jednotlivých úderů metronomu po celou dobu jeho běhu.

2.1.4 Zvukový výstup

Zvukový výstup metronomu je realizován pomocí balíčku `just_audio`, který umožňuje přesné a nízkolatenční přehrávání zvukových souborů. Pro zajištění jasného rozlišení první doby v taktu jsou použity dvě samostatné instance třídy `AudioPlayer`.

- **_accentPlayer**
  Slouží k přehrávání první doby v taktu (akcentované doby).
- **_regularPlayer**
  Používá se pro přehrávání ostatních, neakcentovaných dob.

První doba v taktu je zvukově i hapticky zvýrazněna, aby byla pro uživatele snadno rozpoznatelná. Toto zvýraznění je realizováno kombinací změny parametrů přehrávání a silnější haptické odezvy:

- přehrávání se zvýšenou rychlostí (`speed = 2.0`), čímž dojde ke zvýšení výšky tónu,
- přehrávání s plnou hlasitostí (`volume = 1.0`),
- silnější haptická odezva pomocí metody `HapticFeedback.mediumImpact()`.

Ostatní doby jsou přehrávány s méně výrazným zvukovým i haptickým projevem, aby bylo zachováno jasné rozlišení akcentované doby:

- přehrávání při standardní rychlosti (`speed = 1.0`),
- snížená hlasitost (`volume = 0.7`),
- lehká haptická odezva pomocí metody `HapticFeedback.lightImpact()`.

Zvukový soubor metronomu je uložen v adresáři `assets/sounds/sound-effect-hd.mp3` a je načten při inicializaci widgetu `MetronomePage`. Tím je zajištěno, že zvuk je připraven k okamžitému přehrání bez zbytečného zpoždění během běhu metronomu.

Použití oddělených přehrávačů pro akcentované a neakcentované doby umožňuje nezávislé nastavení parametrů přehrávání a přispívá k lepší čitelnosti rytmu a celkové uživatelské zkušenosti při cvičení.

2.1.5 Tap Tempo funkce

Funkce Tap Tempo umožňuje uživateli nastavit tempo metronomu (BPM) manuálně poklepáním na tlačítko v požadovaném rytmu. Tato metoda je vhodná zejména v situacích, kdy uživatel nezná přesnou hodnotu BPM, ale dokáže tempo intuitivně zadat.

Algoritmus Tap Tempo pracuje s časovými značkami jednotlivých poklepů a skládá se z následujících kroků:

1. **Ukládání časových značek**
   Při každém poklepu je aktuální čas uložen do seznamu `_tapTimes`. Algoritmus uchovává maximálně čtyři poslední poklepy, přičemž starší hodnoty jsou postupně odstraňovány.
2. **Výpočet intervalů mezi poklepy**
   Z uložených časových značek jsou vypočítány časové rozdíly mezi jednotlivými poklepy v milisekundách.
3. **Výpočet průměrného intervalu**
   Z vypočtených intervalů je určen průměrný interval, který slouží ke stabilizaci výsledného tempa a omezení vlivu nepřesného poklepu.
4. **Přepočet na BPM**
   Průměrný interval je následně převeden na hodnotu BPM podle vztahu:

$$BPM = \frac{60000}{\text{průměrný interval v ms}}$$

5. **Omezení výsledné hodnoty**
   Vypočtená hodnota BPM je omezena na povolený rozsah 40–240 BPM, aby odpovídala funkčním limitům metronomu.

Pro zajištění správné funkce je implementováno také časové omezení. Pokud uživatel neprovede další poklep po dobu delší než dvě sekundy, seznam časových značek `_tapTimes` je automaticky vymazán a nový výpočet začíná od začátku.

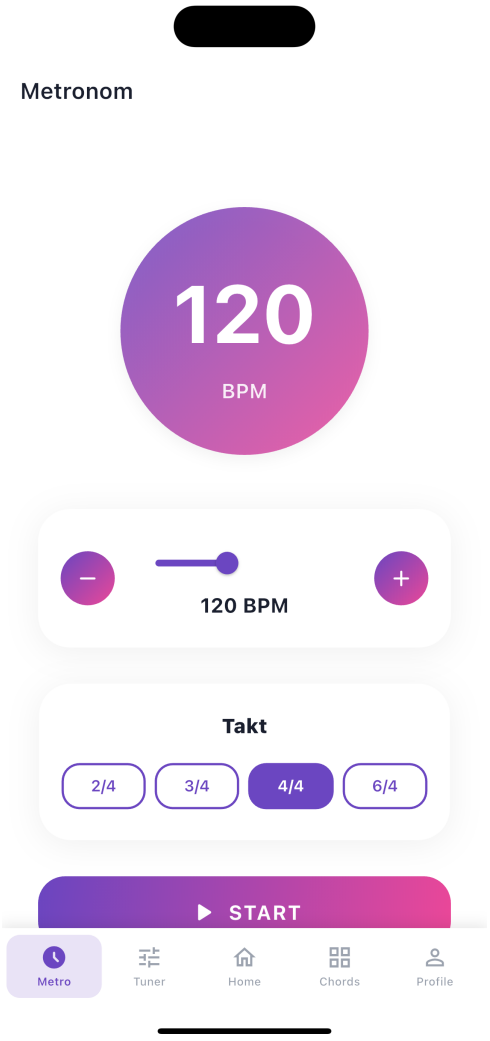
Vlastní implementace Tap Tempo je realizována metodou `_tapTempo()`, jejíž zdrojový kód je uveden níže:

```
void _tapTempo() {
  final now = DateTime.now();
  _tapTimes.add(now);

  if (_tapTimes.length > 4) {
    _tapTimes.removeAt(0);
  }

  if (_tapTimes.length >= 2) {
    final intervals = <int>[];

    for (int i = 1; i < _tapTimes.length; i++) {
      intervals.add(
        _tapTimes[i]
            .difference(_tapTimes[i - 1])
            .inMilliseconds,
      );
    }

    final avgInterval =
        intervals.reduce((a, b) => a + b) / intervals.length;

    final newBpm =
        (60000 / avgInterval).round().clamp(40, 240);

    setState(() {
      _bpm = newBpm;

      if (_isPlaying) {
        _stopMetronome();
```

```
31            _startMetronome();
32          }
33        });
34      }
35  }
```

Kód 2.2: Ukázka kódu pro funkci Tap Tempo.

Po změně hodnoty BPM je metronom v případě aktivního přehrávání automaticky restartován, aby se nové tempo okamžitě projevilo v běhu aplikace.



Obrázek 2.1: Ukázka rozhraní Metronomu.

2.2 LADIČKA

Ladička je nástroj, který dokáže přesně naladit každou strunu bez větší námahy. Na trhu už jsou nějakou dobu klasické ladičky - malá plastová zařízení s analogovým, ale už i s digitálním displejem, které se na kytaru fyzicky připnou a naladí ji. Myslím si ale, že v dnešní době už drtivá většina muzikantů (minimálně těch amatérských) zvolí levnější variantu a to ladičku v mobilu. Já sám jsem nikdy fyzickou ladičku nevlastnil a neměl jsem s mobilní verzí sebemenší problém. Proto v mé aplikaci nemohla chybět.

2.2.1 Standardní ladění kytary

Standardní ladění kytary je označováno jako **E-A-D-G-B-E** a představuje nejběžnější způsob ladění pro šestistrunnou kytaru. Každá struna má přesně definovanou frekvenci, která slouží jako cílový tón při ladění:

Struna	Pořadí	Frekvence [Hz]
E6	6 (nejtlustší)	82,41
A5	5	110,00
D4	4	146,83
G3	3	196,00
B2	2	246,94
E1	1 (nejtenčí)	329,63

Tyto frekvence jsou v aplikaci uloženy v konstantě _stringFrequencies a slouží pro porovnání s detekovanou frekvencí během ladění:

```
static const Map<String, double> _stringFrequencies = {
  'E6': 82.41,
  'A5': 110.00,
  'D4': 146.83,
  'G3': 196.00,
  'B2': 246.94,
  'E1': 329.63
};
```

Digitální ladička porovnává detekovaný tón s touto referenční hodnotou a poskytuje okamžitou vizuální zpětnou vazbu uživateli, což umožňuje přesné a rychlé ladění.

2.2.2 Technická implementace

Ladička v aplikaci je implementována jako interaktivní stránka (*StatefulWidget* TunerPage), která spravuje stav nahrávání a analýzy zvuku. Hlavními prvky jsou:

- **AudioRecorder** – nástroj pro záznam zvuku z mikrofonu.
- **Indikátor nahrávání** – ukazuje, zda aplikace právě nahrává.
- **Detekovaný tón a frekvence** – aktuálně zaznamenaný tón (E, A, D, G, B) a jeho frekvence v Hz.
- **Odchylka od cílového tónu** – vyjádřená v centech, ukazuje, o kolik je struna mimo ladění.
- **Vybraná struna** – možnost zaměřit se na konkrétní strunu.

Ukázka implementace

```
// Inicializace hlavních proměnných
final AudioRecorder _recorder = AudioRecorder();
bool _isRecording = false;
String? _detectedNote;
double _detectedFrequency = 0.0;
double _deviation = 0.0; // odchylka v centech
String? _selectedString;
Timer? _analysisTimer;

// Technické parametry zpracování zvuku
static const int sampleRate = 44100;
static const int bufferSize = 4096;
```

Kód 2.3: Ukázka kódu inicializace ladičky a jejích parametrů.

Tento kód ukazuje, jak aplikace uchovává informace o nahrávání a analyzovaném zvuku. Každých 200 ms probíhá analýza nahrávky, která zjišťuje aktuální frekvenci a odchylku od cílového tónu. Díky tomu uživatel okamžitě vidí, zda je struna naladěna správně, nebo zda je potřeba ji napnout či uvolnit.

Technické parametry

- **Vzorkovací frekvence:** 44100 Hz.

- **Velikost bufferu:** 4096 vzorků.
- **Formát:** PCM16 (16-bit signed integer).
- **Interval analýzy:** každých 200 ms.

Tyto parametry zajišťují přesné a rychlé zachycení frekvence každé struny a umožňují okamžitou vizuální zpětnou vazbu uživateli.

2.2.3 Záznam zvuku a detekce frekvence

Záznam zvuku v aplikaci probíhá pomocí třídy `AudioRecorder`, která zachycuje signál z mikrofonu. Nahrávání je nastaveno s parametry:

- **Formát:** PCM16 (16-bit signed integer).
- **Vzorkovací frekvence:** 44100 Hz.
- **Kanály:** mono (jediný kanál).

Periodická analýza probíhá každých 200 ms, aby aplikace mohla průběžně detekovat aktuální tón. Ukázka inicializace nahrávání v kódu:

```
await _recorder.start(
  const RecordConfig(
    encoder: AudioEncoder.pcm16bits,
    sampleRate: sampleRate,
    numChannels: 1,
  ),
  path: _recordingPath!,
);

// Periodická analýza zvuku každých 200 ms
_analysisTimer = Timer.periodic(
  const Duration(milliseconds: 200),
  (timer) {
    _analyzeAudioFile();
  },
);
```

Kód 2.4: Ukázka kódu spuštění záznamu zvuku.

Detekce frekvence využívá princip **zero-crossing detection**, což je metoda, která počítá, kolikrát signál prochází nulovou hodnotou. Z tohoto počtu a délky nahrávky lze vypočítat frekvenci tónu. Přehled hlavních kroků:

- Normalizace vzorků – odstranění posunu signálu (DC offset).
- Počítání průchodů nulou s filtrováním šumu (threshold).
- Výpočet frekvence: počet průchodů nulou dělený délkou vzorku.
- Filtrování nereálných frekvencí – pro kytaru platí rozsah 50–500 Hz.

Ukázka funkce pro detekci frekvence:

```dart
double _detectFrequency(List<int> samples) {
  final normalized = _normalizeSamples(samples);
  const threshold = 100;
  int zeroCrossings = 0;
  bool wasPositive = normalized[0] > 0;

  for (int i = 1; i < normalized.length; i++) {
    final current = normalized[i];
    if (wasPositive && current < -threshold) {
      zeroCrossings++;
      wasPositive = false;
    } else if (!wasPositive && current > threshold) {
      zeroCrossings++;
      wasPositive = true;
    }
  }

  final duration = samples.length / sampleRate;
  final frequency = (zeroCrossings / 2) / duration;

  if (frequency < 50 || frequency > 500) {
    return 0.0;
  }

  return frequency;
}
```

Kód 2.5: Ukázka kódu detekce frekvence metodou průchodu nulou.

Tento princip umožňuje aplikaci rozpoznat aktuálně hraný tón a jeho frekvenci, což je základ pro následné vyhodnocení odchylky a vizuální indikaci ladění.

2.2.4 Výpočet odchylky a vizuální indikátory

Po detekci frekvence hraného tónu aplikace vypočítá odchylku od cílového tónu v **centech** (1 cent = 1/100 půltónu). Tento způsob měření umožňuje přesně vyjádřit, jak moc je tón nad nebo pod ideálním laděním.

```
double _calculateDeviation(double detectedFreq, double targetFreq) {
  if (detectedFreq <= 0 || targetFreq <= 0) {
    return 0.0;
  }

  final cents =
      1200 * (log(detectedFreq / targetFreq) / log(2));

  return cents;
}
```

Kód 2.6: Ukázka kódu pro výpočet odchylky tónu v centech.

Matematicky lze odchylku vyjádřit vzorcem:

$$\text{cents} = 1200 \times \frac{\log(f_{\text{detekovaná}}/f_{\text{cílová}})}{\log(2)}$$

Pro intuitivní vizuální zpětnou vazbu aplikace používá barevné indikátory:

- **Zelená:** odchylka menší než ±5 centů – tón je perfektně naladěn.
- **Oranžová:** odchylka mezi ±5 a ±20 centy – tón je blízko ideálu.
- **Červená:** odchylka větší než ±20 centů – tón je výrazně mimo ladění.

Aplikace zobrazuje:

- Detekovaný tón a jeho frekvenci v Hz
- Vizuální indikátor ladění s posuvníkem
- Barevný indikátor odchylky (zelená, oranžová, červená)
- Textové instrukce pro ladění – napnout nebo uvolnit strunu

Tento přístup zajišťuje, že uživatel okamžitě vidí, zda je struna správně naladěna, a dostává jasné instrukce, jak postupovat.

Obrázek 2.2: Ukázka rozhraní Ladičky.

2.3 Databáze akordů

Myslím si, že trošku méně atraktivnější část aplikace, ale rozhodně nezbytná. Mít akordy na jednom uspořádaném místě je určitě příjemné. Proto jsem chtěl jednu takovou alespoň s pár akordy i v mé aplikaci. Na začátku jsem plánoval, že udělám databázi lokální, ale jelikož jsem se rozhodl použít Firebase pro ukládání přihlášených uživatelů, použil jsem Firebase i pro vytvoření modelu pro databázi akordů.

2.3.1 Model akordu

Akord je v aplikaci reprezentován třídou `Chord`, která obsahuje následující vlastnosti:

```
class Chord {
  final String id;
  final String name;
  final String category;
  final List<int> fingering;   // [0, 3, 2, 0, 1, 0] pro struny E-A-D-G-B-E
  final int position;          // pozice na hmatníku (1 = první pražec)
  final bool hasBarre;
  final int? barreFret;
  final String? description;
  final List<String> tags;
  final String difficulty;     // "beginner", "intermediate", "advanced"
}
```

Kód 2.7: Ukázka třídy pro reprezentaci akordu.

Význam jednotlivých vlastností:

- **id**: unikátní identifikátor akordu.
- **name**: název akordu (např. C, G7, Am).
- **category**: kategorie akordu (např. dur, moll, sedmý).
- **fingering**: pole pozic prstů pro jednotlivé struny (E-A-D-G-B-E):
  - `0` = otevřená struna
  - `-1` = neznělá struna
  - `1-24` = pozice prstu na pražci
- **position**: pozice akordu na hmatníku (1 = první pražec).
- **hasBarre**: zda akord obsahuje barré.
- **barreFret**: pokud je barré, na kterém pražci.(Barré je kytarová technika, ve které ukazovák stiskne více strun najednou)
- **description**: volitelný text s vysvětlením akordu.
- **tags**: seznam tagů pro vyhledávání nebo filtrování akordů.
- **difficulty**: obtížnost akordu – `"beginner"`, `"intermediate"` nebo `"advanced"`.

2.3.2 ChordsService

ChordsService je specializovaná služba poskytující jednotný přístup k databázi akordů umístěné v *Cloud Firestore*. Tato služba centralizuje operace s akordy, což umožňuje snadné načítání, vyhledávání a filtrování akordů napříč aplikací. Použití služby zajišťuje, že všechny komponenty aplikace pracují s konzistentními daty a snižuje duplicitu kódu.

Hlavní funkce *ChordsService* zahrnují:

- Načtení kompletního seznamu akordů.
- Vyhledávání akordů podle názvu.
- Filtrování akordů podle kategorie (např. mollové, durové).
- Filtrování akordů podle obtížnosti (začátečník, středně pokročilý, pokročilý).

```
class ChordsService {
  static final ChordsService _instance = ChordsService._internal();
  factory ChordsService() => _instance;

  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  static const String _collection = 'chords';

  // Načte všechny akordy z kolekce
  Stream<List<Chord>> getAllChords() {
    return _firestore
        .collection(_collection)
        .orderBy('name')
        .snapshots()
        .map((snapshot) => snapshot.docs
            .map((doc) => Chord.fromMap(doc.data(), doc.id))
            .toList());
  }

  // Vyhledávání akordů podle názvu
  Stream<List<Chord>> searchChords(String query) {
    if (query.isEmpty) return getAllChords();

    return _firestore
        .collection(_collection)
```

```
        .where('name', isGreaterThanOrEqualTo: query)
        .where('name', isLessThan: '${query}z')
        .snapshots()
        .map((snapshot) => snapshot.docs
            .map((doc) => Chord.fromMap(doc.data(), doc.id))
            .toList());
  }
}
```

Kód 2.8: Ukázka implementace služby ChordsService.

2.3.3 Inicializace databáze

Při prvním spuštění aplikace je nezbytné inicializovat databázi akordů s předdefinovanou sadou základních akordů. Tento proces zajistí, že aplikace má ihned k dispozici všechny potřebné akordy, které mohou být dále načítány a filtrovány pomocí služby *ChordsService*.

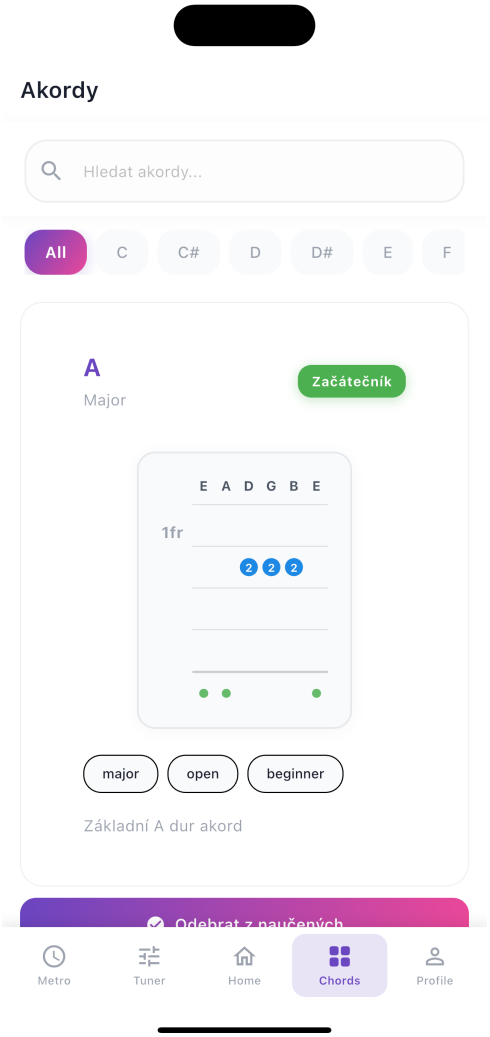
Databáze obsahuje akordy z kategorií *Major*, *Minor* a *Seventh* s různými úrovněmi obtížnosti, což umožňuje uživatelům od začátečníků po pokročilé pohodlné používání aplikace.

```
Future<void> initializeChords() async {
  final chords = [
    Chord(
      id: 'c_major',
      name: 'C',
      category: 'Major',
      fingering: [0, 1, 0, 2, 1, 0],
      position: 1,
      tags: ['major', 'open', 'beginner'],
      difficulty: 'beginner',
    ),
    // ... další akordy
  ];

  for (final chord in chords) {
    final docRef = _firestore.collection(_collection).doc(chord.id);
    final existing = await docRef.get();
    if (!existing.exists) {
      await docRef.set(chord.toMap());
    }
```

```
21    }
22 }
```

Kód 2.9: Ukázka inicializace databáze akordů.

Poznámka: Tento postup zajišťuje, že každý akord je v databázi pouze jednou a nedochází k duplicitám. Inicializace podporuje snadnou rozšiřitelnost – nové akordy mohou být přidány do seznamu a automaticky synchronizovány s Firestore.



Obrázek 2.3: Ukázka rozhraní databáze s akordy.

3 PŘIHLAŠOVACÍ SYSTÉM

Jak už jsem zmínil u kapitoly 2.3 Databáze akordů, pro ukládání uživatelů jsem použil Firebase Authentication. Původně jsem zamýšlel, že zprovozním i přihlášení přes Google a Apple, nakonec jsem od toho hlavně kvůli nedostatku času upustil, a tak je v aplikaci pouze přihlášení přes e-mail. Do budoucna bych však přihlášení přes Google a Apple moc rád do aplikace přidal.

3.1 AUTHSERVICE

AuthService je singletonová služba, která centralizuje veškeré operace související s autentizací uživatelů v aplikaci. Zajišťuje bezpečné přihlášení, registraci a správu přihlášeného uživatele.

Služba obsahuje následující funkcionality:

- Přihlášení pomocí e-mailu a hesla
- Registrace nového uživatele
- Odhlášení uživatele

```
class AuthService {
  AuthService._();
  static final AuthService instance = AuthService._();

  Future<void> signIn({
    required String email,
    required String password
  }) async {
    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: email.trim(),
        password: password,
      );

      await StreakService().updateOnLogin();
```

```
16
17        final p = await _prefs;
18        await p.setString(
19          _keyCurrentUser,
20          FirebaseAuth.instance.currentUser?.email ?? '',
21        );
22      } on FirebaseAuthException catch (e) {
23        throw AuthException(_mapFirebaseError(e));
24      }
25    }
26  }
```

Kód 3.1: Ukázka implementace přihlášení přes e-mail a heslo.

Poznámka: Centralizace autentizace v *AuthService* zajišťuje jednotnou logiku přihlášení a snadnou správu chyb napříč celou aplikací.

3.2 LoginPage

LoginPage poskytuje uživateli rozhraní pro přihlášení do aplikace s validací formuláře. Zajišťuje bezpečný vstup uživatelských údajů a zobrazení chybových hlášení.

Stránka obsahuje:

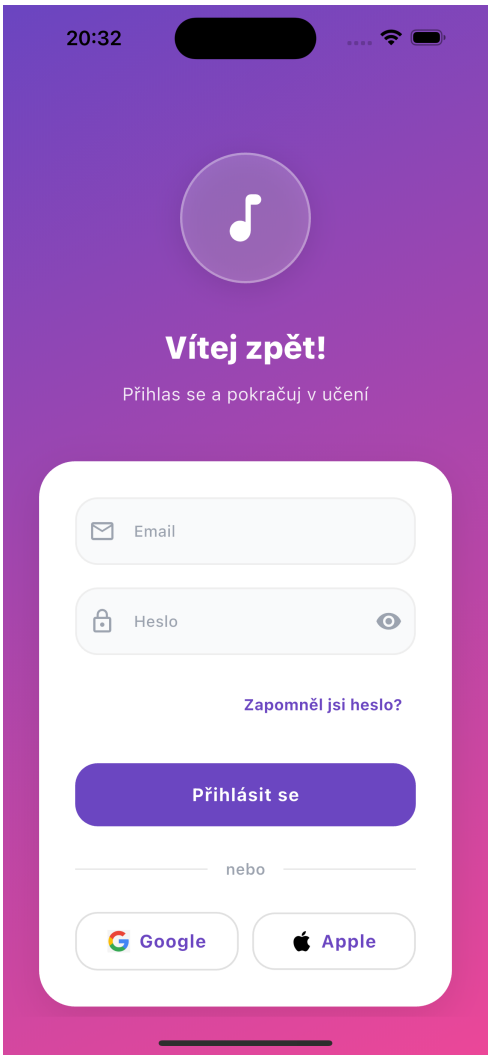
- Formulář s validací e-mailu a hesla
- Odkaz na registraci
- Zobrazení chybových zpráv

```
1  Future<void> _onSubmit() async {
2    if (!_formKey.currentState!.validate()) return;
3    setState(() => _isSubmitting = true);
4
5    try {
6      final email = _emailController.text.trim();
7      final password = _passwordController.text;
8      await AuthService.instance.signIn(email: email, password: password);
9
10     if (!mounted) return;
11     setState(() => _isSubmitting = false);
```

```
12      _navigateToHome();
13    } on AuthException catch (e) {
14      if (!mounted) return;
15      setState(() => _isSubmitting = false);
16      ScaffoldMessenger.of(context).showSnackBar(
17        SnackBar(content: Text(e.message))
18      );
19    }
20  }
```

Kód 3.2: Ukázka zpracování přihlášení uživatele.

Poznámka: Validace a ošetření chyb zajišťují, že uživatel získá okamžitou zpětnou vazbu při nesprávném zadání údajů.



Obrázek 3.1: Ukázka rozhraní LoginPage.

3.3 SIGNUPPAGE

SignUpPage poskytuje uživateli rozhraní pro registraci nového účtu. Formulář zajišťuje validaci vstupních údajů a ošetření chyb.

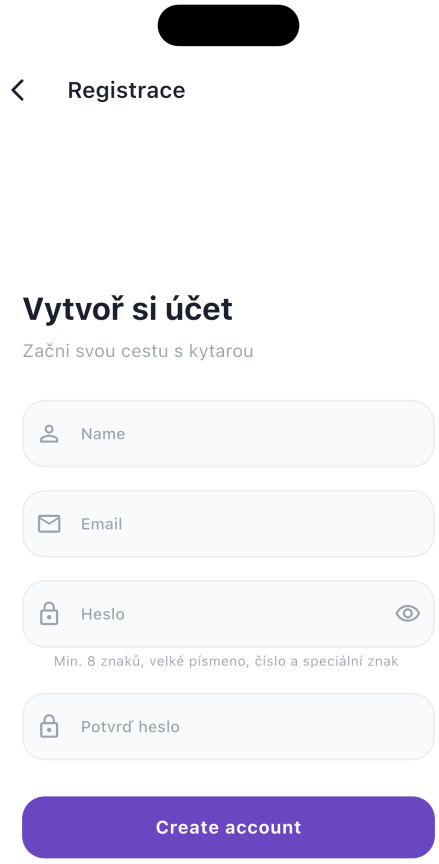
Stránka obsahuje:

- Formulář s validací:
  - Jméno (povinné)
  - E-mail (správný formát)
  - Heslo (minimální délka)
  - Potvrzení hesla (shoda s heslem)
- Zobrazení chybových zpráv

```
Future<void> _onSubmit() async {
  if (!_formKey.currentState!.validate()) return;

  try {
    await AuthService.instance.signUp(
      name: _nameController.text,
      email: _emailController.text,
      password: _passwordController.text,
    );
    Navigator.of(context).pop();
  } on AuthException catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(e.message))
    );
  }
}
```

Kód 3.3: Ukázka zpracování registrace nového uživatele.

Poznámka: Validace a ošetření chyb zajišťují bezpečný a přehledný proces registrace.



Obrázek 3.2: Ukázka rozhraní SignUpPage.

ZÁVĚR

Cílem projektu bylo vytvořit mobilní aplikaci, která bude obsahovat základní nástroje pro kytaristy. Aplikace je postavená na frameworku Flutter, a je celý napsaný v programovacím jazyku Dart. Databáze je řešena přes Firestore Database, a autentizace přes Firebase Authentication. Od začátku jsem se snažil aplikaci navrhnout tak, aby byla co nejvíce jednoduchá, a srozumitelná. Výsledkem je aplikace, která nabízí plně funkční metronom, ladičku, a přehlednou databázi akordů.

Odkaz na projekt: `https://github.com/JakubHalama/Fretfly-Maturitni_projekt`

SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ

[1] *Flutter — Documentation* [online]. Oficiální dokumentace frameworku Flutter. [cit. 2025-01-05]. Dostupné z: `https://docs.flutter.dev`

[2] *Flutter Tutorial — Build apps for any screen* [online]. Oficiální výukové materiály Flutteru. [cit. 2025-01-05]. Dostupné z: `https://docs.flutter.dev/learn`

[3] *Dart — Language Documentation* [online]. Oficiální dokumentace programovacího jazyka Dart. [cit. 2025-01-05]. Dostupné z: `https://dart.dev/guides`

[4] *Firebase for Flutter — Documentation* [online]. Dokumentace k propojení Flutter aplikací se službami Firebase. [cit. 2025-01-05]. Dostupné z: `https://firebase.google.com/docs/flutter`

[5] *Add Firebase to your Flutter app* [online]. Návod na integraci Firebase do Flutter projektu pomocí FlutterFire CLI. [cit. 2025-01-05]. Dostupné z: `https://firebase.google.com/docs/flutter/setup`

[6] *Get to know Firebase for Flutter — Codelab* [online]. Praktický kurz práce s autentizací a databází Firestore ve Flutter aplikaci. [cit. 2025-01-05]. Dostupné z: `https://firebase.google.com/codelabs/firebase-get-to-know-flutter`

[7] *Building layouts in Flutter* [online]. Návod na tvorbu uživatelského rozhraní pomocí widgetů. [cit. 2025-01-05]. Dostupné z: `https://docs.flutter.dev/ui/layout`

[8] *State management — Flutter Documentation* [online]. Přehled přístupů ke správě stavu aplikace ve Flutteru. [cit. 2025-01-05]. Dostupné z: `https://docs.flutter.dev/data-and-backend/state-mgmt`

[9] *Testing Flutter apps* [online]. Oficiální dokumentace testování Flutter aplikací. [cit. 2025-01-05]. Dostupné z: `https://docs.flutter.dev/testing`

[10] *Flutter and Firebase App Build — Full Tutorial* [online]. YouTube, 12.6.2022 [cit. 2025-01-05]. Dostupné z: `https://www.youtube.com/watch?v=1gDhl4leEzA`

[11] *Flutter Firebase Authentication Tutorial* [online]. YouTube, 18.3.2021 [cit. 2025-01-05]. Dostupné z: `https://www.youtube.com/watch?v=7gqCzGgQG0U`

[12] *Flutter Firestore CRUD Tutorial* [online]. YouTube, 5.9.2021 [cit. 2025-01-05]. Dostupné z: https://www.youtube.com/watch?v=DqJ_KjFzL9I

Seznam obrázků

SEZNAM PROGRAMOVÝCH KÓDŮ