Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Mobilní aplikace pomáhající při hře na kytaru

Šimon Smrček

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Šimon Smrček

Systémové inženýrství a informatika
Informatika

Název práce

Mobilní aplikace pomáhající při hře na kytaru

Název anglicky

Mobile app helping with playing guitar

Cíle práce

Práce se zabývá tématikou vývoje mobilních aplikací. Cílem teoretické části práce je charakteristika vývoje aplikací na mobilní operační systém Android, nových trendů v programování mobilních aplikací, programovacího jazyku Kotlin, Kotlin Coroutines a současného stavu vybraných aplikací zabývajících se pomocí při hře na hudební nástroje. V neposlední řadě je shrnuta teorie týkající se hry na kytaru a vytvoření softwarového návrhu.
Cílem praktické části práce je naprogramování multifunkční aplikace na mobilní operační systém Android pomáhající při hře na hudební nástroje včetně testování správne funkcionality aplikace. Následně proběhne porovnání s ostatními volně přístupnými hudebními aplikacemi.

Metodika

Metodika řešené problematiky diplomové práce je založena na studiu a analýze odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude popsána problematika týkající se programování aplikací na operační systém Android, vývoje v jazyce Kotlin a moderních technologií v programování na mobilní zařízení.

Aplikace bude naprogramována za pomoci moderních technologií a přístupů k vývoji mobilních aplikací. Na základně analýzy dostupných technologií (např. Kotlin Coroutines, Model-view-viewmode, Live data, Room database, ConstraintLayout, Android jetpack nebo Fireback) budou vybrány vhodné technologie, které budou použity v kódu. Kompletní aplikace bude testována z hlediska funkcionality jednotlivých částí i celkové. Na základě vytvořené aplikace budou měřeny její vlastnosti, které budou porovnány s vlastnostmi ostatních aplikací zabývajícími se pomocí při hře na hudební nástroje. Z aplikací budou vybrány jednotlivé části, které provádějí totožné aktivity, na kterých bude provedeno porovnání funkce a výkonnosti aplikace. K tomu budou vybrány objektivně měřitelná kritéria, jako je počet řádků nebo rychlost výpočtu procesu.

Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry diplomové práce.

Doporučený rozsah práce

60-80 stran

Klíčová slova

mobilní aplikace, Kotlin, Android, Kotlin Coroutines, RxKotlin, Android Jetpack, hra ta kytaru

---

Doporučené zdroje informací

ANNUZI, Joseph Jr. Advanced Android Application Development. 2014. Addison-Wesley Professional, 2014. ISBN 013389245X, 9780133892451.

EBEL, Nate. Mastering Kotlin: Learn advanced Kotlin programming techniques to build apps for Android, iOS, and the web. Birmingham: Packt Publishing, 2019. ISBN 1838552367.

FRENCH, Richard Mark. Technology of the Guitar. Springer Science & Business Media, 2012. ISBN 1461419212, 9781461419211.

KOUSEN, Ken. Kotlin Cookbook: A Problem-Focused Approach [online]. 2019. New York: O'Reilly Media, 2019 [cit. 2020-01-27]. ISBN 1492046639, 9781492046639.

SPÄTH, Peter. Pro Android with Kotlin: Developing Modern Mobile Apps. New York: Apress, 2018. ISBN 1484238206.

Předběžný termín obhajoby

2021/22 ZS – PEF

Vedoucí práce

Ing. Dana Vynikarová, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 06. 03. 2022

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Mobilní aplikace pomáhající při hře na kytaru" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 23.3.2021 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Daně Vynikarové, Ph.D. za vstřícný přístup, odborné rady, věcné připomínky a trpělivost. Dále bych rád také chtěl poděkovat Vojtovi Smrčkovi za pomoc s objasněním problematiky, Dominiku Mattovi za pomoc s nahráváním tónů pro aplikaci, Irině Popescu s pomocí s citacemi a Martinu Smrčkovi za korekturu textu. Velké díky patří rodině a přátelům, kteří mě podporovali v dlouhém průběhu práce, a nakonec mé největší opoře, která by na základě mých stížností a detailních popisů dokázala stejnou práci zpracovat také (nutno dodat, že pravděpodobně lépe).

Mobilní aplikace pomáhající při hře na kytaru

Abstrakt

Diplomová práce se zabývá problematikou spojenou s vývojem mobilních aplikaci sloužících k výuce hry na kytaru. Cílem práce je vytvořit originální multifunkční mobilní aplikaci na OS Android, která bude samostatně použitelná jako výukový prvek.

Teoretická část práce je rozdělena do pěti částí. První z nich je zaměřena na popis základů kytarové teorie a hry na kytaru. Druhá je zaměřena na popis životního cyklu vývoje mobilních aplikací. Třetí část slouží k popisu operačního systému Android a programování aplikací na tento OS. Čtvrtá část popisuje jednotlivé moderní technologie a přístupy k vývoji mobilních aplikací. Poslední, pátá, část popisuje analýzu trhu s hudebními mobilními aplikacemi.

Na základě teoretických poznatků je vytvořena praktická část, která je opět rozdělena do pěti částí, tentokrát podle jednotlivých částí životního cyklu vývoje mobilních aplikací. První z nich je počáteční část a věnuje se popisu problematiky a vytvoření uživatelských požadavků. Druhou je návrhová část, která slouží k vytvoření UX designu a modelů spojených s aplikací. Třetí je vývojová část, ve které je naprogramována samotná aplikace a popsán způsob vývoje a jednotlivé relevantní části kódu. Ve čtvrté části dochází k testování kvality aplikace. V poslední části, která se jmenuje nasazení/údržba, je provedeno zhodnocení výsledků a diskuse.

Zhodnocení výsledků a diskuse jsou provedeny v porovnání s ostatními obdobnými aplikacemi. V neposlední řadě jsou zmíněny nedostatky aplikace a změny, které by bylo v budoucnu možno provést.

Klíčová slova: mobilní aplikace, hra na kytaru, Kotlin, Android, Kotlin Coroutines, RxKotlin, Android Jetpack, Firebase

Mobile app helping with playing guitar

Abstract

This master's thesis is focused on the issue of mobile applications development for teaching how to play guitar. The objective is to develop an original multifunctional mobile application for OS Android, which will be useful as an independent teaching tool.

The theoretical part of the thesis is divided into five sections. The first section is focused on a description of guitar theory and playing guitar. The second part is focused on the description of the mobile application development lifecycle. The third part describes the operating system of Android and the programming connected to it. The fourth part is talking about modern technologies and approaches to mobile application development. The last, fifth, part is focused on the analysis of the music applications market.

On the theoretical bases set by the previous part was made a practical part, which is once again divided into 5 parts, this time split into parts matching the mobile application development lifecycle. The first part is called a Discovery phase and it's focused on problem description and requirements engineering. The second is the design part, which is creating a UX design and models connected to the app. The third part is the development part, and it includes the programming of the application and the description of relevant pieces of the code. The fourth part is based on testing the quality of the application. The last part, which is called launch/maintenance, serves for evaluating the results and discussion.

The evaluation of results and discussion is based on comparison to other similar applications. Last but not least are mentioned the imperfections of the application and changes, that would be possible to make in the future.

Keywords: mobile application, guitar playing, Kotlin, Android, Kotlin Coroutines, RxKotlin, Android Jetpack, Firebase

Obsah

1 Úvod

V dnešní době jsou chytrá mobilní zařízení nedílnou součástí lidského života. Pro jejich efektivní užívání jsou vyvíjeny aplikace, které mohou sloužit pro mnoho účelů – od firemních až po volnočasové aktivity. Jelikož klientské nároky jsou stále vyšší a častokrát přesahují možnosti některých vývojových cest, odvětví vývoje mobilních aplikací prochází neustálými změnami a vylepšováními. Uživatelům již nestačí pouze poskytnutí základní funkcionality, ale záleží na nefunkčních vlastnostech, jakými jsou například výkon, použitelnost nebo zabezpečení. I proto trh aplikací prochází neustálým vývojem a aplikace, které byly dostatečné v minulosti, už jako takové označitelné být nemohou.

Zaměření aplikací je široké a jen těžko by šlo najít odvětví, pro které ještě aplikace vyvinuta nebyla. Jedním z odvětví, které ještě není plně prozkoumané, je hudba. Ačkoliv aplikací souvisejících s hudbou je nepřeberné množství, stále se nachází na trhu prázdná místa. Jedním takovým místem jsou výukové aplikace. Jestliže má aplikace simulovat výukové hudební prostředí, je potřeba zajištění výkonu, se kterým mohou mít nemoderní aplikace problémy.

Na základě analýzy ostatních hudebních aplikací bude vymezen nepokrytý prostor funkcionality, na který bude vyvíjená aplikace zaměřena. Tato práce tedy bude sloužit v první řadě k vývoji moderní, unikátní a multifunkční hudební aplikace na základě vybraných metodologií. Aplikace bude efektivně zahrnovat moderní technologie a přístupy v programování pro mobilní aplikace. Následně bude porovnána funkcionalita aplikace vůči ostatním a bude zmíněn přínos aplikace.

V teoretické části práce budou popsány základy hry na kytaru a propojení hudby s chytrými telefony. Následně bude popsán životní cyklus vývoje mobilních aplikací, který poskytuje strukturu vývoje aplikace a její součásti a zároveň je v něm popsána metodologie vývoje. Poté bude popsána operační systém Android a programování s ním spojené. Dále budou popsány moderní technologie a přístupy k vývoji mobilních aplikací, jako je reaktivní programování, Kotlin, Android Jetpack, MVVM a Google Firebase. V poslední řadě bude provedena analýza trhu, která slouží k určení unikátních prvků budoucí aplikace.

Samotný vývoj hudební aplikace se bude skládat z jednotlivých fází životního cyklu vývoje mobilních aplikací. Nejprve bude provedena počáteční fáze, ve které bude provedena specifikace aplikace. V následné, návrhové, fázi bude vytvořena softwarová architektura

a prototyp aplikace. Ve vývojové fázi bude popsán kód aplikace a jednotlivé použité technologie. V testovací fázi bude provedeno testování kvality vytvořené aplikace. Dále bude provedeno zhodnocení výsledků a diskuze, kde budou popsány přínosy práce a možné pokračování v budoucnosti.

2 Cíl práce a metodika

2.1 Cíl práce

Práce se zabývá tématikou vývoje mobilních aplikací. Hlavním cílem je vyvinout multifunkční a originální mobilní aplikaci na mobilní operační systém Android v jazyce Kotlin, která slouží k osobní výuce hry na kytaru.

Vedlejšími cíli je charakteristika teorie hry na kytaru, životního cyklu vývoje mobilních aplikací, vývoje aplikací na operační systém Android, moderních programovacích technologií a přístupů a provedení analýzy trhu.

2.2 Metodika

Metodika řešené problematiky diplomové práce je založena na studiu a analýze odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude popsána problematiky týkající se vývoje aplikací na mobilní zařízení na operační systém Android, hry na kytaru a moderních technologií a přístupů k vývoji mobilních aplikací.

Vývoj aplikace bude proveden na základě sledování životního cyklu vývoje mobilních aplikací, pomocí něhož bude vytvořena aplikace zabývající se problematikou spojenou s výukou hry na kytaru.

Na základě analýzy trhu budou vybrány podobné aplikace a bude analyzována jejich funkcionalita. Na základě těchto funkcionalit budou nalezeny nedostatky odvětví, do kterých bude vsazena vytvořená aplikace. Následně bude aplikace porovnána s podobnými aplikacemi na trhu.

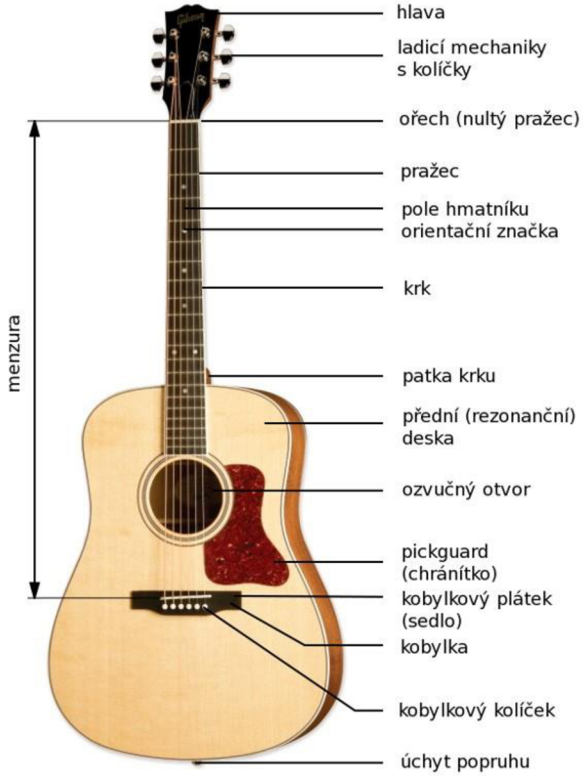
Po vytvoření aplikace budou změřeny objektivní parametry, které budou poskytnuty s výsledky práce.

Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry diplomové práce.

3 Teoretická východiska

3.1 Teorie hry na kytaru

Kytarová teorie vysvětluje jednotlivé hudební součásti a jejich funkci a význam v hudbě. Ačkoliv pro hru na kytaru není nutné hudební nebo kytarovou teorii znát, výrazně pomáhá k detailnímu pochopení kytary jako systému a k propojení jednotlivých součástí, které může vést k intuitivnějšímu hraní.



Obrázek 1 - Zobrazení kytary s popisem jednotlivých částí (kytara.net, 2010)

Na obrázku 1 lze vidět složení kytary. Pro samotné hraní jsou nejdůležitější části pražce, které určují tóninu strun.

K pochopení kytarové teorie je nutné znát některé základní termíny z hudební terminologie:

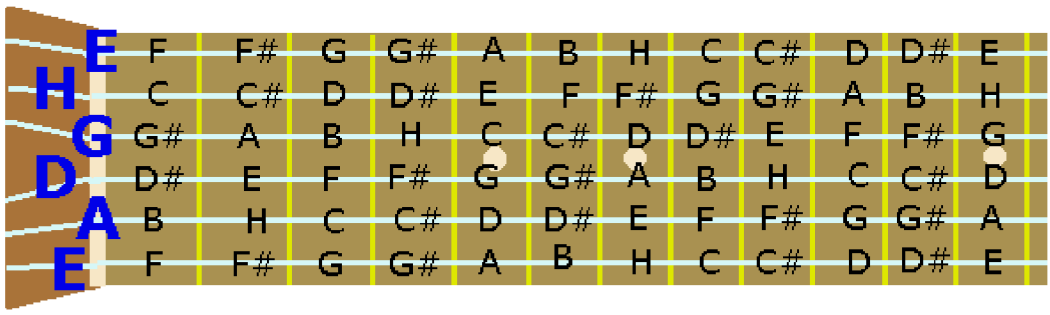
- Výška tónu – Výšku tónu lze definovat jako relativní výšku nebo hloubku zvuku. Jestliže je na kytaře zahrána prázdná struna, výsledkem je hlubší tón. Jestliže je zahrána struna na 10. pražci, jedná se o tón vyšší. Výška tónu je také označením přesnosti noty. Jestliže hraná nota není v přesné shodě s kytarovou

teorií (přesnou frekvencí tónu), jedná se o falešný tón, a tudíž nepřesnou výšku tónu.

- Rytmus – Rytmus je časová organizace zvuků. Některé rytmy mohou obsahovat výšku, ale to není pro rytmus důležité. Hudebníci hrají rytmus spojený s beatem (taktem), kterému se říká také tempo a vyjadřuje se v úderech za minutu (BPM). Rytmus je omezen pouze do daného tempa. Toto tempo je následně rozloženo do částí, které určují rytmus.
- Harmonie/Akordy – Harmonie je souzvuk několika výšek tónů. Pomocí kombinace tónů lze vytvořit zvuky představující různé nálady. Čím více tónu harmonie má, tím komplexnější je. Jestliže tři různé osoby zahrají noty C, E a G, zvukově vytvoří akord C dur.
- Melodie – Melodie je série zvuků tónů, která je obyčejně nad (výškou nebo intenzitou) zbytkem hudby. V popu je melodie především zpěv, kde tvoří harmonii alespoň s jedním dalším hudebním nástrojem. Z hlediska nezasvěceného diváka se jedná o nejdůležitější prvek, protože je unikátním identifikátorem písní.

Mezi další důležité hudební pojmy se řadí témbr neboli kvalita, díky němuž je tón jedinečný, a dále dynamika, což je síla nebo jemnost zvuku. (Hub Guitar, 2018)

Který tón se ozve, při kterém stisknutí struny na hmatníku na obrázku 2.



Obrázek 2 - Hmatník kytary s tóny(Dušan Janovský, 2001)

3.1.1 Propojení hudby a chytrých telefonů

Příchod chytrých telefonů v nedávné době významně přispěl k dalšímu nárůstu zájmu o hudbu. Zároveň také naprosto změnil způsob, kterým je hudba vnímána. Nejen díky technologickému pokroku (přechod z kupovaných nosičů na digitální formu přenosu

hudby), ale také přístupnosti hudby. Chytré telefony umožnili přístup nejen k hudbě z jakéhokoliv místa a v jakékoliv situaci. Aplikací, které jsou s hudbou spojeny, je pro chytré telefony všech OS velké množství a je jich vytvářeno čím dál více. K nejvýznamnějším patří například Spotify, SoundCloud, YouTube Music nebo Apple Music, což jsou poskytovatelé audio streamovacích a mediálních služeb a umožňují tak poslouchání hudby z jakéhokoliv místa. Mimo streamovací služby lze také najít aplikace, které jsou schopné rozpoznávat přehrávat, nahrávat nebo vytvářet hudbu. Jelikož nejrozšířenějším operačním systémem je Android, nejvíce aplikací (mimo multiplatformní) je vyvinuto právě pro něj. (similarweb.com, 2022)

Čistý průnik hudební (nebo kytarové teorie) s aplikacemi není tak prozkoumaný jako přehrávání hudby, nicméně na trhu se vyskytuje několik aplikací, které slouží především k základnímu zprostředkování kytarové teorie (učení a hraní akordů nebo tónů). Většinou nicméně chybí propojení s hlubšími znalostmi hudby i s reálným hraním na kytaru a propojení s teorií je spíše povrchové. Velice populární hudební pomůcky jsou například aplikace sloužící k ladění hudebních nástrojů nebo aplikace zprostředkovávajíc akordy.

3.2 Životní cyklus vývoje mobilních aplikací

Vývoj mobilních aplikace může být velice snadný. Může se skládat pouze z otevření programovací prostředí, naprogramování kódu, rychlého otestování základní funkcionality a publikování v jednom z online obchodů. Na druhou stranu se může také jednat o komplikovaný a náročný proces, který obsahuje přísný design před samotným programováním, plné testování funkcionality, testování na tisíce zařízeních, beta verzi a nasazení v několika formách. (David Ortinau *et al.*, 2021)

Dle dostupných informací se ukazuje, že kvalita (a tudíž i její použitelnost) aplikace nejvíce upadá kvůli nedodržování metodologie vývojového cyklu vývoje aplikací. Většina uživatelů začíná vyvíjet mobilní aplikaci bez toho, aniž by životní cyklus brala v potaz a tím pádem zanedbává důležité součásti, jako jsou například sběr požadavků nebo základní designování. Při vývoji mobilních aplikací, ve větší míře než při vývoji ostatních aplikací, je nutné brát v potaz problémy s vývojem spojené. Tyto problémy často přímo souvisí s dodržováním vývojového cyklu. (N Inukollu *et al.*, 2014)

Životní cyklus vývoje aplikací/software popisuje proces vývoje aplikace/software. Životní cyklus vývoje mobilních aplikací nepřináší oproti životnímu cyklu vývoje web nebo desktop aplikací žádnou velkou změnu. Jedná se o aplikaci standardních business praktik ke správnému vývoji mobilní aplikace (Canda, Azam and Sariman, 2016). Typicky se skládá z pěti až osmi součástí, kterými jsou: výzkum, návrh, vývoj, stabilizace (testování) a nasazení/údržba. Tyto části se mohou prolínat, kombinovat nebo dělit (GORAN JEVTIC, 2019). Například vývoj aplikace může být zahájen již při vývoji uživatelského rozhraní, a dokonce může být i užitečný při finalizaci UI. Pět hlavních součásti lze používat v několika metodologiích životních cyklů vývoje mobilních aplikací, jako jsou například Agile, Spiral, Waterfall a další. (David Ortinau *et al.*, 2021)

Životní cyklus mobilní aplikace napomáhá měřit a zlepšovat proces vývoje aplikace. Umožňuje detailní analýzu jednotlivých částí procesu. Výsledkem dodržování metodiky životního cyklu mobilní aplikace by měla být maximalizace výkonu při každém kroku.(GORAN JEVTIC, 2019)

Životní cyklus vývoje mobilní aplikace se může lišit v závislosti na velikosti budované aplikace. Některé kroky se stávají s velikostí aplikace zbytečné a některé naopak více detailní. Stejně tak se mohou měnit podmínky i v závislosti na cíli aplikace. Jestliže je cílem aplikace pro veřejné užití, musí být zohledněn zákazník jako stakeholder. Jestliže má aplikace financování, musí být jako stakeholder zapojen zástupce financující strany. (Kaur and Research Scholar, 2015)

3.2.1 Metodologie vývoje software

Metodologií vývoje software (SDLC metodologie) jsou procesy a postupy, které jsou používány v projektech pro úspěšné řízení životního cyklu vývoje software. SDLC metodologie slouží k vytvoření kvalitního software, který odpovídá požadavkům zákazníků v daném čase a s danými náklady. Mezi metodologie vývoje software patří waterfall metodologie, prototypující model, iterativní model, spirálový model, V-Shape Model a v neposlední řadě Agilní metodologie (scrum, extrémní programování, DevOps nebo KanBan). (Oliver Trunkett, 2020)

Tato část bude zaměřena na dvě populární metodologie. Těmi jsou waterfall a agile. Nejprve budou krátce popsány a následně bude určeno, pro jaké typy projektů, se která metodologie hodí.

Waterfall metodologie využívá sekvenční nebo lineární přístup k vývoji software. Jednotlivé fáze z životního cyklu jsou rozloženy do sekvence úkolů, které musí být ukončitelné v jasném pořadí a musí mít ukončovací kritérium, bez kterého nemůže začít další fáze (úkol). Toto kritérium je většinou určeno stakeholdery.

Hlavními výhodami waterfall metodologie je, že požadavky jsou vytvořeny na začátku projektu, takže příprava plánu projektu je relativně snadná a předem určitelná. Jednotlivé fáze mohou být rozděleny do menších úkolů a tím pádem lze lépe využít zdroje. Zároveň lze snadno zjistit, ve které fázi se projekt nachází. (Sherman, 2015)

Nevýhodami je, že získání perfektních požadavků v jednom bodě na začátku projektu může být složité. Pro správné využití musí být detailně zhodnoceny a rozloženy všechny úkoly a fáze. V neposlední řadě u waterfall metodologie hrozí zpoždění. (Casteren and van Casteren, 2017)

V agilních metodologiích jsou požadavky a výsledky upravovány v čase pomocí iterací (sprintů). Agile se spoléhá na samo organizující a efektivně komunikující teamy, které objevují a staví projekt za běhu, čímž se liší od waterfall metodologie, která je naplánována ještě před začátkem práce.(McKnight, 2014)

Ačkoliv v moderních firmách jsou čím dál více používány agilní metodologie, waterfall je podle studií stále používán v nadpoloviční většině společností. Použití metodologie se často odvíjí od cíle, kterého má být dosaženo (Kaur and Kaur, 2015). Jestliže vývoj závisí na striktních procesech společnosti nebo mnoha požadavcích, waterfall je užitečnější. To stejné platí, jestliže vlastník produktu není zapojen a čas vývoje a náklady jsou fixní. V opačném případě by měly být použity agile metodologie (Tim Parsons, 2019).

3.2.2 Výzkumná fáze/ Počátek

Určení cílů vytváření aplikace je základní částí vývoje, při které stakeholdeři určují hlavní součásti, které aplikace musí obsahovat a dodržet. Slouží především k tomu, aby vývojář rozuměl definici problému a byl seznámen s požadavky stakeholderů.

Ve fázi výzkumu také probíhá řízení požadavků (Requirements engineering). Řízení požadavků je proces definování, dokumentace a údržby požadavků. Jedná se o proces sběru a definice služeb poskytovaných daným systémem. Skládá se z požadavkového interview, specifikace požadavků, verifikace a validace požadavků a managementu požadavků (Mansi Breja, 2020).Požadavky musí být specifikovány co nejsrozumitelnější pro všechny části (jak

pro vývojáře, tak pro stakeholdery), aby se na jejich základě dalo dále pokračovat ve vývoji aplikace v požadované kvalitě.

Požadavky lze dělit do dvou hlavních částí:

- Funkcionálních požadavky: *„Vlastnosti produktu nebo funkce, které musí vývojář obsáhnout, aby umožnil uživatelům dosažení daných cílů.“* (Altexsoft, 2021) V mobilním programování se jedná o popis jednotlivých součástí UI a funkcionalit (například přihlášení, registrace nebo spuštění cvičení). Často jsou popsány pomocí use case (případů užití).
- Nefunkcionální požadavky: *„Softwarový požadavek, který nepopisuje, co bude software dělat, ale jak to bude dělat.“* Jedná se například o požadavky na výkon, požadavky na uživatelské rozhraní nebo požadavky na dostupnost. Jejich testování je složité, a proto jsou většinou hodnoceny subjektivně. (Kaur and Sharma, 2014)

Pomocnou metodologii pro určení, objasnění a organizování požadavků aplikace jsou také případy užití (use case). Případ užití je vytvořen z možných sekvencí interakcí mezi systémem a uživateli v daném prostředí sloužících k docílení daného cíle.

Kromě požadavků je ve výzkumné části také určena cílová skupina a jsou vytvořeny persony, což jsou fiktivní uživatelé budoucího produktu, které jsou archetypními uživateli určitých cílových skupin. (Dam Rikke Friis and Siang Teo Yu, 2021)

Mimo jiné jsou v požadavkové fázi také popsána technická a business omezení (technical and business constraints) a součástí může být i analýza trhu.

3.2.3 Návrhová fáze

Jakmile jsou důkladně a jasně definovány cíle aplikace, mělo by být jasné, co má vyvíjená aplikace poskytovat. V návrhové (design) fázi je vytvořen UX návrh, UI návrh a softwarová architektura. Vývojář by se měl soustředit na funkcionalitu, která je očekávána od aplikace i uživatelského rozhraní. Namísto vývoje konečného uživatelského rozhraní se v této fázi používají wireframy, které umožňují zobrazit návrh uživatelského rozhraní a rychle ho měnit.(Thomas and Jayanthila Devi, 2021)

3.2.3.1 UX návrh

User experience design je nejčastěji vytvořen pomocí wireframů, mockupů nebo prototypů. Při vytváření UX designu je nutné brát v potaz omezení platformy. Každá má jiné, což znamená, že user experience jedná platformy nemusí odpovídat omezení platformy jiné. Ty jsou často dostupné ve formě návodů přímo na webových stránkách platformy (viz. (Android Developers, 2021b)). Mimo platformy také UX rozhodnutí ovlivňuje zařízení, na které je aplikace vyvíjena. Například UX wireframy na smartphony nebudou stejné jako na tablet (Gavrilova Julia, 2020).

3.2.3.2 UI návrh

Po vytvoření UX návrhu je vytvořen UI návrh. Zatímco UX návrh je většinou černobílý, UI návrh zobrazuje všechny barvy a grafiky. Stejně jako u UX má každá platforma a zařízení své vlastní specifické vzhledy. (David Ortinau *et al.*, 2021)

3.2.3.3 Softwarová architektura

Krom UX a UI návrhu by také měla návrhová fáze obsahovat softwarovou architekturu, která specifikuje užívaný programovací jazyk, celkový design i jednotlivé technologie. Dále také platformu, na které bude aplikace fungovat, komunikaci aplikace (například se serverem atd.) a bezpečnost. (GORAN JEVTIC, 2019)

Softwarová architektura je ve zkratce organizace systému. Ta obsahuje všechny komponenty a jejich vzájemné interakce, prostředí, ve kterém fungují a základy použité pro vývoj aplikace.

3.2.4 Vývojová fáze

Jakmile jsou správně splněny první dvě fáze, měl by být jasně zřetelný obraz toho, co a jak je potřeba naprogramovat. Vývojová fáze slouží k vývoji aplikace na základě znalostí z počáteční a návrhové fáze. Na jejím konci je vytvořena aplikace se všemi předem definovanými funkcionalitami. (Thomas and Jayanthila Devi, 2021)

Vývoj web/desktop aplikace obsahuje akce, které obsahují nalezení správných technologií, jako je například programovací jazyk nebo specifická platforma. Vývoj

mobilních aplikací je do určité míry podobný, nicméně obsahuje jisté limity spojené se softwarovými a hardwarovými omezeními. Mobilní trh se neustále vyvíjí a s tím přichází i velké množství nových technologií. To poskytuje větší výběr, ale i možnost nesprávné volby použité technologie. (N Inukollu *et al.*, 2014)

3.2.5 Testovací fáze

Jakmile je aplikace hotová, je začít nutné průběžné a postupné testování, sloužící k odhalení všech chyb, které aplikace může obsahovat. Důkladné provedení QA (quality assurance) testování během vývojového procesu udělá aplikaci stabilní, použitelnou a bezpečnou na používání. Aplikace by měla být testovaná důsledně s co nejvyšším počtem reálných scénářů, které odhalí časté i náhodné chyby. Ve většině případů je doporučeno, aby aplikaci testovala jiná osoba nebo tým, než který ji vyvíjel, protože do testování je pak přinesen nový vhled a je možné narazit na problémy, které vývojářům uniknou. Jestliže je aplikace důsledně protestována, může se posunout do další fáze. (Thomas and Jayanthila Devi, 2021)

Možností testování je celá škála. Většina z nich si žádá vytvoření testovacích případů (test case), ukládání výsledků testů pro evaluaci kvality softwaru a retestování. Mezi možné způsoby testování patří: UX testování, testování funkcionality, testování výkonu, bezpečnosti atd. (Invonto, 2021)

3.2.6 Nasazení/Údržba

Jako nasazení je běžně brán moment, při kterém je mobilní aplikace nahrána do jednoho z obchodů s aplikacemi (jako je App Store nebo Google Play) a uživatelé tak mohou získat kopii aplikace. V případě nekomerčních aplikací se jedná o začátek užívání aplikací společností.

Tato fáze může (nemusí) mít údržbu. Údržba je občas brána jako vlastní fáze a slouží k udržování kvality a funkce aplikace. Jestliže je tato fáze zanedbána, stává se po nějaké době aplikace nepoužitelnou nebo zastaralou. Při údržbě jsou opravovány chyby, které nebyly nalezeny při testování a vydávány nové verze aplikace, které mohou přidávat novou funkcionalitu nebo pouze upravují problémy staré aplikace. (Upadhayay Dev, 2021)

3.3 Operační systém Android

Operační systém android je open-source operační systém založený na upravených verzích Linux jádra a slouží primárně pro dotyková mobilní zařízení jako jsou například chytré telefony nebo tablety. Android je vytvořený převážně společností vývojářů známou jako Open Handset Alliance (OHA) a komerčně sponzorovaný společností Google. Jedná se o nejvíce rozšířený operační systém na mobilních zařízení (k roku 2020 vlastní 75 % trhu). (Agarwal Tarun, 2020)Vývoj na Android podporuje především programovací jazyk Java, i když možné jsou i další jazyky. První verze Android Development Kitu (SDK) byla zveřejněna v roce 2008.(Chen James, 2021)

3.3.1 Android Architektura

Android je strukturovaný ve formě softwarového zásobníku, který obsahuje aplikace, operační systém, run-time prostředí, middleware, služby a knihovny. Každá vrstva tohoto zásobníku a její části je blízce integrovaná a vyladěná tak, aby umožnila optimální vývoj aplikací a spouštěcí prostředí pro mobilní zařízení. (Smyth, 2022)

Vývoj aplikací na OS Android vyžaduje porozumění celkové architektuře Android. Aplikace jsou napsané v jazyce Java nebo Kotlin a následně zkompilovány do bytekódového formátu v Android Studiu (vývojové prostředí). Když jsou tyto aplikace instalovány na zařízení, tento bytekód je opět kompilován pomocí Android Runtime do přirozeného formátu používaného procesorem. Hlavním účelem Android architektury je výkon a účinnost, obojí při spuštění aplikace i při implementaci opětovného použití při vývoji aplikací. (Android Developers, 2021e)

Od spodní vrstvy je OS Android složen z:

- Linux jádro – Linux jádro je „srdce“ operačního systému, které spravuje vstupní a výstupní žádosti software. Tím poskytuje základní funkce systému, jako je například správa procesů, správa paměti, správu zařízení, jako například kamera, display atd.
- Knihovny – V architektuře je nad Linux jádrem set knihoven, které obsahují open-source webové prohlížeče, jako je například WebKit. Tyto knihovny slouží k přehrávání a nahrávání audia a videa. Dále také vlastní SQLite databázi, která

slouží k ukládání a sdílení aplikačních dat a SSL knihovny, které slouží k udržení bezpečí na internetu. (Thompson Barbara, 2022)

- Android Runtime – Android Runtime poskytuje klíčový komponent, který se nazývá Dalvik Virtual Machine. Tento virtuální přístroj je speciálně vyrobený a optimalizovaný pro Android. Jedná se o software, který umožňuje běh aplikací na zařízení. Užívá základní schopnosti Linux jádra, jako je například správa paměti nebo multithreading a umožňuje aplikacím spuštění jejich vlastních procesů.
- Aplikační framework – Vrstva aplikačního frameworku poskytuje higher-level služby aplikacím, jako například správu oken, zobrazovací systém, správu packagů nebo zdrojů. Vývojáři mohou tyto služby využívat ve svých aplikacích.
- Aplikace – Vrstva aplikací obsahuje všechny android aplikace a umožňuje další vytváření a instalování aplikací. (Agarwal Tarun, 2020)

První verze OS Android byla vydána roku 2008 a na rozdíl od ostatních verzí neměla žádnou přezdívku. Roku 2010 byla vytvořena první vizuální identita Android ve verzi 2.3 (Gingerbread). Zatím nejnovější verzí je Android 12, která byla uvedena 12. října 2021. (Raphael JR, 2022)

3.3.2 Android Studio

Android Studio je oficiální IDE (integrované vývojové prostředí) pro vývoj v Android, které obsahuje vše nutné pro vývoj Android aplikací (Android Developers, 2022b). Android Studio je volně ke stažení pro OS Windows, macOS a Linux. Jedná se o náhradu za Eclipse Android Development Tools jako hlavní IDE pro vývoj Android aplikací. V roce 2019 vyměnil Kotlin Java jako preferovaný programovací jazyk pro vývoj Android aplikací. Java je nadále podporovaná, stejně jako C++. (Lardinois Frederic, 2019)

Android Studio je založené na IntelliJ IDEA a krom jeho základních funkcí poskytuje několik vylepšení zvyšujících produktivitu při vývoji aplikací, jako je například (Android Developers, 2022c):

- flexibilní build systém založený na Gradle
- rychlý a víceúčelový emulátor
- unifikované prostředí umožňující vývoj pro všechna Android zařízení

- kódové šablony a GitHub integraci pro snadný vývoj jednoduchých aplikačních funkcí
- rozsáhlý výběr nástrojů a frameworků
- Lint nástroje umožňující měření výkonnosti, použitelnosti, kompatibility verzí a dalších problémů
- podporu C++ a NDK (Native Development Kit)
- v neposlední řadě také vestavěnou podporu pro Google Cloud Platformu

3.3.2.1 Struktura projektu v Android Studio

Každý projekt v Android Studio obsahuje alespoň jeden modul, který obsahuje složky zdrojového kódu (Android aplikační moduly, moduly knihoven a Google App Engine moduly). Projekt se skládá z gradle skriptů (build složky) a aplikačního modulu, který obsahuje:

- Manifesty – Obsahují AndroidManifest.xml složku
- Java – Obsahuje složky Java zdrojového kódu společně s JUnit testovacím kódem
- Res – Obsahuje všechny nekódové zdroje, jako například XML rozložení nebo bitmap obrázky (Android Developers, 2022b)

3.3.2.2 Android Emulator

Další důležitou součástí IDE Android Studio je Android Emulator. Android Emulator simuluje Android zařízení přímo v Android Studio, což umožňuje testování aplikace na různých zařízeních a API úrovních bez vlastnictví každého takového zařízení fyzicky. Emulované zařízení poskytuje téměř všechny možnosti reálného zařízení. Lze simulovat příchozí hovory i SMS, specifikovat lokaci, simulovat rychlost sítě nebo rotaci zařízení a mnoho dalšího. Emulované zařízení může být mobilní telefon, tablet, Wear OS (chytré hodinky) nebo například Android TV.

I testování aplikace je na emulátoru snadnější a rychlejší než na fyzickém zařízení. Například přenos dat je rychlejší na emulované zařízení než na reálné připojené pomocí USB.

Použití emulátoru je možné jak manuální skrz grafické uživatelské rozhraní, tak i skrze příkazovou řádku a konzoli emulátoru. (Android Developers, 2022f)

3.3.3 Programování Android aplikace

Programování Android aplikace zahrnuje vytváření logiky a algoritmů, které slouží k základu kódu v business softwaru, dodávání potřebných multimediálních prvků a poskytnutí zdrojů pro uživatelské rozhraní, jako například rozložení v XML, ikony nebo lokalizační řetězce.

Hlavní koncepty (Macadamian Technologies, 2012) tvořící Android aplikaci jsou:

- Aktivity – Aktivitou se označuje jedna zaměřená funkce, kterou uživatel může provést. Téměř všechny aktivity interagují s uživatelem, takže třída Aktivita zajišťuje vytvoření okna, ve kterém bude viditelné uživatelské rozhraní. Aktivity jsou nejčastěji zobrazené jako celoobrazovková okna, ale jejich nastavení může být i jiné (Späth, 2018).
- Views a View Skupiny – View (zobrazení) je prvek v rozhraní, jako například tlačítko nebo textový box. Několik view je spojeno do view skupiny, která reprezentuje hierarchickou organizaci rozložení a obsahu.
- Intents – Intent (Úmysl) je specifikace žádání o akci. Intent umožňuje komunikaci mezi aktivitami, a to buď explicitně, kdy lze přímo jmenovat aktivitu, na kterou intent odkazuje, nebo implicitně tím, že je jmenována vyžadovaná akce, se kterou je svázána aktivita, jež ji může vykonat. Implicitní intent je vykonán pomocí navázání aktivit na intent filtry, což jsou podmínky specifikující akce, které může aktivita provést. Intents lze také použít pro odesílání a příjem přenosových zpráv, které upozorňují systém nebo aplikační událost. Intents mohou také obsahovat data pro umožnění předávání parametrů mezi aktivitami.
- Events a event listeners – Event je událost uvnitř aktivity, kterou lze provést přímo metodou spojenou s view prvkem, která tento event produkuje. Mimo to může být také event zveřejněn, aby registrované metody (listeners) mohli zasáhnout. (Dewang Nautiyal, 2021)
- Fragments – Ačkoliv se nejedná přímo o hlavní koncept tvořící Android aplikaci, za zmínku stojí také fragment. Fragment reprezentuje znovu použitelnou část uživatelského rozhraní aplikace. Fragment definuje a spravuje své vlastní rozložení, má vlastní životní cyklus a ovládá vlastní vstupní události. Fragmenty, na rozdíl od aktivit, nemohou existovat nezávisle, ale musí být spojené s aktivitou nebo dalším

fragmentem. Hierarchie zobrazení fragmentu je přímo vázaná na aktivitu nebo fragment. (Brambilla and Fraternali, 2015)

3.4 Moderní technologie a přístupy k vývoji mobilních aplikací

V této sekci jsou uvedeny moderní technologie a přístupy, které budou použity při tvorbě praktické části diplomové práce. Ty jsou vybrány analýzou moderních přístupů a trendů v programování a analýzou použitelnosti v prostředí hudební aplikace. Nejprve budou technologie a přístupy popsány a následně budou zmíněny některé další moderní technologie a důvod, proč pro aplikaci zvoleny nebyly.

3.4.1 Synchronní a asynchronní programování

Programování lze označit jako synchronní, jestliže jeho kód je vykonán přímo na hlavním vlákně. V tom případě synchronní operace blokuje instrukce do doby, než je provedena. Jestliže máme několik za sebou jdoucích operací, každá z nich musí počkat, než je dokončena operace jdoucí před ní. V aplikaci na mobilním zařízení to může znamenat, že při běhu aplikace zamrzne a čeká na vykonání operace předtím, než jí lze opět používat a pokračovat v jejím ovládání. Na počítači je blokující operace většinou naznačena zablokováním funkcí a například změnou ikony kursoru (Hardy Scott, 2019). Aby běh aplikace nebyl přerušený čekáním na dlouhotrvající procesy, využívá se asynchronní programování. Asynchronní programování je forma paralelního programování, která umožňuje provést operaci odděleně od hlavního vlákna. Když je operace provedena (ať už úspěšně nebo neúspěšně) tak je hlavní vlákno upozorněno. Tento přístup má za využití v doporučených případech své výhody, mezi hlavní jde zařadit vyšší výkonnost aplikace a responzivitu (Vogel Eric, 2011). Především responzivita je základem dobré UX (user experience). Každá knihovna umožňující asynchronní funkci by tedy měla zvládat 4 základní věci: explicitní provedení, jednoduchou správu vláken, snadnou skládatelnost a minimální množství postranních efektů. (Patel Keval, 2016)

3.4.1.1 Vlákna

Pro snazší porozumění asynchronnímu programování je nutné vysvětlit, co je to vlákno a jak funguje práce s vlákny. Vlákno je jeden proces, který program používá k dokončení

úlohy. Někdy se také vlákno označuje jako odlehčený proces. Ačkoliv jednotlivé procesy jsou od sebe striktně odděleny, vlákna procesů sdílejí paměť i jiné systémové prostředky a tím pádem mají jednodušší vzájemnou komunikaci.(Kavi, 1998)

Každé vlákno může provádět v jedné chvíli pouze jednu úlohu. Jedno vlákno v počítači je reprezentováno jedním jádrem procesoru. Přirozeně vlákna na jednom jádru fungují sekvenčně a tím pádem musí být úloha ukončena, aby mohla být provedena úloha následující. Činnosti na jednom jádru mohou být prováděny také kvaziparalelně (rychlé přepínání mezi vlákny, čímž dochází ke zdánlivému paralelismu). Technicky vzato v systémech a programech nepodporujících více jader má každý proces pouze jediné vlákno.

Jelikož v dnešní době již mají zařízení jader několik, může být úloh v dané chvíli provedeno více. Tím pádem lze v rámci jednoho procesu vytvořit větší počet vláken, které ho budou zpracovávat. Programovací jazyky, které podporují více vláken mohou použít několik jader k dokončení několika úloh (Bamberg, Schonning Nick and Willee Hamish, 2022).Tyto úlohy se nemusí týkat stejné činnosti (například mohou číst požadavky uživatele, zpracovávat data a vykreslovat výstupy zároveň). Použití více vláken v jedné chvíli se označuje jako „multithreading“. (Burns Bill, 2020)

Asynchronní programování se nedá označit za něco, co by se mělo použít v každé instanci. V programování se nachází několik situací, kdy je asynchronní programování zbytečné. Jestliže je například použito pro snadný a krátký výpočet, může kód být pomalejší a méně přehledný. Asynchronní programování by tedy mělo být použito především u operací se složitějším výpočtem a u takových operací, které by mohly zbrzdit funkci aplikace.

3.4.2 Reaktivní programování

Reaktivní programování je programovací paradigma zaměřené na datové toky a propagaci změn. To znamená, že by mělo snadno vyjádřit statistické nebo dynamické datové toky v programovacích jazycích a model následně automaticky propaguje změny v celém datovém toku.

Zjednodušeně v reaktivním programování datové toky vysílané jedním komponentem a vlastní struktura poskytnutá reaktivními knihovnami bude propagovat tyto změny na ostatní komponenty, které jsou oprávněné k obdržení těchto změn (Ryax technologies, 2021).

Reaktivní programování se skládá ze tří základních součástí: observable, observers a schedulers.

3.4.2.1 Observable

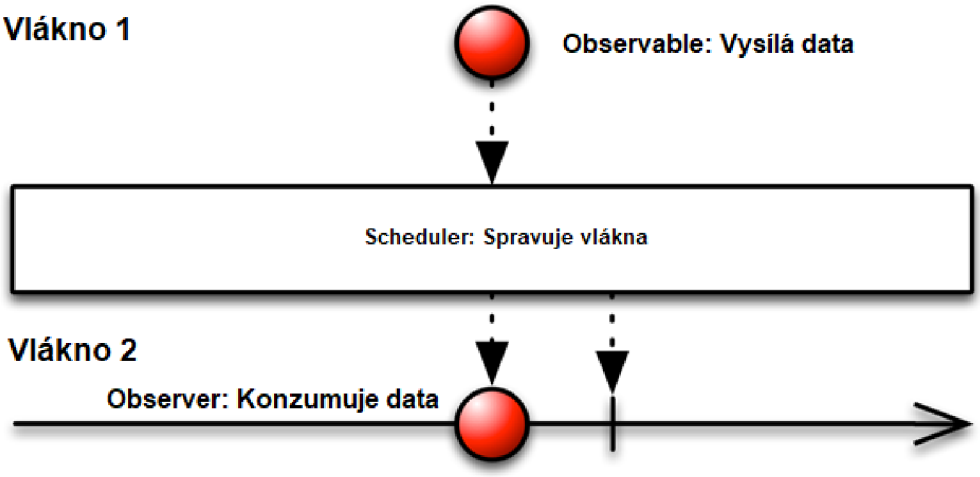
Observable jsou datové toky. Observable balí data, které mohou být předávána mezi jednotlivými vlákny tím, že vysílají data v časových intervalech (jednou nebo víckrát, podle jejich nastavení). Ve zkratce se jedná o dodavatele, kteří zpracovávají data a následně je rozesílají ostatním komponentám (Patel Keval, 2016).

3.4.2.2 Observers

Observers zpracovávají datové toky vysílané pomocí observable. Observers jsou přidělené observable pomocí metody subscribeOn(), která zajistí chycení všech dat poslaných pomocí observable. Jestliže observable vyšle data, všechny observers s připsáním obdrží data při dalším volání onNext() metody. Při obdržení mohou provádět širokou škálu operací jako například změnu uživatelského rozhraní. Jestliže je výsledek observable chybový, observer se o tuto informaci dozví při volání onError() metody (Ari Dler, 2019).

3.4.2.3 Schedulers

Schedulers slouží ke správě vláken, a proto jsou součástí reaktivního programování schedulers. Schedulers přímo říkají observable i observers na kterém vlákně mají být provedeny. Jestliže má být observers řečeno, na kterém vlákně mají běžet, je použita metoda scheduleOn(). Jestliže má být stejná informace řečena observable, je použita metoda scheduleOn(). ReactiveX (knihovny pro práci s reaktivním programováním) poskytují několik předdefinovaných vláken jako například Schedulers.newThread(), který vytvoří nové vlákno nebo Schedulers.io(), které spustí kód na IO vlákně (Stoyanov Dennis, 2017).



Obrázek 3 - Podstata reaktivního programování (Patel Keval, 2016)

Na Obrázku 3 lze vidět jednoduchý příklad funkce reaktivního programování.

3.4.2.4 Reactive Extentions

Reactive Extentiosn (ReactiveX) je knihovna (sada nástrojů) sloužící k psaní asynchronních a reaktivních programů pomocí observable sekvencí. Rozšiřují observer vzor tak, aby podporoval sekvence dat nebo událostí a přidává operátory, které umožňují napsat deklarativní sekvence, a přitom abstrahovat starosti spojené s nízko úrovňovou správou vláken, synchronizací, bezpečností vláken a neblokující I/O (ReactiveX contributors, 2011).

Reactive Extensions jsou vytvořeny pro většinu moderních programovacích jazyků (RxJava, Rx.NET, RxPY, RxKotlin, RxSwift, RxPHP) a několik platform a frameworků (RxNetty, RxAndroid, RxCocoa).

3.4.3 Kotlin

Kotlin je volně dostupný a víceúčelový programovací jazyk pro JVM (Java Virtual Machine) a operační systém Android, který kombinuje výhody objektově orientovaného a funkcionálního programování. Je zaměřen především na interoperabilitu, zabezpečení, jednoduchost a velké množství dostupných a podporovaných nástrojů.

Kotlin byl vyvinut v roce 2010 společností JetBrains (která vlastní IntelliJ IDEA) a volně dostupný je od roku 2012(Heller Martin, 2020). Od roku 2019 je Kotlin oficiálně podporován jako preferovaný jazyk pro všechny vývojáře mobilních aplikací na platformě Android (Lardinois Frederic, 2019). Kotlin je designovaný tak, aby plně spolupracoval

s programovacím jazykem Java a verze JVM závisí na knihově tříd Java, ale díky odvození typu je jeho syntaxe stručnější (JetBrains, 2021).

3.4.4 Kotlin Coroutines

V dnešní době je již asynchronní a neblokující běh aplikací standardem. Ať už se jedná o aplikaci na straně serveru, desktop aplikaci nebo aplikaci mobilní, je nutné, aby její běh byl pro uživatele plynulý. V programování se nachází mnoho možností, jak asynchronního běhu dosáhnout a v jazyku Kotlin se používají vysoce flexibilní Coroutines, které jsou podporovány na jazykové úrovni skrz knihovny. Coroutines neslouží pouze k asynchronnímu programování, ale také poskytují další možnosti jako třeba concurrency, actors atd (Semy0nov Pavel, Petrakovich Victoria and Polyakov Andrey, 2021).

Coroutine je vývojářský vzor, který lze použít v kódu při programování na operační systém Android, který běží asynchronně. Do jazyku Kotlin byly Coroutines přidány ve verzi 1.3 a jsou přebrané z podobných konceptů jiných jazyků (Kousen, 2020). Na Android řeší především dva hlavní problémy: Běh dlouhotrvajících operací, které by jinak mohli zablokovat hlavní vlákno a tím způsobit zamrznutí aplikace a dále také poskytují bezpečnou cestu pro síťová volání a operace s diskem z hlavního vlákna (Android developers, 2022).

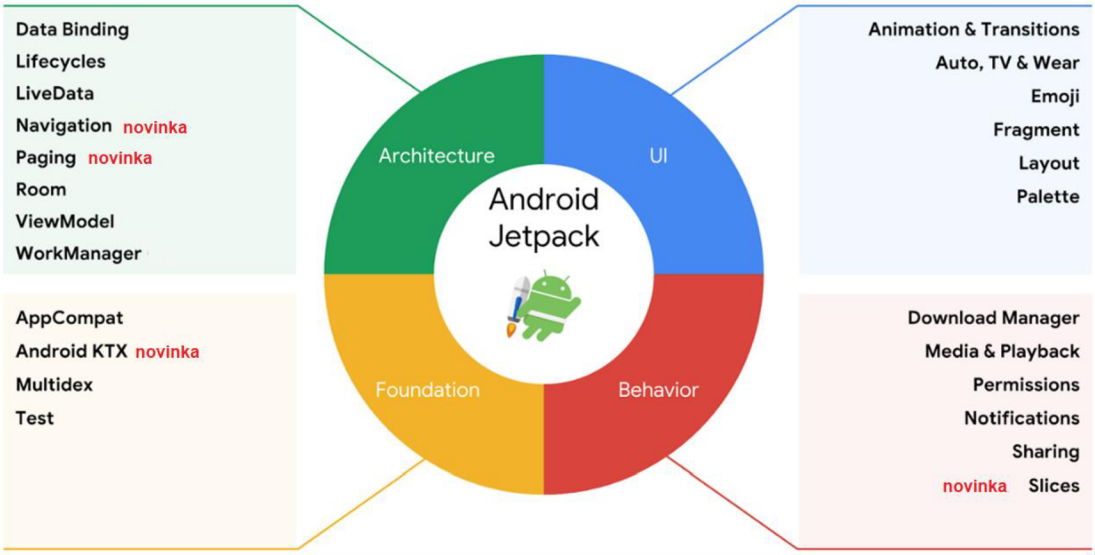
Jazyk Kotlin poskytuje pouze minimální množství nízko úrovňových API (application programming interface) ve své standardní knihovně, aby mohly ostatní knihovny dostatečně coroutines využít. Stejně jako ostatní podobné jazyky, async a await nejsou ve standardní knihovně Kotlin zabudovány. Krom toho také koncept suspendování funkce Kotlin poskytuje bezpečnější a méně chybovou abstrakci pro asynchronní operace.

Kotlinx.coroutines je objemná knihovna pro coroutines vyvinutá společností JetBrains. Sama o sobě obsahuje velké množství vysokoúrovňových funkcí, jako je například launch nebo async.(Polyakov Andrey *et al.*, 2021)

3.4.5 Android Jetpack

Android Jetpack je set knihoven, která slouží k snadnému psaní vysoce kvalitních aplikací podporujících starší verze operačního systému Android. Google nabízí strukturovanou dokumentaci, která velmi zjednodušuje začátky práce s nástrojem. Podle oficiálních dat používá Android Jetpack 99 % aplikací z PlayStore.

Android Jetpack se skládá primárně ze čtyř částí: architektury, uživatelského rozhraní, základu a chování (Muntenescu Florina, 2020).



Obrázek 4 - Součásti Android Jetpack (Kaseb Kayvan, 2020)

Aplikace podporované OS Android mohou běžet na různých verzích android platformy, protože součástí Android Jetpack je podpora zpětné kompatibility. Android Jetpack je mimo jiné navržen tak, aby zvládal i moderní designové postupy, jako je například oddělení hrozeb, testovatelnost, volná párování (loose coupling), Observer vzor, inverzi kontroly, stejně jako produktivní vlastnosti, jako je například integrace Kotlin. To všechno je nápomocné pro vytvoření robustních a vysoce kvalitních aplikací za pomoci co nejméně kódu co nejsnadněji. Ačkoliv jednotlivé komponenty Android Jetpack jsou postavené pro společnou součinnost, není vyžadováno využití všech. To v důsledku znamená, že lze integrovat pouze ty části, které přímo řeší určité problémy a zároveň zachovat komponenty aplikace, jejichž funkcionalita je odpovídající (Kaseb Kayvan, 2020).

3.4.5.1 Android Jetpack: Architecture (Architektura)

Architektura obsahuje osm různých knihoven a nástrojů, které slouží k architektuře aplikace a ovládání a zobrazení dat používaných aplikací. Většina knihoven obsažených v Android Jetpack architektuře jsou již existující, nové jsou tři: Navigation, Paging a WorkManager.

3.4.5.1.1 Room

Room je vytrvalostní knihovna, která obsahuje abstraktní vrstvu přes SQLite. To je C-jazyk, který implementuje malý, rychlý, spolehlivý a nezávislý engine SQL (SQLite Consortium, 2022). Tato vrstva umožňuje robustnější přístup do databáze při plném využití SQLite (Android Developers, 2022e).

Room je objektově/relační mapování metadat a na rozdíl od ostatních API má lepší implementaci, nevyžaduje příliš času ani výkonu a zároveň je použitelnější pro změny v datových grafech, protože nevyžaduje žádné aktualizace dotazů (Adamovic Milan, 2019).

Samotná společnost Google doporučuje užívání Room místo SQLite, protože efektivně řeší některé předchozí problémy.

Nejdůležitější komponenty Room knihovny mohou být klasifikovány do tří kategorií:

- Entity – Třída, která reprezentuje tabulku databáze, když pracuje s Room.
- DAO (Data Access Object) – Jedná se o mapování SQL dotazů přímo k funkcím. Obsahuje metody využívané k přístupu k databázi.
- Room databáze – Zjednodušuje běh databáze a slouží jako přístupový bod ke skryté SQLite databázi (Kaseb Kayvan, 2020).

3.4.5.1.2 WorkManager

WorkManager je zpětně kompatibilní, flexibilní a jednoduchá knihovna pro práci, kterou je možné odložit na pozadí. Jedná se o komponent architektury pro práci v pozadí, který kombinuje oportunistické a garantované spuštění. Oportunistické spuštění znamená, že WorkManager provede práci z pozadí co nejdříve to půjde. Garantované spuštění znamená, že WorkManager ovládá logiku a spouští práci v situacích, jako je například odchod z aplikace.

WorkManager je především flexibilní, ale má i další výhody. Podporuje asynchronní i periodické úlohy, omezuje problémy se sítěmi, pamětí a nabíjením, pomáhá řetězit komplexní úlohy včetně paralelního běhu, zvládá zpětnou kompatibilitu API a není závislý na práci s Google Play.

Knihovna WorkManager je nejlépe využitelná v případech, kdy je prováděna úloha, která je užitečná dokončená i za předpokladu, že už byla opuštěna plocha spojená s ní. Například se jedná o nahrávání logů (záznamy výkonu), aplikace filtrů a ukládání obrázku nebo pro pravidelné obnovování lokálních dat se sítí (Android Developer Relations, 2022).

3.4.5.1.3 Lifecycle-aware komponenty

Pro pochopení lifecycle-aware komponent je nutné definovat, co to vůbec znamená „life-cycle awareness“ (vědomost životního cyklu). Objekt je lifecycle-aware, jestliže je schopen detekovat a odpovídat na změny v životním cyklu ostatních objektů v aplikaci. Komponenty, které jsou si vědomé životního cyklu, provádějí akce v závislosti na změnu v statusu životního cyklu další komponenty, jako je například aktivita nebo fragment. Tyto komponenty pomáhají dosáhnout lépe organizovaného kódu, který je snazší na údržbu (Android Developers, 2021c). Některé Android komponenty, jako například LiveData, jsou přirozeně lifecycle-aware. Jestliže třída není lifecycle-aware, může tuto vlastnost implementovat pomocí LifecycleObserver rozhraní v této třídě.

Android Jetpack přichází se třídou LifeCycle, což je abstraktní třída, která má Android LifeCycle přiřazený. LifeCycle je konkrétně třída, která drží informaci ohledně stavu životního cyklu komponenty. Objekty mohou tento stav zkoumat a následně podle něj jednat. Aby mohly objekty tento stav sledovat, jsou vytvořeny dva hlavní koncepty, které jsou reprezentovány enumeracemi Events a States (Kaseb Kayvan, 2020).

Neschopnost zvládnout kontrolu životních cyklů je dlouhodobě jeden z největších problémů ve vývoji aplikací na Android. Jestliže není korektně zvládnuto aplikování životních cyklů, může dojít k závažným problémům, jako je například memory leak a další chyby. V aplikacích, které lifecycle-aware komponenty nepoužívají, se většinou implementují jednotlivé akce závislých komponent přímo do metod životních cyklů aktivit a fragmentů. To často vede ke špatné organizaci kódu a k množení chyb. Jestliže použijeme komponenty vědomé životního cyklu, můžeme kód závislých komponent vyjmout z metod životních cyklů a dát je přímo do komponent (Android Developers, 2021c).

3.4.5.1.4 ViewModel

ViewModely jsou objekty, které podporují data pro komponenty uživatelského rozhraní a zajišťují výdrž konfiguračních změn. Mezi ty patří například otočení mobilního zařízení. Konfigurační změny jsou takové změny, které vyžadují úpravu celé aktivity. Jestliže nejsou data z aktivity pořádně uložena a obnovena z přerušené aktivity, jsou ztracena a uživatelské rozhraní přestává fungovat (úplně nebo částečně). Proto se místo hromadění dat uživatelské rozhraní v aktivitě používá ViewModel.

Další problém je, že ovladače uživatelského rozhraní (aktivity a fragmenty) často provádějí asynchronní volání, která mohou zabrat delší čas, než dostanou výsledek. Proto musí ovladače uživatelského rozhraní spravovat volání a následně zajistit, aby je systém ukončil, a tudíž aby nedocházelo k potenciálním memory leak situacím. To vyžaduje údržbu, protože objekt je po konfigurační změně opět obnovený, což vede k velké ztrátě zdrojů kvůli opakování procesů (Chopra Hitesh, 2021).

3.4.5.1.5 LiveData

LiveData je observable data holder třída. Narozdíl od ostatních observable, LiveData je lifecycle-aware, což znamená, že respektuje životní cyklus ostatních aktivit. LiveData bere observer, který je reprezentovaný třídou Observer, v aktivním stavu, jestliže její životní cyklus je ve stavu „started“ nebo „resumed“. LiveData upozorňuje pouze aktivní observers a neaktivní observers registrované k odběru LiveData objektů upozorněny nejsou (Android Developers, 2021d). Jinými slovy LiveData zkoumají stav observeru před tím, než cokoliv provedou. Hlavním rozdílem od ostatních observable přístupů je lifecycle vědomí. To znamená, že rozumí tomu, kdy je uživatelské rozhraní na ploše, mimo plochu nebo ukončené. LiveData ví o stavu uživatelského rozhraní, protože byl poslán při volání observe (Kaseb Kayvan, 2020).

LiveData zajistí, že uživatelské rozhraní odpovídá stavu dat a to tím, že informují observer objekty, když se změní životní cyklus. Je možné změnit kód tak, aby upravil uživatelské rozhraní v observer objektech. Místo toho, aby se uživatelské rozhraní změnilo vždy, když se změní uživatelská data, změní se při každé změně v observeru. Mezi velké výhody patří také to, že nedochází k memory leak situacím, protože observers jsou přímo spjaty s Lifecycle objekty a uklidí po sobě po provedení životního cyklu. Dále také nedochází k přerušením programu díky pozastaveným aktivitám, není nutné manuálně ovládat životní cykly, dochází k automatické aktualizaci dat a je možné snadno sdílet zdroje (Android Developers, 2021d).

Problém LiveData je, že nemají žádnou veřejnou metodu pro úpravu své hodnoty. Proto jsou používány MutableLiveData, které tuto hodnotu upravují pomocí setValue, která okamžitě upraví hodnotu (synchronně) nebo postValue, která hodnotu upraví asynchronně, jakmile je UI vlákno aktivní (Sharma Prateek, 2020).

3.4.5.1.6 Navigation

Navigace je název pro interakce, které uživatelům dovolují pohyb z různých míst v aplikaci. Android Jetpack Navigation pomáhá s implementací navigace, od jednoduchého kliknutí na tlačítko až po více komplexní vzory, jako jsou například aplikační okna a navigační složky (Kaseb Kayvan, 2020).

Zatímco aktivity jsou vstupní body do uživatelského rozhraní aplikace, jejich nízká flexibilita při sdílení dat mezi sebou a přechody způsobila, že nejsou ideální architekturou pro stavbu navigace v aplikaci. Navigation je framework pro strukturalizaci aplikačního uživatelského rozhraní tím, že se soustředí na jednoaktivitovou aplikační architekturu. Navigation umožňuje ukázat přechody a Back chování. Dále také poskytuje plnou podporu pro deep odkazy a pomoc pro propojení Navigation do odpovídajících widgetů uživatelského rozhraní (Cechetto Thiago, 2018).

Navigation se skládá ze tří částí:

- Navigační graf – XML zdroj, který obsahuje všechny informace týkající se navigace centralizované v jedné lokaci. To se týká všech individuálních obsahů v aplikaci, které se nazývají destinace, stejně jako cest, kterými uživatel může skrz aplikaci procházet.
- NavHost – Prázdné uložiště, které zobrazuje destinace z navigačního grafu. Navigation obsahuje defaultní NavHost implementaci, která se jmenuje NavHostFragment a který zobrazuje cíle fragmentů.
- NavController – Objekt, který spravuje aplikační navigaci v NavHost. NavController ovládá výměnu obsahu v NavHost když se uživatelé pohybují skrz aplikaci (Lyla Fujiwara and Android Developers, 2021).

Mezi hlavní výhody použití Navigation patří zvládání transakce fragmentů, implementace Back, podpora deep spojení, poskytování snadno animovaných přechodů a podpora běžného navigačního vzoru (Kaseb Kayvan, 2020).

3.4.5.1.7 Paging

Většina android aplikací pracuje s velkými soubory dat. Přesto při nahrání a zobrazení potřebují z těchto dat v danou chvíli pouze malou část. Jestliže není ovládnutý proces nahrávání dat, může se stát, že požadovaná data jsou jiná, než o které bylo zažádáno. To vede k plýtvání baterie i šířky pásma. Jestliže jsou zobrazovaná data neustále aktualizována,

stává se udržení synchronizovaného uživatelského rozhraní a zasílání pouze malého množství dat přes síť složité. Paging knihovna dovoluje nahrávat data postupně a elegantně. Poskytuje totiž objemné, ale ohraničené listy, stejně jako listy neohraničených stránek, jako například neustále se měnící výpisy. Důležitá je u Paging knihovny také integrace s RecyclerView, která je u aplikací typicky spojena se zobrazováním velkých data setů a spolupracuje s LiveData, RxJava i RxKotlin, kvůli zkoumání nových dat v uživatelském rozhraní (Cechetto Thiago, 2018).

Součásti Paging knihovny je:

- DataSource – DataSource je třída, ve které se určuje, kolik dat má být v aplikaci nahráno. Skládá se ze tří podtříd, které určují, co a odkud chceme nahrát a těmi jsou ItemKeyedDataSource, PageKeyedDataSource a Positional DataSource.
- PagedList – PagedList je třída, která umožňuje nahrát aplikační data. Jestliže uživatel přidává nebo vyžaduje stále větší množství přidaných dat, jsou zapsána do předchozího PagedList. Jestliže je v datech změna, je vytvořena nová instance PagedList a následně poslána observer pomocí LiveData a RxJava.
- New Data – Kdykoliv PagedList načte novou stránku, PagedListAdapter upozorní RecyclerView o přítomnosti nových dat a ten aktualizuje data uživatelského rozhraní.

Jestliže je Paging knihovna užívána dle doporučení, bude aplikace zobrazovat svůj obsah rychleji, bude využito méně paměti, nebudou se načítat zbytečná data (například je možné načíst pouze jednu nebo dvě stránky v jeden moment) a je možné lépe spolupracovat s LiveData a ViewModel a tím snadněji aktualizovat a zkoumat data (Team MindOrks, 2019b).

3.4.5.1.8 Data Binding

Data Binding je pomocná knihovna, která umožňuje spojení prvků uživatelského rozhraní v jednotlivých rozloženích s datovými zdroji v aplikaci pomocí deklarativního formátu namísto nutnosti přímého programování tohoto spojení (Android Developers, 2021a). To znamená, že kód, který volá například metoda findViewById() sloužící k nalezení TextView a následně navázání na viewModel username se změní z:

```
findViewById<TextView>(R.id.sample_text).apply {
    text = viewModel.userName
}
```

Obrázek 5 - findViewById() příklad (Android Developers, 2021a)

na:

```
<TextView
    android:text="@{viewmodel.userName}" />
```

Obrázek 6 - Data Binding příklad (Android Developers, 2021a)

Tím se z aktivit dostane většina framework volání uživatelského rozhraní a stanou se snazšími na údržbu. Zároveň se také zvýší výkonnost aplikace a přestávají se objevovat memory leak situace a null pointer výjimky (Adefioye Temidayo, 2018).

3.4.5.2 Android Jetpack: Foundation (Základy)

Základy Android jetpack obsahují základní systémové komponenty, kotlin rozšíření a testovací knihovny (Moore Kevin D., 2018).

3.4.5.2.1 AppCompat

Když je publikovaná nová verze Android, Google musí pokračovat v podpoře starších verzí. AppCompat je set podpůrných knihoven, který umožňuje aplikacím vyvíjeným pro nové verze Android pracovat se staršími verzemi Android (Banes Chris, 2014). AppCompat knihovna v Jetpack základech obsahuje všechny komponenty v7 knihovny. Navíc také poskytuje implementační podporu pro material design uživatelské rozhraní, které je pro vývojáře užitečné. Bez AppCompat knihovny je velmi složité naprogramovat aplikační panel, navigační menu, oprávnění k hardware nebo dialog box. Navíc také umožňuje využití méně zdrojů pro získání stejné funkcionality a snadné sdílení dokumentů a složek mezi platformami (Mishra Rishu, 2021b).

3.4.5.2.2 Android KTX

Android KTX (Android Kotlin Extensions) je jednou z nejnovějších součástí Android Jetpack. Jedná se o set funkcionalit, který umožňuje Kotlin programátorům snadný a příjemný vývoj aplikací. Android KTX se nesnaží přidat nové vylepšení do již existujících API Android, ale snaží se zjednodušit vývoj na Android pomocí plného využití dosavadních API v Kotlin (Team MindOrks, 2019a). KTX využívá z Kotlin především rozšiřující funkce, vlastnosti, lambdy, pojmenované parametry, defaultní hodnoty parametrů a coroutines (Muntenescu Florina and Android Developers, 2022).

KTX obsahuje i několik spojení mezi KTX moduly a ostatními knihovnami z Android Jetpack. Jestliže chce pracovat s Navigation knihovnou, můžeme například použít například android.arch.navigation:navigation-common-ktx (mimo další) (Moore Kevin D., 2018).

3.4.5.2.3 Test

Testování je základní část procesu vývoje aplikací. Při konzistentním testování lze ověřit bezchybnost aplikace, funkcionalitu a uživatelnost před veřejným vydáním. Krom jiného také získáváme okamžitou zpětnou vazbu na chyby, rychlou detekci ve vývojovém cyklu aplikace, bezpečnější refactoring kódu a s ním i optimalizaci a v neposlední řadě stabilní vývojovou rychlost (Android Developers, 2022g).

Testovací část Základů obsahuje testovací framework Espresso UI a AndroidJUnitRunner pro unit testování. Unit testy jsou testy malých částí kódu týkajícího se logiky, většinou se jedná o testování na úrovni individuálních metod. Espresso slouží k testování jednotlivých prvků uživatelského rozhraní (Mishra Rishu, 2021b).

3.4.5.2.4 Multidex

Multidex OS Android je jedním z nejdůležitějších nástrojů při vývoji mobilních aplikací. Dex je formát spustitelného souboru, který funguje na Android virtuálním přístroji (též známý pod jménem Dalvik). Jestliže má být udělán .dex soubor odpovídající Dalvikově specifikaci, nesmí obsahovat více než 65 535 metod včetně všech knihoven projektu. Ačkoliv toto číslo se zdá nedosažitelné, v rozsáhlejších projektech je počet metod obvykle snadno přesažen. V takovém případě může vývojář využít Multidex knihovny, které rozloží .dex soubor aplikace do několik .dex souborů. Multidex komponenta zároveň poskytuje podporu kolektivních .dex souborů aplikace (Mishra Rishu, 2021b).

3.4.5.3 Jetpack: Behavior (Chování)

Jetpack Behavior obsahuje knihovny, které umožňují komunikaci s uživatelem skrz uživatelské rozhraní, včetně videa a zvuku. Obsahuje široké množství komponent jako například media, notifications, permissions, downloading, sharing nebo slices (Moore Kevin D., 2018).

3.4.5.3.1 DownloadManager

DownloadManager je systémová služba v Android, která přijímá žádosti uživatelů a provádí dlouhotrvající HTTP stahování na pozadí a následně soubor ukládá do předem specifikované destinace. Umožňuje tak potřebnou pomoc při problémech s připojením k internetu, problémy aplikace nebo rebooty systému. Jelikož DownloadManager je systémová služba, uživatel pouze iniciuje stahování a následně dostánu zprávu o dokončeném stahování (Ochieng Clement, 2021).

3.4.5.3.2 Notifications

Android Notifications jsou součástí Android od jeho začátků, ačkoliv neustále procházejí úpravami. Notifikace je zpráva, kterou Android zobrazí mimo aplikační uživatelské rozhraní a má sloužit jako připomenutí, komunikace s uživateli nebo jiná informace z aplikace. S postupem času přibyla možnost přidat do notifikací tlačítko nebo obrázek, stejně jako další součásti. Od verze Android 8.0 přibyla například možnost vypnout notifikace pro určitý kanál namísto celé aplikace nebo změnu pozadí notifikace (Android Developers, 2022d).

3.4.5.3.3 Permissions

Ukazuje, jak používat a žádat o povolení. Od Android 6.0 Marshmallow musí být o povolení požádáno a teprve po přidělení jsou dostupné některé elementy zařízení jako například kontakty, lokace nebo kamera. V nejnovějších verzích byly zveřejněna ActivityResult API, které zjednodušují žádání o povolení a poskytuje kontrakty i pro jednoduší situace, jako je například focení nebo otevíraní dokumentu (Muntenescu Florina, 2020).

3.4.5.3.4 Media

Jetpack poskytuje zpětně-kompatibilní API pro Android multimedia framework. Obsahuje media knihovny a třídy, jako například MediaPlayer a AudioManager, které umožňují přehrávání médií. Použitím MediaPlayer jako služby je možné získat kontrolu nad zvukem zařízení. Android podporuje různé formáty médií. Mimo jiné také obsahuje Playback knihovny nebo ExoPlayer knihovnu, kterou používá Google pro vlastní media přehrávače, jako například YouTube (Mishra Rishu, 2021a).

3.4.5.3.5 Slices

Jak už bylo dříve zmíněno, Slices knihovna je novinkou, která umožňuje vytvoření šablon uživatelského rozhraní pro sdílení dat skrz flexibilní layouty. Jedním z příkladu poskytnutých Googlem je aplikace týkající se počasí, která umožňuje zobrazit více informací závisejících na prostoru, který musí ukázat. V dnešní době jsou Slices používané především pomocí Google Search a Google Assistant a v uživatelských aplikacích jsou nejčastěji používány právě při použitích těchto služeb (Patel Saurabh, 2018).

3.4.5.4 Jetpack: UI

Většina UI knihoven v Jetpack je založených na existujícím kódu. Obsahují animace, fragmenty, palety, layouty, emoji, Android auto, Android wear a Android TV. Největší novinkou je EmojiCompat knihovna, které poskytuje nejnovější emoji a možnost jejich psaných zkratek. Většina těchto prvků je relativně dobře pochopitelná již ze jména. Za zmínku tedy stojí především Jetpack Compose, což je moderní sada nástrojů, sloužících ke stavbě přirozeného uživatelského rozhraní (Android Developers, 2022a).

3.4.5.4.1 Jetpack Compose

Jetpack Compose je framework, který zjednodušuje a urychluje vývoj uživatelského rozhraní pro Android. Slouží především jako řešení pro vysoké uživatelské nároky, které předchozí sady nástrojů nebyly schopné řešit.

Při psaní kódu jsou tvořeny moduly, které se skládají z několika jednotek. Pro pochopení Compose je potřeba pochopit dva základní pojmy týkajících se funkce těchto modelů. Prvním je párování (coupling), což je závislost mezi složkami v jiných modulech a určuje, jak jedna část modulu ovlivňuje část modulu druhého. Druhou cestou je soudržnost

(cohesion), která určuje vztahy mezi jednotkami v daném modulu a jak dobře spárované jsou. Základní principem programování udržitelného softwaru je snaha o minimalizaci párování a maximalizaci soudržnosti. V typickém případě je view model (jeden modul) programován v programovacím jazyku jako Java nebo Kotlin a rozložení (druhý modul) v XML. Jelikož oba jazyky jsou rozdílné, jsou oba moduly separované, i když jejich jednotky mohou být v blízkém páru. Compose umožňuje programování UI ve stejném jazyku (kotlin) a tím zvyšuje explicitu některých vazeb a soudržnost. Tento koncept jde proti obecnému přesvědčení, že logika by se neměla mixovat s uživatelským rozhraním. Reálně ale bude UI-related logika aplikace v obou případech.

Tento problém sám Compose neřeší, ale poskytuje nástroje, které usnadňují rozdělení. Největším z těchto nástrojů je Composable funkce, kterou lze volat a následně představuje uživatelské rozhraní v hierarchii. Pro složitější UI logiku lze použít if nebo for cykly pro bližší kontrolu toku dat (Richardson Leland, 2020).

Krom jiného Jetpack Compose také podporuje deklarativní uživatelské rozhraní, díky kterému není nutné sledovat předchozí stav, ale pouze specifikovat jaký má být stav nynější. Framework kontroluje změny mezi stavy, čímž zbavuje vývojáře povinnosti s řešením podobných problémů. Mezi další výhody patří možnost snazšího a efektivnějšího zapouzdření a rekompozice (opakování volání Composable funkce).

V celkovém měřítku Compose poskytuje moderní přístup k definování uživatelského rozhraní, který umožňuje efektivně třídit problémy (Moore Kevin D., 2018).

3.4.6 MVVM

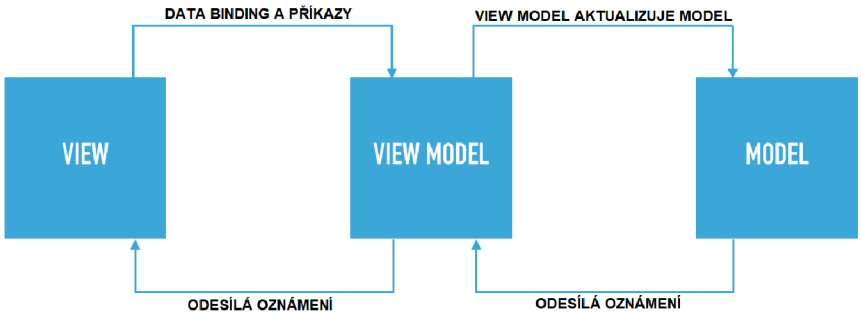
Model-View-Viewmodel (MVVM) je strukturální návrhový vzor, který umožňuje rychlou reakci na změny v designu. MVVM rozděluje objekty do tří rozdílných skupin.
V první řadě o Modely, které reprezentují aplikační data, se kterými je manipulováno. To jsou struktury nebo jednoduché třídy, které slouží jako implementace aplikačního doménového modelu, který obsahuje datový model s business i validační logikou. Například se tedy jedná o uložiště, business objekty, data transfer objekty a generované entity a proxy objekty (Microsoft Docs, 2012).

Druhou skupinou jsou Views, které zobrazují vizuální prvky a ovladače na obrazovce. Typicky se jedná o podtřídy UIView (objekt, který spravuje obsah obdélníkových oblastí na obrazovce). Views obecně jsou základní stavební kameny uživatelského rozhraní aplikace

a třída UIview definuje chování, které mají všechny Views společné (Kay Ryan Michael, 2020).

Poslední skupinou jsou View modely transformují informace modelu na hodnoty, které mohou být zobrazeny jako View. Obyčejně se jedná o třídy, aby mohly být posílány jako reference. V podstatě se tedy jedná o prostředníka mezi view a modelem, který zajišťuje předání view logiky (Strawn Jay, 2018).

Celkový návrhový vzor vypadá následovně:



Obrázek 7 - Model-View-Viewmodel vzor (Pickmans Alvaro Ortega, 2019)

Pro bližší porozumění je potřeba upřesnit, jak pracují jednotlivé komponenty. V nejvyšší vrstvě si je view vědoma view modelu a view model zaznamenává přítomnost modelu, ale ani model ani view model si není vědom vyšší vrstvy. View model izoluje view od tříd modelu a umožňuje modelu nezávislý vývoj na view. Komponenty jsou tedy rozděleny do párů, čímž je umožněna jejich výměna, změna vnitřní implementace komponent bez ovlivnění ostatních, jejich nezávislý běh a izolované unit testování.

Hlavní výhodou MVVM je zjednodušení pracovního procesu a komunikace mezi developery a designery. Především možnost pracovat ve stejnou chvíli na jednotlivých komponentách bez toho, aby byla ovlivněna práce na jiné komponentě a usnadňuje i urychluje práci na aplikaci. Mezi další výhody patří možnost testování View modelu a modelu bez nutnosti přístupu k view (Microsoft Docs, 2012).

3.4.7 Firebase

Firebase je vývojová platforma pro mobilní zařízení od Google s výkonnými funkcemi pro vývoj, správu a úpravu aplikací. V základu se jedná o sbírku nástrojů pro programátory sloužící k volnému vytváření a rozšiřování aplikací. Vývojáři používající tuto platformu

dostanou přístup ke službám, které by bez použití Firebase musely být vytvořeny jimi, což umožňuje vytvořit robustní aplikace s menším množstvím problémů (Rosencrance Linda, 2019). Firebase slouží k řešení tří problémů:

- Rychlé vytvoření aplikace
- Zveřejnění a monitorování aplikace
- Zapojení uživatelů

Hlavní výhody Google Firebase jsou databáze, autentizace, push zprávy, analytické nástroje, správa složek atd. Jelikož služby Firebase jsou hostované na cloudu, vývojáři mohou snadno používat on-demand škálování.

Mimo jiné lze jako alternativu za Firebase použít Back4App, Backendless, AWS Amplify, Parse nebo například Kinvey, které poskytují podobné možnosti, nicméně jsou méně podporovány a často nenabízejí stejnou škálu prvků (Batschinski George, 2022b).

3.4.7.1 Google Firebase Nástroje

Firebase má širokou škálu nástrojů, které lze kategorizovat pod do tří kategorií: build, release, a monitorování a zapojení uživatelů.

3.4.7.1.1 Build nástroje

- Databáze – Dvě databáze platformy Firebase jsou Cloud Firestore a Realtime Database, jež jsou obě užitečnými nástroji z hlediska výstavby moderních aplikací. Cloud Firestore je flexibilní, škálovatelná databáze pro mobilní, web a server vývojářství. Udržuje uživatelova data synchronizovaná skrz všechny aplikace díky listeners v reálném čase a nabízí off-line podporu pro mobil i web (Firebase Documentation, 2022a). Firebase Realtime Database je cloud-hostovaná databáze. Data jsou uschována v JSON a synchronizovaná s každým připojeným klientem (Firebase Documentation, 2022c). Realtime Database umožňuje aplikacím aktualizovat nejnovější verze a data a díky data perzistenci funguje i v off-line modu.
- Machine learning – Firebase Machine Learning je mobilní SDK, které využívá výhod Google machine learningu v Android a iOS aplikacích pomocí snadně použitelného balíčku (Firebase Documentation, 2022c). Firebase Machine Learning poskytuje řadu algoritmů, což umožňuje snadnou a rychlou implementaci ML do jakékoliv

aplikace. Firebase ML má tři možné varianty: API (předtrénované modely), Custom (implementace vlastního algoritmu) a AutoML (nejméně práce s nejvyšší účinností). https://blog.back4app.com/firebase-ml/

- Cloudové funkce – Firebase Cloud Functions je framework, který umožňuje vývojářům spouštění backend kódu pro odpovědi HTTPS a Firebase události. Pro skladování TypeScript a JavaScript kódu je použit Google cloud, který může být spuštěn v spravovaném prostředí (Batschinski George, 2022b).
- Autentizace – Firebase Autentizace poskytuje backend služby a snadno uživatelné SDKs sloužící k autentizaci uživatelů aplikace. Podporuje především autentizaci použití hesel, telefonních čísel a populárních identit poskytnutých Googlem, Facebookem nebo například Twitterem (Cid Joaquin, 2020).
- Cloudová komunikace – Firebase Cloud Messaging je cross-platformní komunikační služba sloužící k volné komunikaci. Vývojářům aplikace umožňuje především upozornit klientovu aplikaci na synchronizaci dat nebo emailů.
- Hosting – Firebase nabízí různé scalable a agile hosting nástroje pro mikro služby, webové aplikace a další typy obsahu. Obsah může být hostován v různých kategoriích a Firebase zajišťuje zabezpečení a ochranu proti některým chybám.
- Cloudové uložiště – Cloud Storage je služba pro ukládání zdrojů vývoje aplikace, včetně objektů. Uživatelé mají přístup ke Google stahovací a nahrávací ochraně, která je s touto službou spojena a je ideální pro ukládání medií a uživatelského obsahu (Batschinski George, 2022b).
- Emulátory – Aby vývojáři mohli integrovat a testovat všechny výše zmíněné nástroje, poskytuje Firebase Local Emulator Suite. Ten umožňuje testovat kód bez zbytečných nákladů. Firebase Local Emulator Suite nyní poskytuje emulátory pro autentizaci, cloudové funkce, databáze (RTDB), hosting a Google Cloud Pub/Sub. Zároveň je poskytnuto uživatelské rozhraní odpovídající Firebase konzoli.

3.4.7.1.2 Firebase Release & Monitor

Firebase Release & Monitor nástroje slouží k přípravě vývojářů na zveřejnění aplikace. Přináší řadu testovacích, analytických a distribučních nástrojů, které mají zajistit dostupné používání aplikace pro uživatele (Batschinski George, 2022b).

- Crashlytics – Firebase Crashlytics je nástroj sloužící k hlášení nefunkčnosti aplikace, který slouží především k prioritizaci a následnému opravení největších problémů aplikace s ohledem na důsledek na uživatele (Firebase Documentation, 2022b).
- Google Analytics pro Firebase – Google Analytics poskytují Firebase monitorování 500 různých událostí v aplikaci. Google Analytics SDK automaticky zachytává různé klíčové události a vlastnosti uživatelů a mimo jiné umožňuje definovat vlastní měřitelné události, které mají být chytané a jsou relevantní programovanou aplikaci (Firebase Documentation, 2022e).
- Remote Config – Remote Config poskytuje viditelnost a detailní kontrolu nad chováním aplikace a jejím vzhledem. Tím pádem je možné snadně měnit aplikaci pomocí upravování konfigurace ve Firebase konzoli. To znamená, že lze dynamicky zapínat a vypínat nástroje a ovládat experimenty, a to vše bez nastavení komplexní infrastruktury nebo vytvoření nové verze (Firebase Documentation, 2022d).
- Performance Monitoring – Firebase Performance Monitoring umožňuje dohlížet na aplikaci během změn konfigurace nebo zveřejňování změn. Zároveň také dává kontrolu nad daty zobrazujícími výkon aplikace s možností upravitelné tabulky, ve které lze zpřehlednit nejdůležitější metriky (Abdelhadi Ali, 2017).
- Test Lab – Firebase Test Lab je cloudová infrastruktura pro testování aplikací. Umožňuje uživatelům testovat jejich iOS a Android aplikace s různými operacemi a konfiguracemi. Uživatelům jsou zobrazeny výsledky testů, screenshoty, zápisy a videa ve Firebase konzoli (Batschinski George, 2022b).
- App Distribution – V neposlední řadě je nástrojem Firebase App Distribution. Ta umožňuje holistický pohled do beta testovacího programu, který poskytuje cennou zpětnou vazbu před uvolněním do produkce (Firebase Documentation, 2022c).

3.4.8 Další moderní technologie

V této sekci budou krátce popsány další moderní technologie a přístupy k vývoji mobilních aplikací na OS Android, které pro tuto práci vybrány nebyly a bude popsáno, proč byly upřednostněny dříve zmíněné části.

- **MotionLayout** – MotionLayout umožňuje správu pohybu a widget animací v aplikaci. Jedná se především o složité pohyby v aplikaci. Jedná se o součást

ConstraintLayoutu, nicméně pro účely hudební aplikace se jedná o příliš sofistikované řešení, protože žádné složité pohyby na více obrazovkách nebudou použity (Hidden Brains Blog, 2022).

- **MVP vzory** – Stejně jako MVVM, i Model-View-Presenter je odnoží Model-View-Controller. MVVM byl vybrán především kvůli vyšší kompatibilitě. Ačkoliv pomocí MVP lze zvýšit výkon aplikace, výpočet hudební aplikace nebude komplexní a byla by pouze zvýšena komplexita aplikace. MVVM obecně obsahuje méně uživatelských rozhraní, samostatné prvky fungují nezávisle, a především funguje lépe na aplikacích s OS Android (Pedemkar Priya, 2020).
- **Machine learning, umělá inteligence a rozšířená realita** – Ačkoliv některé z těchto technologií by využití v hudební aplikaci mít mohli, komplexita aplikace by se několikanásobně zvýšila a vývoj multifunkční aplikace by přesáhl rozsah diplomové práce.
- **Ostatní cloud služby** – Cloudových služeb pro ukládání dat je k dispozici několik. Jedná se například o AWS, MongoDB, Azure, Xamarin nebo Laravel. Firebase je vlastněná Googlem, což znamená, že neustále přibývají její možnosti a poskytuje lepší podporu pro vývoj na OS Android (Batschinski George, 2022a).
- **RxKotlin** – RxKotlin je vpodstatě jen možnost použití RxJava v Kotlinu. Coroutines jsou více obecný koncept, lze je použít ve více situacích. Největší výhodou je že coroutines jsou jednoznačně lepší při práci se zdroji aplikace, mají jednoduší API a vyvarují se některých chyb, které RxKotlin měl, jako například problémy s back-pressure nebo únikům paměti. Kotlin kvůli strukturované konkurenci umožňuje snazší management životního cyklu konkurenční části kódu (Lew Dan, 2021).
- **SQLite databáze** – SQLite neposkytuje některou funkcionalitu, kterou Room ano. Jedná se například o zobrazování chyb (SQLite chyby nezobrazuje, je nutné debugovat). Nevýhodami SQLite databáze jsou také náročné změny a mapování dat k objektům. V neposlední řadě také nekompatibilita s novějšími technologiemi (Kuntal Sunil, 2017).

3.5 Analýza trhu

Aplikací tykajících se hraní na kytaru je na mobilní zařízení mnoho. Především se jedná o aplikace pomáhajících při ladění kytary nebo o virtuální zpěvníky, které obsahují písně, akordy a vybrnkávání. Na takové aplikace primárně analýza trhu zaměřena nebude, protože se nejedná přímo o aplikace pomáhající při hře na kytaru, ale především o pomůcky pro hudebníky, které už jisté základy mají. Zaměření bude především na komplexní aplikace, které pomáhají při hře na kytaru začátečníkům i pokročilým v různých formách a budou preferovány populární aplikace (s největším počtem stažení). Většina analyzovaných aplikacích je dostupných na Google Play, digitální distribuční službou spravovanou a vyvíjenou Googlem.

U vybraných aplikací budou popsány jejich obecné parametry, jednotlivé funkce a bude-li volně dostupný i kód, tak i použité technologie.

3.5.1 REAL GUITAR: Virtuální kytara zdarma

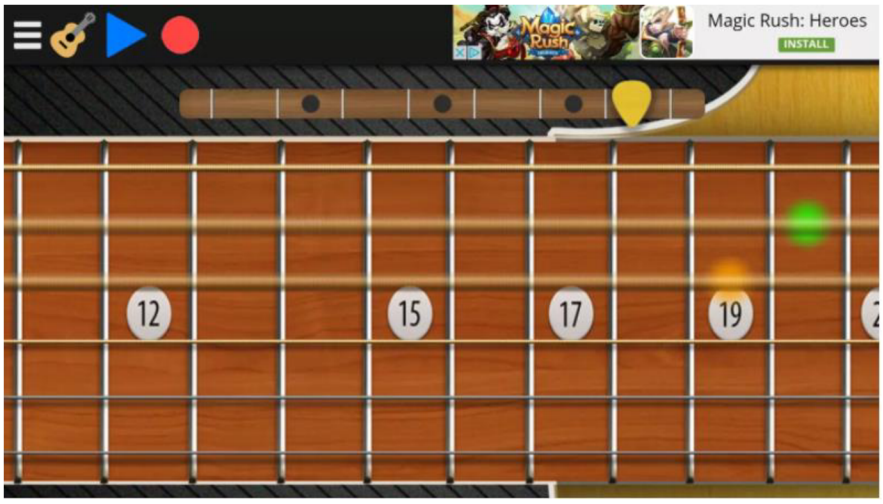
Nejvíce vyhledávanou a jednou z nejlépe hodnocených je aplikace Real Guitar. Aplikace Real Guitar je hra simulující kytaru na obrazovce telefonu nebo tabletu a dotykem prstu na obrazovce (virtuální kytaře) zahraje daný tón (viz. Obrázek 8). Mimo základní hrací plochy také obsahuje 40 lekcí, které slouží k pomoci při základech a umožňují vyzkoušet různé hudební styly. Zároveň také obsahuje katalog 1500 akordů, které umožňují doprovázet skladby.

Jako hlavní přednosti aplikace Real Guitar zmiňuje zvuk bez zpoždění, kvalita zvuku jako ve studiu, přizpůsobitelná sekvence akordů, 5 druhů kytary pro různý zvuk, 3 režimy přehrávání, 16 smyček pro synchronní hraní, více než 1500 akordů, věrné kytarové zvuky, možnost záznamu, exportování nahrávek v MP3, přizpůsobení obrazovce, jedná se o volně dostupnou aplikaci, která je snadno použitelná. Hudební aplikace je zdarma, nabízí ale i prémiovou verzi, která zbaví aplikaci reklam a odemkne některé rozšířené funkce (například metronom).

Základní balíček Real Guitar má pouze 63Mb a je (stále) vyvíjen společností Kolb Apps. V dnešní době byla tato aplikace instalována více než 10 000 000 uživateli a vyžaduje Android 5.0 vyšší. Je také dostupná ve 44 jazycích. (Kolb Apps, 2021)

Hlavní funkcionalitou aplikace jsou virtuální hmatníky kytary se strunami. Při doteku dané struny na hmatníku se ozve odpovídající tón. V aplikaci je možno hrát více prsty v jednu chvíli, a tudíž kombinovat jednotlivé zvuky i akordy. Místo hmatníku lze také nastavit akordy, které při hře přes struny budou hrát (v jedné chvíli jich může být více). V aplikaci je na výběr zvuk akustické kytary, čistý zvuk, zkreslený zvuk, nylonové struny a křoupavý zvuk. Na těch lze hrát v jednom ze tří připravených módu, které jsou normální akordy, jednoduché akordy a sólové hraní.

K výukovým možnostem patří možnost kytarových lekcí, které nejprve zahrají píseň a následně na hmatnících ukazují, které struny je nutné zmáčknout pro zopakování dané písně. Další výukovou možností je hraní rytmických smyček, přes které lze hrát melodii. V aplikaci je možné i nahrát vlastní písně a na virtuální kytaře je následně zopakovat. V neposlední řadě je možné písně nahrávat a následně je spouštět a metronom, který pomáhá s udržením tempa po čas hraní.



Obrázek 8 - REAL GUITAR hra (Kolb Apps, 2021)

3.5.2 Simply Guitar by JoyTunes

Simply Guitar je aplikace na Android, která poslouchá uživatelovo hraní na kytaru a pomáhá s hrou. Poskytuje step-by-step video tutoriály udělané předními učiteli hry na kytaru a při opakování přijímá zpětnou vazbu (zvukovou stopu) při hraní slavných písní. Následně poskytuje okamžitý feedback o chybách a nedostatcích při hraní. Pro užívání aplikace nejsou nutné žádné předchozí zkušenosti s hraním na kytaru, pouze člověk musí

kytaru vlastnit. Uživatel se může pohybovat libovolně mezi funkcemi a vybírat si sám, co se chce učit (na výběr je například možnost mezi linkami hlavní nebo doprovodné kytary).

Základní balíček Simply Guitar by JoyTunes je zdarma, avšak jeho funkcionalita je bez zaplacení prémiové verze značně omezená a po určité době bez ní nejde pokračovat. Aplikace má 517Mb a využívá Android 6.0 a více. Aplikaci je možné nainstalovat jak na chytrý telefon, tak na tablet, a to jak na OS Android, tak na iOS (JoyTunes, 2022).

Hlavní funkcí aplikace je kontrola hry na kytaru uživatele sledovaná přímo aplikací. Mobilní zařízení by měl být blízko kytaře, aby bylo snímání přesné. Následně aplikace naviguje uživatele a přes začátky, jako je například jak držet kytaru nebo jak ladit struny, se uživatel dostane až k hraní riffů, akordů, vybrnkávání i změnu pozic. To vše podpořené video tutoriály, ve kterých je vysvětlována podstata učeného i grafické zobrazení toho, jak se má uživatel danou věc naučit. Na konci každé lekce si pomocí známé písně uživatel ověří, jestli danou látku ovládl.

3.5.3 Yousician

Stejně jako Simply Guitar, tak i Yousician snímá zvuk kytary i ostatních nástrojů a poskytuje okamžitý feedback na přesnost a načasování. Stejně také poskytuje pomoc všem úrovním hudebníků, ať už začátečníkům, tak i profesionálům či učitelům hudby. Poskytuje step-by-step video lekce v herní formě. Pro užívání vlastník potřebuje vlastní kytaru. Lekce v Yousician jsou napsané hudebními učiteli a je v ní přes 1500 „misí“ a cvičení.

Aplikace Yousician je zdarma, ale stejně jako předchozí aplikace, její funkcionalita je bez prémiové verze omezená. Premium verze umožňuje hraní bez reklam a bez časového omezení. Yousician má 60Mb a vyžaduje Android 4.4 a vyšší. Aplikace funguje na Android, iOS i jako desktop aplikace a všechny její verze jsou stále aktualizovány (Yousician, 2022).

Yousician slouží především ke sledování hry na hudební nástroj uživatele. Aplikace je kombinací vysvětlujících tutoriálů a sledování virtuálního hmatníku kytary, které potvrzuje přesnost zahraných tónů a rytmu a oznamuje možnosti zlepšení. Stejně jako Simply Guitar, i Yousician učí detailně i základy, jako je držení kytary nebo ladění. Kromě praktických lekcí poskytuje i základy teorie. Většina výukového materiálu je prolnuta s populárními písněmi.

3.5.4 The Gibson App

Aplikace Gibson je aplikace nabízející step-by-step lekce poskytující optimální učební prostředky pro kytaristy všech úrovní. Poskytuje průvodce, který umožňuje hru od úplných základů a tím poskytuje ideální podmínky pro učení. Lekce jsou interaktivní a stejně jako předchozí aplikace, i Gibson poslouchá hraní kytary. V aplikaci je možné vybrat si z několika hudebních stylů, které jsou zprostředkovány experty v hře na kytaru. Pro plné užití aplikace je nutní mít vlastní elektrickou nebo akustickou kytaru a sluchátka.

Aplikace Gibson stojí necelých 15 euro na měsíc nebo 90 za rok (280 nebo 1700 kč). Před zakoupením aplikace je také možné vyzkoušet volnou zkušební verzi. Velikost aplikace je 92Mb a do dnešní doby je stále vyvíjena. I proto ve verzi 1.7.3 vyžaduje Android 6.0 a vyšší. Aplikaci lze kromě OS Android stáhnout také na iOS (Yousician, 2022).

The Gibson App nabízí možnost naučit se hrát slavné kytarové písně od autorů jako Eric Clepton, Santana nebo The Beatles. Systém poskytuje akordy, načež dává přímou zpětnou vazbu po každém tónu, čímž přidává lekcím interaktivitu. Tuto technologii nazývá Gibson Audio Augmented Reality. Každá píseň je přizpůsobena úrovni, ve které se uživatel nachází a tím poskytuje optimální výzvu. Aplikace dále nabízí vestavěnou ladičku pro standardní i alternativní ladění kytary, včetně video tutoriálu, které při ladění pomáhají. Mimo jiné také aplikace poskytuje osobní video konzultaci s profesionálním kytarovým technikem společnosti Gibson, který vysvětlí, jak kytaru ladit i udržovat.

3.5.5 Další aplikace

Další aplikace, které již mají nižší počet stahování, popřípadě jejich funkcionalita přímo neodpovídá výukovým lekcím, (Breathnach Cillian, 2021) jsou:

- Ultimate Guitar: Chords & Tabs – Podle počtu uživatelů na Google Play se jedná o jednu z nejvíce používaných aplikací. Aplikace Ultimate Guitar obsahuje širokou škálu písní, akordů a výukových materiálu. Nevlastní ale velké množství výukových možností.
- Justin Guitar Lessons & Songs: Learn How to Play – Poskytuje snadné kytarové lekce především pro začátečníky. Aplikace je založena na YouTube kanálu Justin Guitar, ze kterého také většinu lekcí čerpá. Jedná se o více osobnější přístup, než má

většina ostatních aplikací, ale aplikace je velmi omezená, a to především počtem lekcí.

- Guitar Lessons by Guitar Tricks – Výuka založená na step-by-step video tutoriálech populárních písní. Poskytuje výukové materiály pro začátečníky i pokročilé. Mimo špičkové instruktory nabízí například také kytarové taby ke stažení, možnost najít akordy, ladičku nebo možnost naučit se pohyb na hmatníku.

3.5.6 Závěr analýzy trhu

Aplikací sloužících k výuce hry na kytaru je velké množství a liší se funkcionalitou od pomocných aplikací při hře až k samostatným aplikacím nahrazujícím kytaru. Většina z nich vyžaduje vlastnictví kytary nebo jiného hudebního nástroje, tudíž užití je omezenou pouze na momenty, kdy uživatel hudební nástroj vlastní a je v situaci, kdy ho může používat. Takových situací může být omezené množství, protože jsou velmi limitovány okolnostmi, a tudíž daná aplikace nemusí být snadno použitelná a pravidelně užívaná. Mezi takové aplikace se z analyzovaných řadí například Simply guitar (viz. kapitola 3.5.2), Yousician (viz. kapitola 3.5.3) nebo The Gibson app (viz. kapitola 3.5.4). Všechny tyto aplikace slouží jako studijní pomůcky pro hraní, ale všechny mají omezené využití spojené s vlastnictvím kytary.

Dalším druhem pomocných hudebních aplikací je simulátor kytar (nebo dalších hudebních nástrojů). Ačkoliv v tomto případě se jedná o druh aplikací, které nevyžadují hudební nástroj pro funkcionalitu, a tudíž jsou využívatelé samostatně, nejedná se o aplikace, které přímo pomůžou uživateli se hrou. Ukázkou takové aplikace je REAL GUITAR (viz. kapitola 3.5.1), která nám na obrazovce ukazuje pražce se strunami, které po doteku zahrají odpovídající tón. Samotné užití aplikace ale nijak nepřipomíná hru na kytaru a jako studijní pomůcka aplikace využitelná příliš není. Jedná se tedy spíše o nějakou hru, která simuluje funkcionalitu kytary a nepřímo pomáhá vysvětlit i některé základy hraní i kytarové teorie.

V neposlední řadě se jedná aplikace pomáhají uživatelům ve formě tutoriálů. Takovými aplikacemi jsou například Justin Guitar Lessons & Songs nebo Guitar Lessons by Guitar Tricks (viz. kapitola 3.5.5). Stejně jako první typ, i tyto bezpodmínečně vyžadují vlastnictví hudebního nástroje. V tomto případě se ale jedná vlastně čistě o „Do it yourself“ cestu, při které máme pouze výukové materiály, ale žádnou zpětnou vazbu. Tato cesta v některých případech neposkytuje uživatelům motivaci v pokračování užívání aplikace,

protože podobným způsobem se lze učit hře na kytaru i z jiných zdrojů, jako například specializované weby nebo video tutoriály na Youtube. Obě alternativní cesty jsou diskutabilně snáze dostupné a většinově i užitelné, tudíž motivace stáhnout a používat specializovanou aplikaci může být snížená.

Žádná z uvedených aplikací neměla dostupný kód, tudíž není možné vidět použitý jazyk ani technologie. Jelikož je většina těchto aplikací na trhu přibližně od roku 2014, lze odhadovat, že nejnovější technologie nebudou použity. Ačkoliv všechny aplikace mají pravidelné updaty, dá se očekávat, že programovací jazyk a některé základní koncepty se v nich nezměnily, protože drastické změny kódu jsou pro společnosti velmi nákladné. Výjimkou jsou aplikace The Gibson app (viz. kapitola 3.5.4) a Simply guitar (viz. kapitola 3.5.2), které jsou vytvořeny po roce 2020 a jejich kód tedy může být za použití novějších technologií naprogramován.

Vedlejší nevýhodou je, že většina aplikací pomáhajících při výuce hry na kytaru je v nějaké formě zpoplatněna. Ačkoliv většina aplikací je označena jako freeware, jejich funkcionalita je omezená, pokud uživatel nemá předplacené prémiové členství. Aplikace také zobrazují velké množství propagačních materiálů, které způsobují horší user experience a znepříjemňují proces učení.

Z výsledků analýzy lze vidět, že na trhu není bezplatná aplikace, která by byla plně soběstačná, tedy nevyžadovala použití hudebního nástroje, a zároveň sloužila k výukovým účelům. Praktická část diplomové práce bude zaměřena na vývoj aplikace pomocí moderních přístupů k vývoji mobilních aplikací na OS Android, která se těchto nedostatků vyvaruje. Hlavní záměr bude vytvoření aplikace, která bude užitečná pro učení se hraní na kytaru, bude soběstačná a zkušenosti z jejího užití budou použitelné při samotné hře na kytaru. Zároveň bude poskytovat prostředky pro učení se hry na kytaru, jestliže již uživatel hudební nástroj vlastní. Tato aplikace bude vytvořena za použití moderních technologií, které zpříjemní user experience i uživatelské rozhraní.

4 Vývoj hudební aplikace

Po analýze možností, které jsou odpovídající formě vývoje aplikace pro diplomovou práci, byla pro tento projekt byla přijata zjednodušená waterfall (vodopád) metodologie softwarového inženýrství. V této metodologii je celý proces členěn do rozdělených fází a výstup jedné fáze je vstupem další. Ačkoliv v nynější praxi jsou populárnější agilní metodologie, v případě aplikace vyvíjené jedním programátorem na relativně krátkém a přesně specifikovaném časovém úseku by iterativní přístup neměl smysl. V případě vývoje aplikace pro diplomovou práci je sekvenční provádění jednotlivých fází ideální a znamená, že posun do další fáze proběhne pouze po ukončení fáze předchozí, a tudíž budou již všechny potřebné detaily specifikovány.

Mimo to je nutné dodat, že některé fáze (nebo jejich součásti) životního cyklu vývoje mobilních aplikací jsou vynechané nebo relativně strohé. Vzhledem k podstatě diplomové práce jsou některé součásti životního cyklu nevyužité a jejich detailní popis nemá v rozsahu diplomové práce smysl. Jako příklad lze vzít nasazení a údržbu, které jsou pouze zmíněny, nicméně nevyužity, neboť jako nasazení se v tomto případě bere odevzdání diplomové práce a následná údržba již obsažena touto prací není.

4.1 Počáteční fáze

4.1.1 Definice cíle

Pro specifikace cíle bylo individuálně vytvořeno několik prvotních možností a pomocí analýzy proveditelnosti a trhu byl vybráno vytvoření čistě edukativní aplikace na mobilní operační systém Android, která slouží k pomoci při hře na kytaru bez nutnosti použití hudebního nástroje a dalších pomůcek. Cílem je vytvořit nezávislou pomůcku, jejímž používáním uživatel získá schopnosti, které bude moct přenést do používání hudebního nástroje. Aplikace bude zaměřena na tři základy při hře na kytaru, kterými jsou noty, akordy a rytmus. Na základě těchto předpokladů budou vytvořena tři cvičení, jedno sloužící k výcviku not a vybrnkávání, druhé k procvičení akordů a třetí sloužící k naučení držení rytmu.

Z definice zadání diplomové práce také vyplívá, že aplikace bude naprogramována za pomocí moderních technologií a přístupů k vývoji na OS Android, což také bude zohledněno v jednotlivých fázích vývoje.

Tento cíl bude splněn do konce února v kvalitě odpovídající standardům diplomové práce.

4.1.2 Cílová skupina

Cílovou skupinou hudební aplikace budou uživatelé, kteří se chtějí naučit hře na kytaru a zároveň vlastní chytrý telefon s operačním systémem Android. Podle dostupných zdrojů je 72 % kytaristů v posledních dvou letech ve věku 13-34 let (Childers Chad, 2021). Ačkoliv pro aplikaci věkové omezení platit nebude a uživatelem by mohl být kdokoliv s dostupnými prostředky, největší úspěch lze očekávat právě ve skupině 13-34 let.

4.1.2.1 Persony

Persony definují skupinu archetypního uživatele aplikace. Na základě popisu cílové skupiny jsou vytvořeny tři persony (každá popisující jinou skupinu) a jedna antipersona, která slouží cílové skupiny, na kterou aplikace nemíří.

4.1.2.1.1 Jan Bosák Jr. - Persona

Obrázek 9 – Persona Jan Bosák (@cookie_studio, 2022)

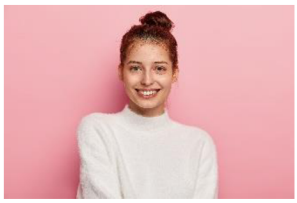
Jan Bosák je muž, je mu 23 let, bydlí v Praze a studuje na České zemědělské univerzitě. Žije s rodiči a ve volném čase se věnuje sportu.

Jeho obyčejný den začíná okolo osmé ráno, kdy vstane a buď jde do školy nebo na brigádu, kde většinou stráví celý den. Večer se pak věnuje trénování florbalu nebo tráví čas s kamarády

Poslední dobou hledá nové koníčky a volnočasové aktivity, kterými by pokryl volný čas, který nevěnuje jiným aktivitám. Už delší dobu má chuť se zaměřit na hudbu, protože jí často poslouchá a je kreativní. Jelikož nejvíce poslouchá rockovou hudbu, rád by vyzkoušel nástroj, který bude mít v tomto stylu využití. Proto přemýšlí o kytaře. Bohužel nemá žádné základy a potřebuje tak pomoci s učením. Jelikož jeho příjem je založen pouze na brigádách ke škole a ví, že kytara není levná záležitost, nežli se do samotné hry pustí, rád

by vyzkoušel, jestli by mu kytara vyhovovala. Proto si stáhl hudební aplikaci, na které zkouší hru na kytaru a zjišťuje, jestli má v hudebním světě budoucnost. Zároveň se také učí a jestliže se rozhodne začít, tak bude mít základky.

4.1.2.1.2 Irina Popescu – Persona



Obrázek 10 – Persona Irina Popescu (@wayhomestudio, 2022)

Irina Popescu je žena, je jí 16 let, studuje střední školu a bydlí v Teplicích. Žije s rodiči a volný čas tráví primárně s přáteli a zvířaty.

Její obyčejný den začíná v 6:30 ráno, kdy vstává, připravuje se a jde do školy. Její škola většinou končí okolo 15:00 a následně jde ven s přáteli. Mimo to má každý druhý den lekce hry na kytaru vedené učitelem. Večer se většinou věnuje domácím úkolům nebo se dívá na seriály.

Protože začala hrát na hudební nástroje v pozdním věku, snaží se dohnat svoje nedostatky. Jelikož však doma netráví příliš času, nemůže tréninku věnovat tolik, kolik potřebuje. Proto jí byla učitelem doporučena hudební aplikace, ve které může kontrolovat svůj posun a zároveň nepotřebuje trávit tolik času doma.

4.1.2.1.3 Lukáš Hromada – Persona

Obrázek 11 – Persona Lukáš Hromada (Biazar Reza, 2018)

Lukáš Hromada je muž, je mu 45 let a pracuje v administrativě aukční síně v Prostějově, kde také žije. Je ženatý a má dvě dcery ve věku 15 a 12 let.

Jeho obyčejný den začíná v 7:00, kdy vstává, snídá a jde do práce. Po práci jde většinou domů, kde tráví čas s rodinou. O víkendu chodí s rodinou na pěší túry po České republice. Ve volném čase sleduje politickou situaci a zajímá se o auta.

Jelikož ho už od dětství zajímalo cestování a pěší výlety s kamarády, naučil se v útlém věku trochu hrát na kytaru, což využíval hlavně při hraní u ohně. V poslední době již na kytaru tolik času nemá, ale jednou za čas stále hraje. Jelikož je mu líto, že se ve hře na kytaru nijak nerozvíjí, rozhodl se používat hudební aplikaci, kterou mu doporučila jeho dcera. V práci má často volný čas, který nemůže produktivně využít, a proto hudební aplikaci používá právě v této situaci.

4.1.2.1.4 Radana Valentová – Antipersona

Obrázek 12 – Antipersona Radana Valentová (@pressphoto, 2022)

Radana Valentová je žena, je jí 70 let, důchodkyně a žije v Líbeznicích u Prahy. Je vdaná a má dva syny, dceru a 5 vnoučat.

Její den začíná v 6:00, kdy vstane, nasnídá se a dívá se na televizi. Následně se věnuje zahrádkaření a vaření. Večer se dívá na svůj oblíbený seriál Ulice a následně jde relativně brzo spát.

Jako dítě hrávala na kytaru, ale od svého koníčku upustila a s hudbou ztratila kontakt. Jelikož jí to celý život bylo líto, zmínila se o tom před rodinou a ta se rozhodla jí k hudbě opět posunout. Jelikož paní Valentová k narozeninám dostala nový chytrý telefon, se kterým se ještě snaží naučit pracovat, jeden vnuk jí nainstaloval hudební aplikaci, která nejen přiblíží hru na kytaru, zároveň i pomůže citlivě ovládat práci s mobilním telefonem. Paní Valentová aplikaci téměř okamžitě zavrhla, protože jí nepřipomíná hru na kytaru, kterou v dětství měla ráda a ovládání nového mobilu bylo náročné i bez aplikace.

4.1.3 Řízení požadavků

Analýza požadavků cíli na specifikaci požadavků spojených s hudební aplikací a slouží k poskytnutí nutných informací pro návrh a architekturu softwaru, stejně jako pro implementaci a ostatní fáze projektu. Ve standardním firemním prostředí by řízení požadavků mělo být vedeno všemi stakeholdery aplikace a relativně velké slovo by měl mít klient. Jelikož v případě diplomové je pozice většiny stakeholderů prázdná, řízení požadavků je založeno na materiálech o vývoji mobilních aplikací a základech hry na kytaru, které by měly reflektovat úroveň, kterou bude uživatel procvičovat pomocí hudební aplikace.

Na základě analýzy bylo určeno, že nejdůležitějšími praktickými prvky jsou vybrnkávání, akordy, rytmus a taby.

Pro určení priority jednotlivých požadavků byla přijata MoSCoW technika, která slouží k rozdělení požadavků do čtyř skupin. Těmi jsou:

- MUST have: Nutné části, které jsou pro aplikaci povinné a bez nichž aplikace nemůže fungovat

- SHOULD have: Důležité části, které nejsou povinné, ale jejich přidání má znatelnou hodnotu
- COULD have: Části, jejichž vynechání bude mít na aplikaci nízký vliv, ale jejich přidání bude do určité míry hodnotné.
- WON'T have: Části, které na aplikaci mají velice nízký/žádný vliv.(Clegg Dai, 2020)

4.1.3.1 Funkční požadavky

Funkční požadavky mají několik nutných parametrů. ID je specifikováno pomocí názvu kapitoly, aby byla zvýšena přehlednost a jedná se o jedinečný specifikátor funkčního požadavku. Prvním parametrem je popis, který krátce shrnuje, co musí aplikace umožňovat. Uživatelem je vždy uživatel aplikace, a proto nebude zmíněn. Následně bude zmíněna složitost a priorita. Priorita je nutnost uskutečnění požadavku určená MoSCoW technikou.

4.1.3.1.1 Přihlášení

Popis: Aplikace umožňuje uživateli přihlášení pomocí uživatelského jména/emailové adresy a hesla, které je možné uskutečnit, jestliže je uživatel vlastníkem účtu.

Složitost: 5/10

Priorita: Must have

4.1.3.1.2 Registrace

Popis: Aplikace umožňuje uživateli vytvoření osobního účtu pomocí jména/emailové adresy a hesla, které uživatel může uskutečnit, jestliže ještě účet nemá.

Složitost: 5/10

Priorita: Must have

4.1.3.1.3 Obnova hesla

Popis: Aplikace umožňuje uživateli změnu hesla na základě emailové adresy.

Složitost: 4/10

Priorita: Must have

4.1.3.1.4 Odhlášení

Popis: Aplikace umožňuje přihlášenému uživateli odhlášení, po kterém se bude moci opět přihlásit.

Složitost: 2/10

Priorita: Must have

4.1.3.1.5 Změna osobních informací

Popis: Aplikace umožňuje přihlášenému uživateli zprostředkování změny osobních informací.

Složitost: 5/10

Priorita: Can have

4.1.3.1.6 Zapnutí offline režimu

Popis: Aplikace umožňuje uživateli zapnutí offline režimu.

Složitost: 5/10

Priorita: Must have

4.1.3.1.7 Zobrazení informací (o účtu, o aplikace, o hrách – pravidla)

Popis: Aplikace umožňuje uživateli zobrazení doplňujících informací pro pochopení aplikace.

Složitost: 1/10

Priorita: Should have

4.1.3.1.8 Provedení lekce

Popis: Aplikace umožňuje uživateli spuštění a ukončení jedné z lekcí, kterou procvičuje hru na kytaru.

Složitost: 9/10

Priorita: Must have

4.1.3.1.9 Zobrazení výsledků

Popis: Aplikace umožňuje uživateli zobrazení výsledků ze cvičení, které umožňují náhled do historie lekcí.

Složitost: 3/10

Priorita: Should have

4.1.3.2 Nefunkční požadavky

Nefunkční požadavky budou obsahovat tři informace. Kromě priority, která funguje na stejném principu jako u funkčních požadavků, se bude jednat o popis, který krátce vysvětluje vztah aplikace k danému nefunkčnímu požadavku a rizika neboli problémy, které při plnění požadavku mohou nastat.

4.1.3.2.1 Použitelnost

Popis: Jelikož aplikace bude používána především cílovou skupinou, která již má s používáním mobilních aplikací zkušenosti, použitelnost musí být v dostatečné formě, nemusí se ovšem jednat o perfektní provedení. Nicméně při hrách bude preciznost provedení nutná. Uživatelovi aplikace by nemělo vybrání možnosti trvat déle než 5 sekund.

Rizika: Pro některé uživatele nemusí být uživatelské rozhraní intuitivní (především pro uživatele mimo hlavní cílovou skupinu). Cvičení nebudou mít příliš přísné UI a budou složité na hraní.

Priorita: Must have

4.1.3.2.2 Výkon

Popis: Jelikož hlavní součást aplikace, kterou jsou cvičení, bude na výkonu závislá (jestliže nebude fungovat rychle přepínání během cvičení, uživatel se bude potýkat s prodlevami, které budou snižovat chuť používat aplikaci), jedná se o požadavek s vysokou prioritou. Přepínání ve cvičení by nemělo trvat déle než 1 sekundu. Přepínání mezi obrazovkami by nemělo trvat déle než 1 sekundu.

Rizika: Aplikace může mít problémy s nahráváním dat při hraní her a s následným ukládáním.

Priorita: Must have

4.1.3.2.3 Zabezpečení a soukromí

Popis: Mimo první vrstvu zabezpečení (emailová adresa a heslo) by žádná zadaná data neměla zasahovat do uživatelského soukromí. Systém by neměl mít možnost zjistit a přísně tak dodržovat GDPR. Zaměření zabezpečení bude tedy především na ochranu přihlašovacích údajů. Jiné informace při případných únikách dat neposkytnout žádné podstatné informace.

Rizika: Zabezpečení přihlašovacích údajů.

Priorita: Must have

4.1.3.2.4 Uložení dat

Popis: Aplikace musí ukládat všechna data do databáze, včetně hodnocení ze cvičení uživatelů. Aplikace by měla mít k dispozici offline i online databázi a zpřístupnit tím oba režimy používání. Ukládání do databáze by nemělo zdržovat běh aplikace (mělo by trvat pod 0,5 sekundy).

Rizika: Jestliže cvičení přestane fungovat uprostřed běhu, nasbíraná data nebudou uložena a zobrazena v uživatelových skóre.

Priorita: Should have

4.1.3.2.5 Škálovatelnost

Popis: Systém by měl být připraven na potenciální nárůst uživatelů. Z podstaty aplikace psané k diplomové práci tento požadavek není nutný, nicméně jestliže má aplikace užívat moderní technologie, dosaženo by ho být mělo.

Rizika: Žádné.

Priorita: Could have

4.1.3.2.6 Jazyková podpora

Popis: Aplikace bude použitelná ve více jazykových formách, především v češtině a angličtině. V budoucnu jazykových forem přibude víc a aplikace by tak měla být připravená na snadnou implementaci těchto změn.

Rizika: Některé grafické prvky budou potřebovat obměnu kvůli různé délce slov v jazycích.

Priorita: Won't have

4.1.3.3 Případy užití (Use Case)

Případy užití jsou metodologie, která pomáhá určit a organizovat systémové požadavky. Skládá se ze sekvence interakcí mezi systémem a uživateli za snahou dosáhnutí určitého cíle. Případy užití mají vytvořeny tabulky s několika parametry. Prvním parametrem je unikátní ID. Následně jsou popsány předpoklady, které musí uživatel splnit, aby mohl daný případ užití realizovat. Následně je popsáno bod, na kterém uživatel skončí

po úspěšném ukončení a post-stav (provedená akce). Poté jsou popsány kroky, které vedou k úspěšnému scénáři a alternativní scénář. Jako poslední je, opět pomocí MosCow techniky) označena priorita a možné potenciální problémy. V tabulkách nejsou zmíněni aktéři, protože se jedná vždy pouze o uživatele a pro přihlašovací akce Firebase Authorization. Situace ohledně aktérů bude detailněji znázorněna v Use Case Diagramech.

4.1.3.3.1 Registrace

Vytvoření osobního účtu k přístupu do aplikace.

Tabulka 1 – Use Case - Registrace

ID	1
Předpoklady	-
Úspěšné ukončení	Domovská stránka
Post-stav	Odeslaný potvrzovací email uživateli
Úspěšný scénář	1. Uživatel vybere vytvoření účtu 2. Systém zobrazí náhled registrace 3. Uživatel vloží emailovou adresu 4. Uživatel vloží heslo 5. Uživatel potvrdí heslo 6. Systém odešle potvrzení o úspěšné registraci
Alternativní scénář	3a. Zadané heslo neodpovídá podmínkám 4a. Zadané heslo není stejné 6a. Systém obeznámí uživatele o problému v registraci
Priorita	Must Have
Problémy	Kontrola vytváření účtů uživatelů (např. dvakrát stejný email)

4.1.3.3.2 Přihlášení

Přihlášení do aplikace a provázání s osobním účtem.

Tabulka 2 – Use Case - Přihlášení

ID	2
Předpoklady	Uživatel má účet
Úspěšné ukončení	Homepage
Post-stav	Uživatel je přihlášen
Úspěšný scénář	1. Uživatel vybere možnost přihlášení 2. Uživatel vloží email 3. Uživatel vloží heslo 4. Uživatel zvolí možnost přihlášení 5. Systém obeznámí uživatele o přihlášení
Alternativní scénář	4a. Systém obeznámí uživatele o nesprávném účtu nebo heslu 4b. Systém obeznámí uživatele o chybící emailové adrese 4c. Systém obeznámí uživatele o nevloženém heslu
Priorita	Must Have
Problémy	-

4.1.3.3.3 Obnova hesla

Obnova hesla již vytvořeného účtu.

Tabulka 3 – Use Case - Obnova hesla

ID	3
Předpoklady	Uživatel má účet
Úspěšné ukončení	Homepage
Post-stav	Heslo změněno
Úspěšný scénář	1. Uživatel vybere možnost pro obnovu hesla 2. Systém ukáže náhled obnovy hesla 3. Uživatel vloží email

	4. Vybere odeslání emailu s novým heslem 5. Systém odešle email na danou adresu 6. Uživatel otevře obdržený email a otevře link 7. Vloží nové heslo 8. Potvrdí nové heslo 9. Zvolí změnit heslo
Alternativní scénář	4a. Systém obeznámí uživatele o tom, že zadaný email není spojen s účtem 7a. Systém obeznámí uživatele o tom, že zadané heslo neodpovídá podmínkám 7b. Systém obeznámí uživatele o tom, že potvrzovací heslo není stejné
Priorita	Must Have
Problémy	-

4.1.3.3.4 Zobrazení osobních informací

Zobrazení osobních informací zadaných uživatelem. Součástí osobních informací bude možnost změnit heslo a kteroukoliv osobní informaci.

Tabulka 4 – Use Case - Zobrazení osobních informací

ID	4
Předpoklady	Uživatel je přihlášen
Úspěšné ukončení	Stránka s osobními informacemi
Post-stav	Zobrazené osobní informacemi
Úspěšný scénář	1. Uživatel vybere zobrazení osobních informací 2. Systém zobrazí osobní informace jsou zobrazeny
Alternativní scénář	-

Priorita	Can have
Problémy	Jaké informace zobrazovat s danou aplikací

4.1.3.3.5 Změna osobních informací

Změna osobních informací (Součástí je i jejich nastavení, uživatel nemusí mít zadané žádné kromě emailu a hesla). Součástí je možnost změnit heslo. Email je jediná informace, která nejde změnit.

Tabulka 5 – Use Case - Změna osobních informací

ID	5
Předpoklady	Uživatel je přihlášen
Úspěšné ukončení	Zobrazení nových osobních informací
Post-stav	Informace jsou změněny a jsou viditelné
Úspěšný scénář	1. Uživatel zadá nové informace 2. Uživatel klikne na změnit informace 3. Systém potvrdí změnu
Alternativní scénář	3a) Uživatel měnil heslo a to neodpovídá požadavkům
Priorita	Can have
Problémy	Jaké informace zobrazovat a umožnit změnit

4.1.3.3.6 Cvičení

Výběr lekce dá na výběr mezi cvičebními lekcemi, které následně půjde spustit. Po odehrání lekcí je zobrazeno uživatelovo hodnocení.

Tabulka 6 – Use Case - Cvičení

ID	6
Předpoklady	Uživatel je přihlášen
Úspěšné ukončení	Aplikace zobrazila hodnocení cvičení

Post-stav	Uživatel vidí hodnocení svého cvičení
Úspěšný scénář	1. Uživatel otevře možnosti lekcí 2. Systém zobrazí dostupné lekce 3. Uživatel vybere možnost nové cvičení 4. Systém zobrazí náhled cvičení 5. Uživatel projde cvičením 6. Systém ukončí cvičení 7. Systém zobrazí hodnocení
Alternativní scénář	-
Priorita	Must have
Problémy	-

4.1.3.3.7 Zobrazení skóre

Zobrazení uživatelových hodnocení s informativním obsahem.

Tabulka 7 – Use Case - Zobrazení skóre

ID	7
Předpoklady	Uživatel je přihlášen
Úspěšné ukončení	Aplikace zobrazila uživatelova skóre (hodnocení)
Post-stav	Uživatel vidí svá hodnocení a postup
Úspěšný scénář	1. Uživatel vybere možnost zobrazit skóre 2. Systém zobrazí náhled skóre
Alternativní scénář	-
Priorita	Could have
Problémy	Reakce na to, když uživatel nemá odehrané cvičení

4.1.3.3.8 Odhlášení

Odhlášení z profilu uživatele.

Tabulka 8 – Use Case - Odhlášení

ID	8
Předpoklady	Uživatel je přihlášen
Úspěšné ukončení	Uživatel je vrácen na přihlašovací stránku
Post-stav	Uživatel se může znovu přihlásit
Úspěšný scénář	1. Uživatel zvolí možnost odhlášení 2. Systém zobrazí potvrzení odhlášení 3. Systém zobrazí přihlašovací stránku
Alternativní scénář	-
Priorita	Could have
Problémy	Analýza, jestli je odhlášení potřebné nebo bude provedeno automaticky při vypnutí aplikace

4.1.3.3.9 Zobrazení informací o aplikaci

Zobrazení informací o aplikaci, které přibližují, k čemu aplikace slouží a jak jí správně využívat.

Tabulka 9 – Use Case - Zobrazení informací o aplikaci

ID	9
Předpoklady	Uživatel je přihlášen
Úspěšné ukončení	Uživateli jsou zobrazeny informace o aplikaci
Post-stav	-
Úspěšný scénář	1. Uživatel zvolí možnost zobrazit informace o aplikaci 2. Systém zobrazí stránku informací o aplikaci
Alternativní scénář	-
Priorita	Should have
Problémy	-

4.1.3.3.10 Zobrazená pravidel cvičení

Zobrazení pravidel jednotlivých cvičení.

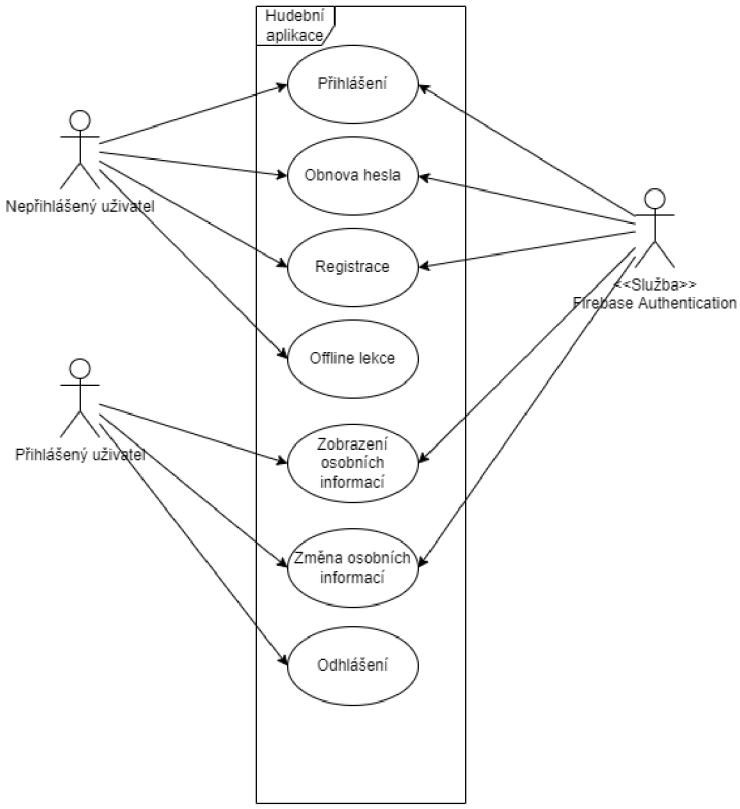
Tabulka 10 – Use Case - Zobrazení pravidel cvičení

ID	10
Předpoklady	Uživatel je přihlášen
Úspěšné ukončení	Uživateli jsou zobrazeny informace o cvičení
Post-stav	Uživatel má znalosti nutné ke korektnímu použití cvičení
Úspěšný scénář	1. Uživatel zvolí možnost zobrazit pravidel cvičená 2. Systém zobrazí stránku pravidel cvičení 3. Uživatel vybere cvičení, které chce pochopit 4. Systém zobrazí pravidla daného cvičení
Alternativní scénář	-
Priorita	Should have
Problémy	Pravidla nebudou pochopitelně podaná

4.1.3.4 Use Case Diagramy

4.1.3.4.1 Use Case Diagram pro operace s účtem

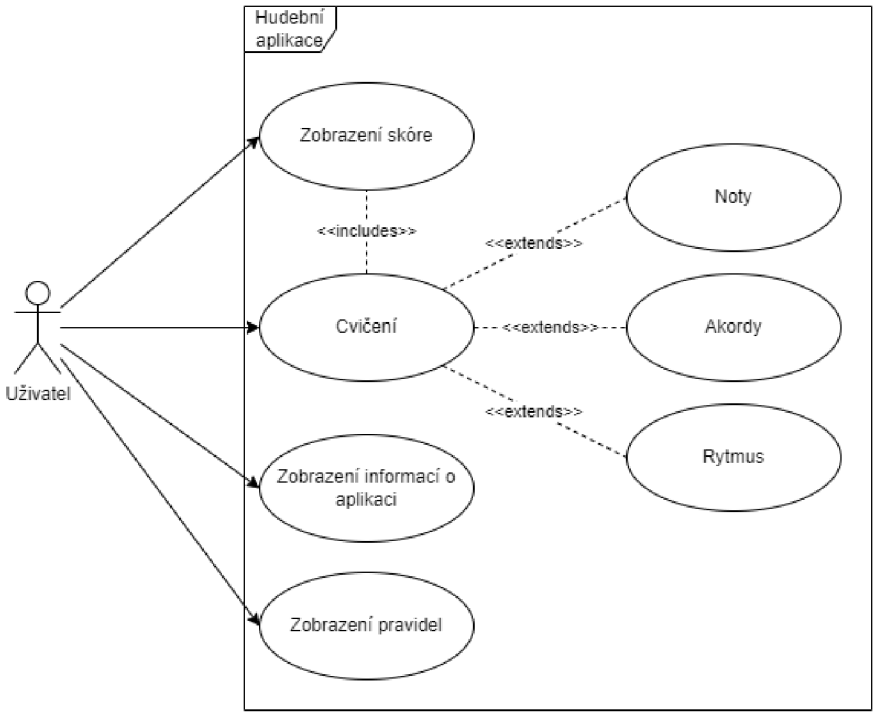
Use Case Diagram (viz. Obrázek 14) zobrazuje případy užití pro přihlášení, obnovu hesla, registraci, offline lekci, zobrazení osobních informací, změny osobních informací a odhlášení. V diagramu se objevují tři aktéři: Nepřihlášený uživatel, přihlášený uživatel a služba Firebase Authentication, která poskytuje data a umožňuje většinu operací.



Obrázek 13 - Use Case Diagram pro práci s účtem

4.1.3.4.2 Use Case Diagram pro funkcionalitu

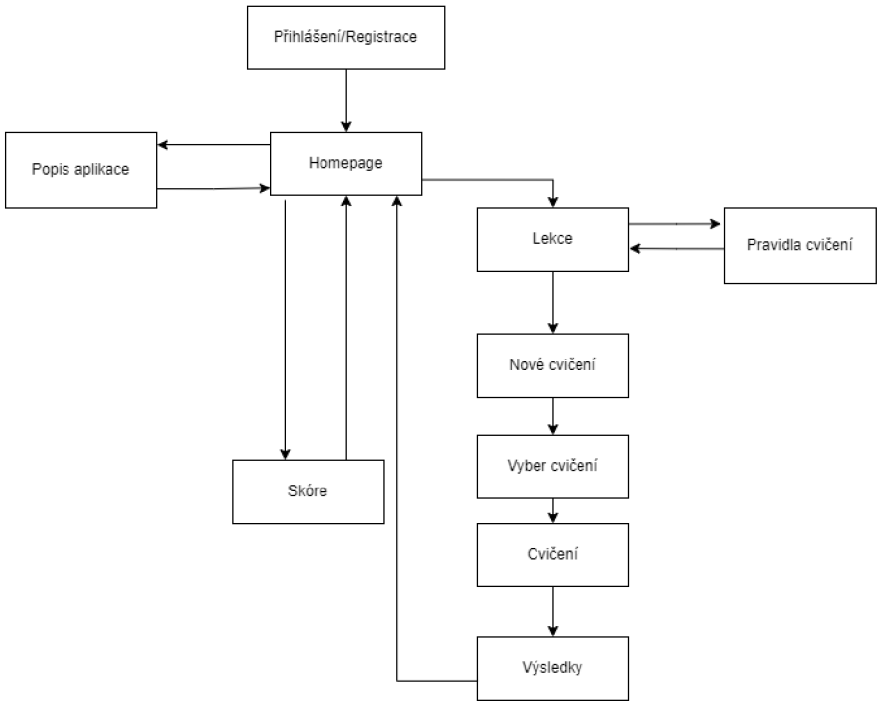
Use Case Diagram (viz. Obrázek 15) pro funkcionalitu chování systému z hlediska uživatele. Obsahuje zobrazení skóre, cvičení, zobrazení informací o aplikaci (které ukrývá více informativních částí) a zobrazení pravidel. Cvičení je následně rozšířeno do tří dalších lekcí: Not, Akordů a Rytmu.



Obrázek 14 – Use Case Diagram pro funkcionalitu

4.1.4 Navigační diagram

Navigační diagram (viz. Obrázek 13) zobrazuje funkční požadavky hudební aplikaci, jejich propojení a přechody mezi nimi. Na úvod se uživatel přihlásí nebo zaregistruje (práce s účtem), z domovské stránky může zobrazit skóre, popis aplikace nebo přejít na výběr lekcí. Ve výběru lekcí může zobrazit jejich pravidla nebo vybrat možnost nového cvičení. Následně vybírá, které cvičení chce zapnout a po odehrání jsou mu zobrazeny výsledky, načež se vrací na úvodní stránku.



Obrázek 15 - Navigační diagram hudební aplikace

4.1.5 Regulační, technická a business omezení

4.1.5.1 Regulační omezení

Hlavním regulačním omezením, které je nutné vzít v potaz, je copyright (autorské právo). Autorské právo je forma legálního zabezpečení uměleckých děl jako například písně, textu nebo nahrávky (CD, LP, singly) (Strand Peter J., Kouchoukas Robert and Rattner William, 2005). Například Google Play popisuje problematiku takto: „Nepovolujeme aplikace, které porušují autorská práva. Změna díla, na kterou se autorská práva vtahují, může být stále bráno jako porušení podmínek. Vývojáři mohou být požádáni o předložení práv k užívání děl.“ (Google Console Help, 2022) S tímto jsou spojená i dvě další omezení, kterými jsou vzorkování (sampling; užívání částí předchozích nahrávek) a publikování hudby (music publishing; užívání nahrávek za účelem vlastního zisku). Při použití takových nahrávek k tomu musí mít autor právo.

V důsledku programované aplikace to znamená, že nesmí obsahovat žádnou píseň, ani její část, na níž se vztahují autorská práva a ke které programátor nemá přístup. Mimo jiné se na aplikaci vztahují i všechna ostatní regulační omezení, která se běžné vztahují na

mobilní aplikace. Znění některých z nich lze najít na stránkách Iubenda. Dosažení tohoto omezení bude zajištěno použitím volně dostupných zdrojů nebo vlastních nahrávek.

V neposlední řadě musí také aplikace splňovat omezení daná GDPR (to zahrnuje například právo být zapomenut, práva na přístup k datům a sledování souhlasů a povolení). Mezi další práva vztahující se k GDPR, nicméně z podstaty diplomové práce nijak zásadní, se jedná právo na vědomí o hackerském útoku, při kterém došlo ke kompromitování dat a datová přenositelnost (data portability), při které poskytovatel služby musí poskytnout v elektronickém formátu (PrivacyPolicies.com Legal Writing Team, 2021).

4.1.5.2 Technická omezení

- Hardwarová omezení – Uživatel musí mít mobilní zařízení s operačním systémem Android s kvalitami dostatečnými pro zvládnutí základní mobilní aplikace. Zařízení by mělo být schopné reflektovat kytaru bez grafických nedostatků a nepřesností, a tudíž by mělo mít více než 5,5 palce.
- Omezení operačního systému – Zařízení by mělo mít dostupný alespoň Android 8.
- Programovací jazyk – K programování aplikace bude použit jazyk Kotlin, který vychází ze specifikace zadání diplomové práce a který byl zvolen, protože se jedná o moderní jazyk sloužící k vývoji mobilních aplikací.

4.1.5.3 Business omezení

Specifikace business omezení je komplikované, protože se jedná především o omezení spojená s podnikáním nebo pozicí stakeholderů, která jsou v případě diplomové práce těžko dosažitelná. Mezi univerzální business omezení nicméně lze vzít:

- Plán – Diplomová práce, a tím pádem i vývoj aplikace, musí být hotová do data určeného k odevzdání diplomové práce.
- Kompozice teamu – Team se skládá pouze z jedné osoby, která zajišťuje všechny součásti aplikace
- Trh – Kvalita specifických funkcí aplikace by se měla být nejméně rovna kvalitě aplikací na trhu. Zároveň však aplikace musí nabízet funkcionality a přístupy, které konkurenti nemají.

4.2 Návrhová fáze

Po počáteční (research) fázi jsou jasně specifikovaná omezení a požadavky pro aplikace. Dalším krokem vývoje mobilní aplikace je návrhová fáze, ve které je rozhodnuto, jak bude aplikace organizována v závislosti na funkcích, které budou implementovány. Na jejich základě by mělo být možné vytvořit prototypy (v tomto případě se bude jednat o spojení statického designu mockupů s dynamickým přístupem prototypů) a softwarovou architekturu. Zároveň už by měly být specifikované technologie, které budou použity.

4.2.1 Softwarová architektura

K návrhu softwarové architektury je použit C4 model. C4 model je založen na strukturální dekompozici systému do kontejnerů a komponentů, což zobrazuje několik způsobů náhledu na systém, vazby mezi jednotlivými celky a vztahy s uživateli. C4 model spolupracuje s již existujícími modelovacími technikami, jako je například UML (Unified Modelling Language) nebo ERD (Entity Relationship Diagram).(Brown Simon, 2011)

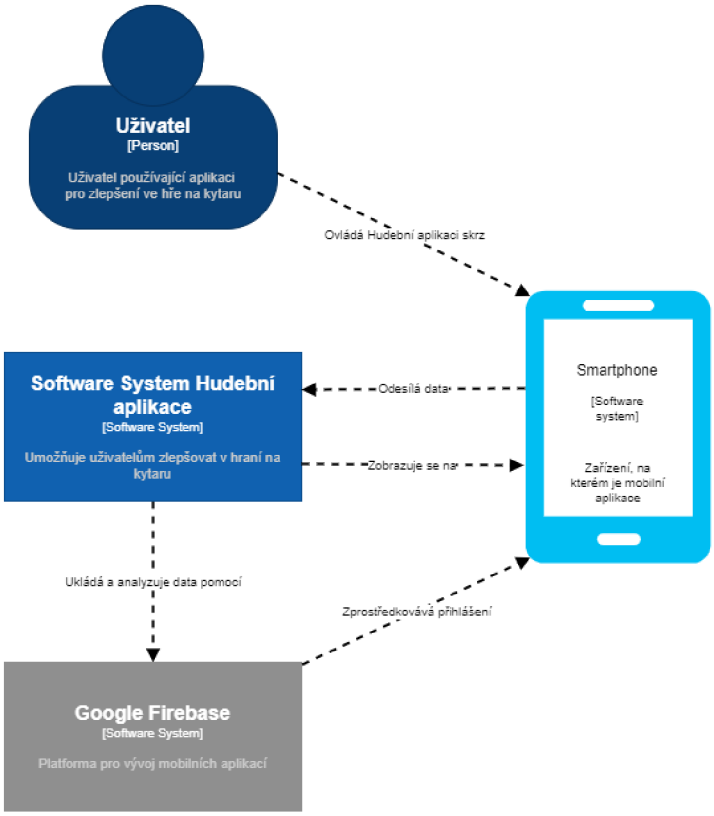
C4 model je organizován do 4 úrovní:

- System context diagram – Diagram, který zobrazuje systém a jeho vztah k uživatelům a dalším relevantní systémům
- Container diagram – Diagram, ve kterém je systém dekomponován do propojených kontejnerů. Kontejner by měl být samostatný a spustitelný subsystém.
- Component diagram – V tomto diagram jsou kontejnery dekomponovány do propojených komponent a ty jsou spojeny s ostatními kontejnery nebo systémy.
- Code diagram – Každý komponent může být přiblížen, aby šlo vidět, jak vypadá jeho kód. Na tuto úroveň se většinou používá UML diagram tříd nebo ERD a většinou se jedná o nepovinnou část, která se dá vygenerovat pomocí IDE. V případě této práce Code diagram součástí C4 modelu prozatím nebude.

4.2.1.1 System context diagram

V system context diagramu (viz. Obrázek 16) jsou zobrazeny čtyři hlavní části. Uživatel, smartphone, hudební aplikace (její softwarový systém) a Google Firebase. Uživatel je specifikovaný jako jakýkoliv uživatel aplikace, jež ji používá ke vzdělávacímu účelu.

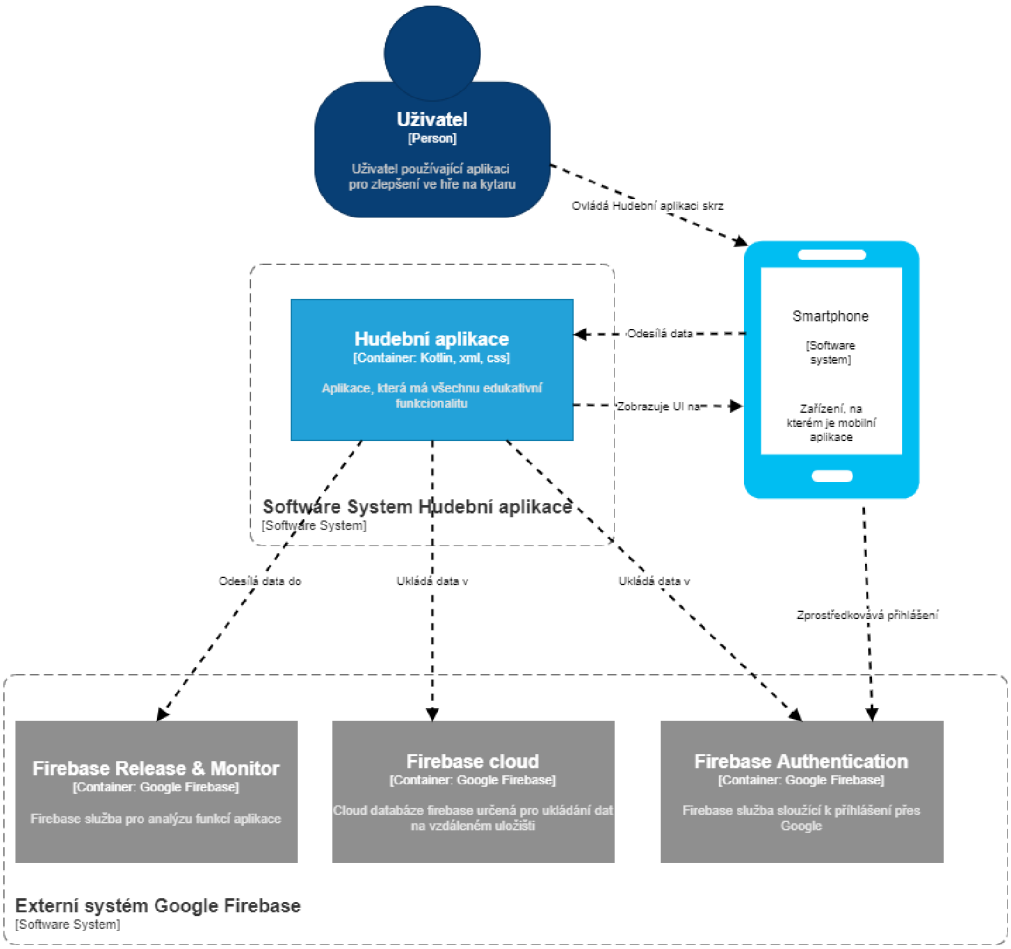
Smartphone je v tomto případě jakékoliv zařízení pracující s OS Android. Software Systém hudební aplikace je hudební aplikace a její součásti. Google Firebase je externí systém, který vlastní cloudovou databázi, ve které se ukládají data a následně je možné je analyzovat.



Obrázek 16 - System context diagram hudební aplikace

4.2.1.2 Container diagram

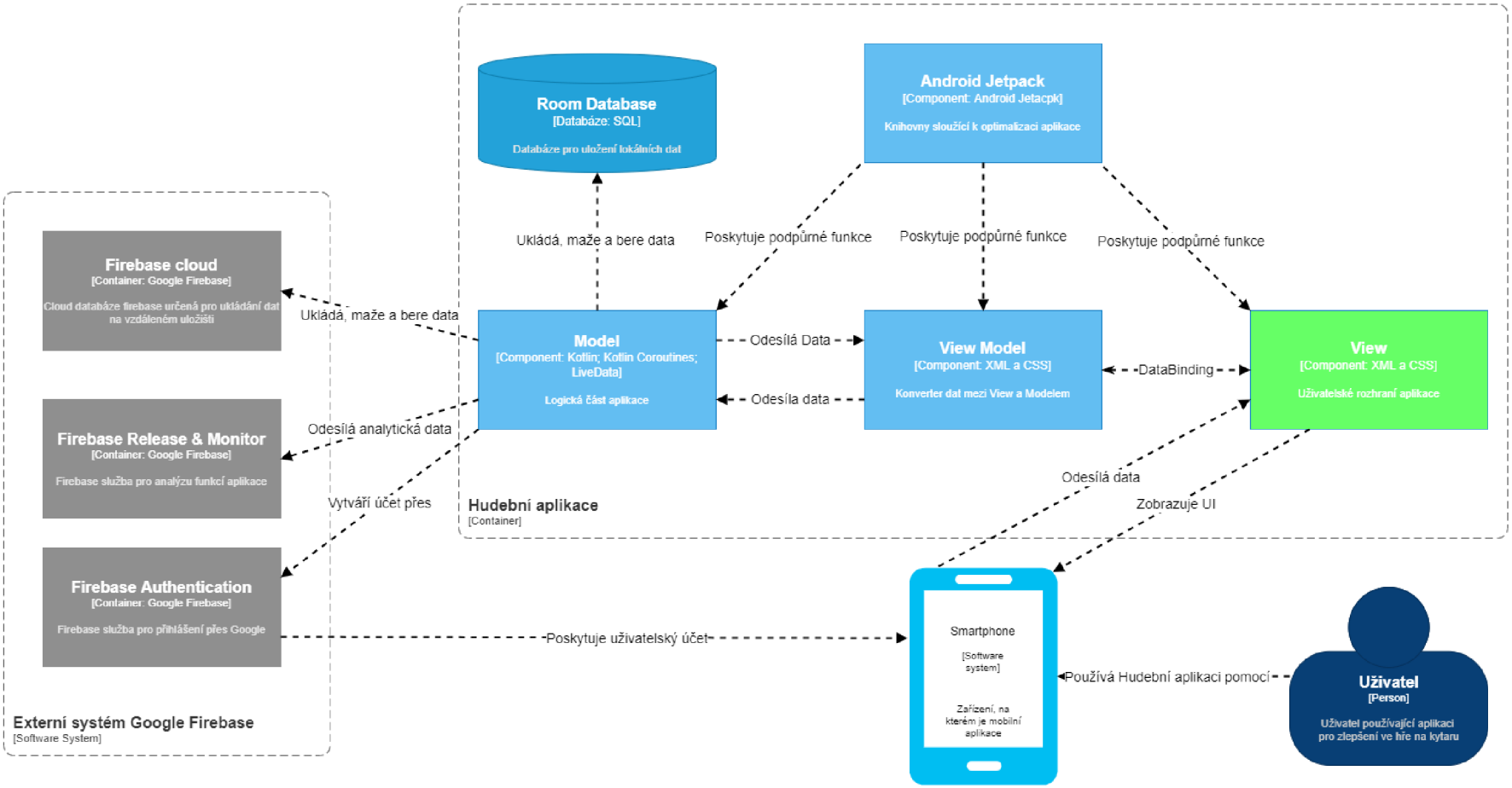
V Container diagramu (viz. Obrázek 17) jsou zobrazeny kontejnery systémů. V tomto případě se v hudební aplikaci jedná opět pouze o jeden, kterým je Hudební aplikace, která v sobě drží uživatelské rozhraní i logiku celého systému. Druhým rozloženým systémem je Firebase, který v sobě drží dva používané kontejnery. Jedním z nich je Release & Monitor, což je monitorovací systém, který slouží především k analýze součástí aplikace (například chyb) a cloud, na který hudební aplikace ukládá data. Ostatní prvky jsou přeneseny ze system context diagramu.



Obrázek 17 - Container diagram hudební aplikace

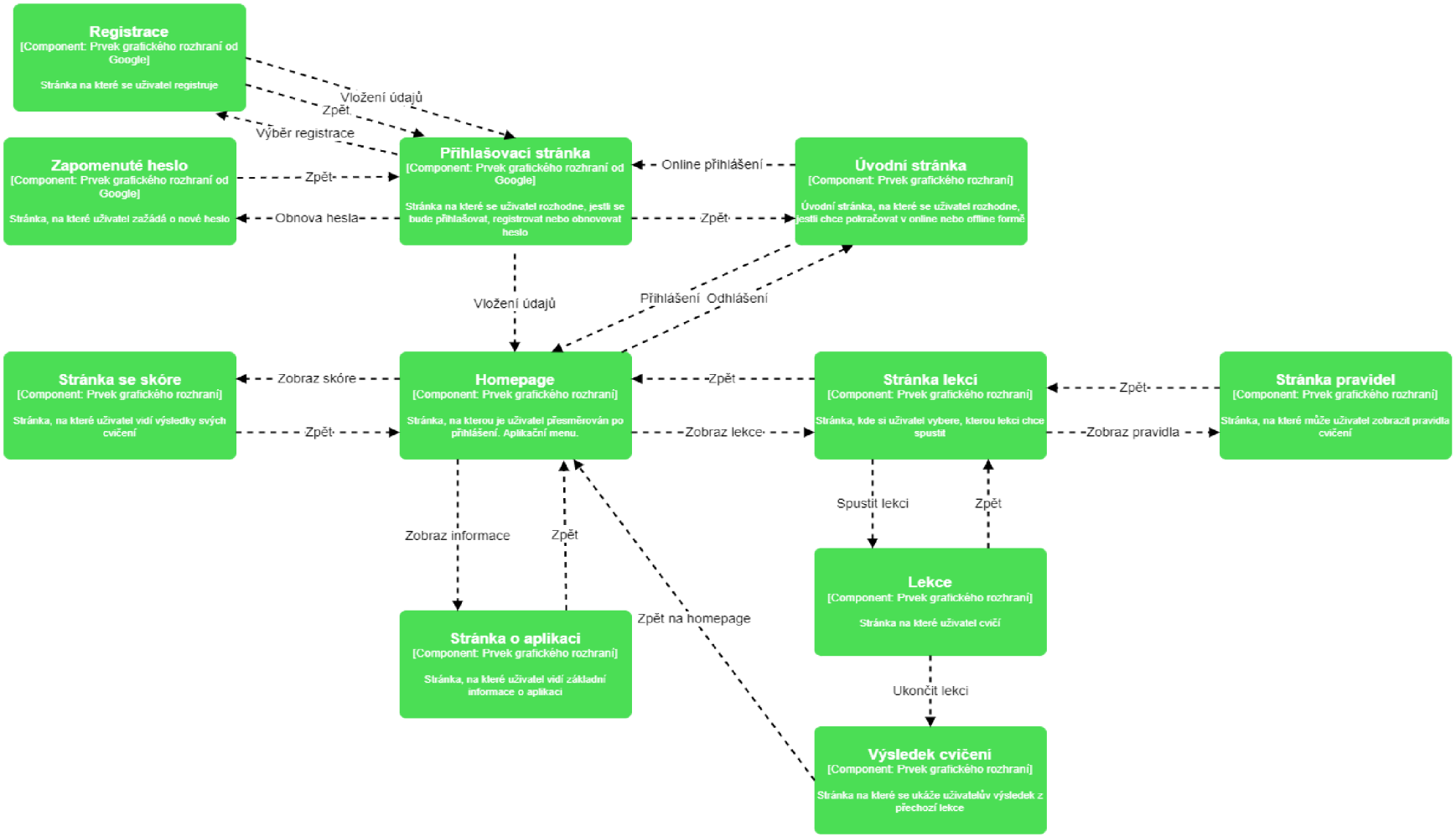
4.2.1.3 Component diagram

Třetí úrovní je Component diagram. Ten rozkládá kontejnery do komponent. V tomto případě se jedná o kontejner Hudební aplikace, který vlastní hlavní část celého systému. Pro přehlednost je diagram rozdělen do dvou částí. První z nich (hlavní část) ukazuje jednotlivé části systému (viz. Obrázek 18). Mimo součásti předchozích úrovní lze vidět View (což je uživatelské rozhraní), Model (logická část aplikace) a View Model, který zajišťuje jejich hladkou komunikaci. Toto rozložení odpovídá návrhovému vzoru MVVM (viz. Kapitola MVVM). Mimo to také obsahuje Android Jetpack, který slouží k zajištění některých funkcionalit aplikace, například právě MVVM, jež je jeho součástí nebo například Jetpack Navigation, který bude sloužit k přechodům. V neposlední řadě se skládá z Room Database, která slouží jako uložiště při režimu bez připojení.



Obrázek 18 - Component diagram bez rozložení view hudební aplikace

Druhá součást Component diagramu (správně by měla být součástí první) je rozložení View součástí (uživatelského rozhraní, viz. Obrázek 19). To ukazuje jednotlivé stránky (screeny) aplikace a jejich vzájemné přechody. Jejich informační toky v softwarové architektuře zařazené nejsou a většinu jejich komunikace zprostředkovává ViewModel a Smartphone (v případě kompletního grafu by byly vázány právě na ně).



Obrázek 19 - Rozložení View komponentu

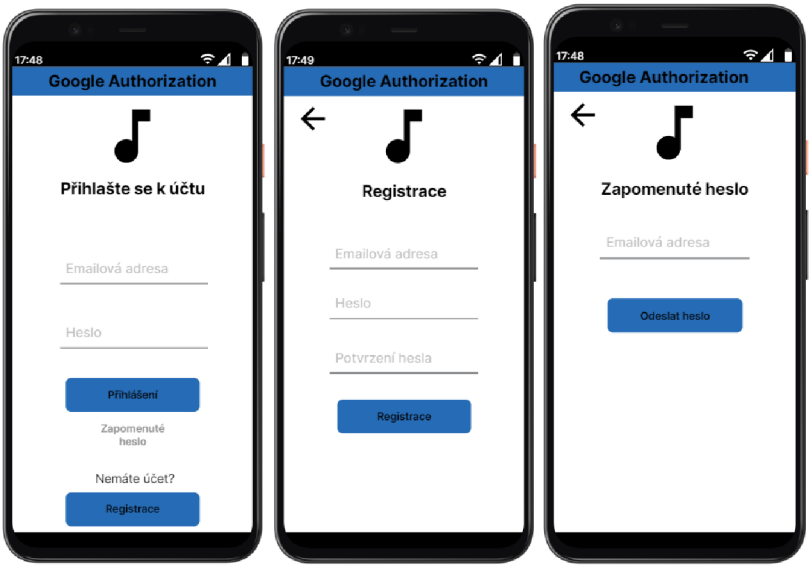
4.2.2 Prototyp

Po softwarové architektuře byl udělán prototyp aplikace. Ten v případě hudební aplikace poskytuje základní prvky UX (a částečně i UI) a slouží k představě o celkovém workflow aplikace po její implementaci. Prototyp nemá kompletní uživatelské rozhraní, které bude v aplikaci, nicméně ve zjednodušené podobně pokrývá všechny prvky, které obsahovat bude. Jelikož aplikace nemá přímou validaci, jako základ prototypu nebyl vytvořen Wireframe a prototyp tak částečně zastává i jeho účel. Pro vytvoření prototypu byl použit online nástroj Framer (Framer B.V., 2022). Jedná se o multifunkční nástroj, který slouží k modelování UX rychle a bez kódu. Jako model byl použit mobilní telefon Google Pixel 4.

Jednotlivé obrazy prototypu jsou seskupeny do skupin podle funkcionalit. Těmi jsou správa účtu (přihlášení, registrace, změna hesla), menu (úvodní stránka, homepage, lekce), výsledky (výsledky, skóre), vysvětlivky (o aplikaci, pravidla) a pak samostatné zobrazení cvičení. Pohyb mezi jednotlivými skupinami je popsán v component diagramu softwarové architektury a v navigačním diagramu.

4.2.2.1 Správa účtu

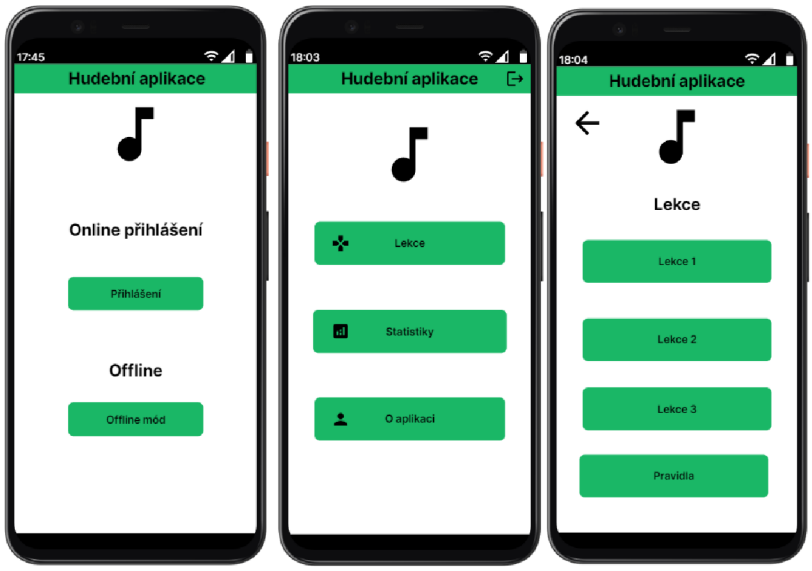
Správa účtu slouží k připojení uživatele do zařízení a k manipulaci s osobními informacemi. Kvůli problematice spojené s bezpečností je celý proces vytvořen přes Firebase Authorization (Google). Jako nutné informace pro připojení jsou brány pouze emailová adresa a heslo (vynecháním ostatních je limitován problém s GDPR). Hlavní stránkou je přihlašování, které slouží k zadání emailů a hesla pro přenesení na domovskou stránku (homepage). Mimo to se z něj lze dostat i na registraci a obnovu hesla. Registrace umožňuje účet vytvořit, kdežto obnova hesla umožňuje se připojit k účtu se ztraceným heslem. Prototyp správy účtu lze vidět na Obrázek 20.



Obrázek 20 - Prototyp správy účtu

4.2.2.2 Menu

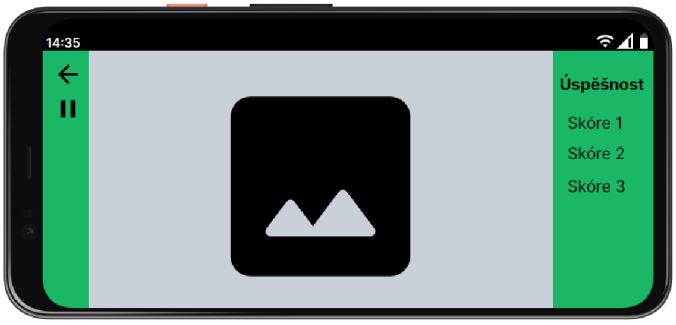
Menu je v podstatě rozcestí aplikace. S prvním menu se lze setkat hned po zapnutí aplikace. Ta poskytuje možnost přihlášení a offline režim. Obě tyto stránky postupně vedou na hlavní stránku. Hlavní stránkou je homepage (domovská stránka), na které dostane uživatel na výběr ze tří akcí hlavních akcí (mimo ně se ještě může odhlásit). Po přesunutí do výběru z lekcí dostane uživatel na výběr ze dvou lekcí, kde jedno slouží k cviku akordů a druhé rytmu. Prototyp menu lze vidět na Obrázku 21.



Obrázek 21 - Prototyp menu

4.2.2.3 Cvičení

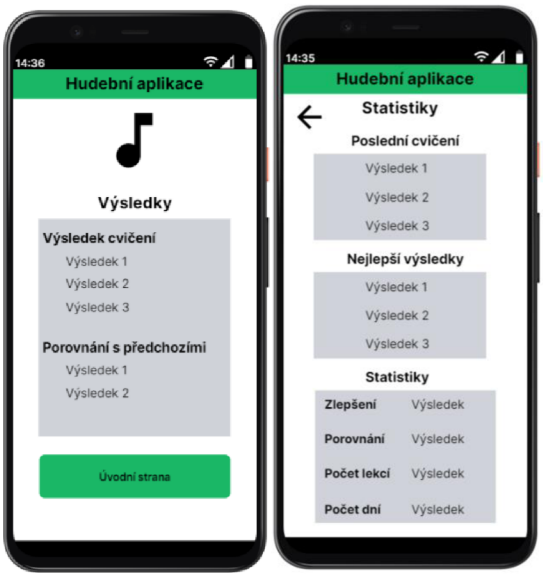
Všechna cvičení budou mít obdobnou obrazovku. Na ní bude samostatné grafické rozhraní cvičení, které bude simulovat kytarové pražce. Na pravé straně obrazovky bude zobrazena úspěšnost přepínání a na levé bude možnost cvičení přerušit nebo ukončit. Cvičení bude ukončeno také jeho splněním nebo selháním. Prototyp cvičení lze vidět na Obrázku 22.



Obrázek 22 - Prototyp zobrazení cvičení

4.2.2.4 Výsledky

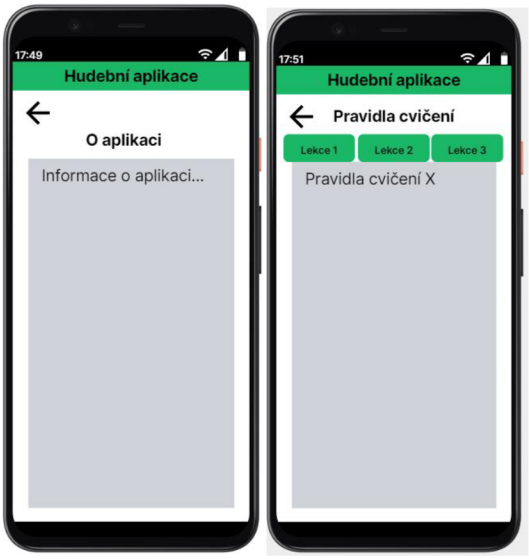
Předposlední, především informativní částí, jsou výsledky. První obrazovka výsledků je zobrazena bezprostředně po dokončení nebo po selhání při cvičení. Na té lze vidět pouze výsledek z daného cvičení a následné porovnání s nejlepšími dosaženými hodnotami. Druhá obrazovka ukazuje celkový souhrn ze všech her a jejich statistiky. Slouží především za účelem motivace uživatele pomocí postupného zlepšování a překonávání vlastních cílů. Prototyp stránky výsledků lze vidět na Obrázku 23.



Obrázek 23 - Prototyp zobrazení výsledků cvičení

4.2.2.5 Vysvětlivky

Doplňující částí jsou informativní obrazovky o aplikaci a pravidlech. První z nich, O aplikaci, uvádí uživatele do kontextu aplikace. Pravidla slouží k pochopení samotných her a ke korektnímu užití. Obě slouží primárně jako poskytovatelé informací o aplikaci, jejím kontextu a správném užívání. Prototyp stránky vysvětlivek lze vidět na Obrázku 24.



Obrázek 24 - Prototyp pomocných vysvětlivek

4.3 Vývojová fáze

Na základě předchozích částí je vyvíjena aplikace odpovídající specifikaci. Návrhová fáze je naprogramována do aplikace, která obsahuje všechnu specifikovanou funkcionalitu. Jestliže jedna z naprogramovaných částí nebude odpovídat předchozí specifikaci, změna je popsána a vysvětlena. Jelikož v předchozích fázích se mohlo stát, že chyběl kontext pro některé vyvíjené částí, může dojít k ojedinělé zpětné změně předchozích částí.

Ve vývojové fázi je popsán způsob implementace aplikace a jednotlivé důležité prvky, které popisují funkcionalitu. V případě obdobných využití metod a tříd je zmíněna pouze jedna z nich, která je nejlépe shrnovat obecné prvky všech podobných částí.

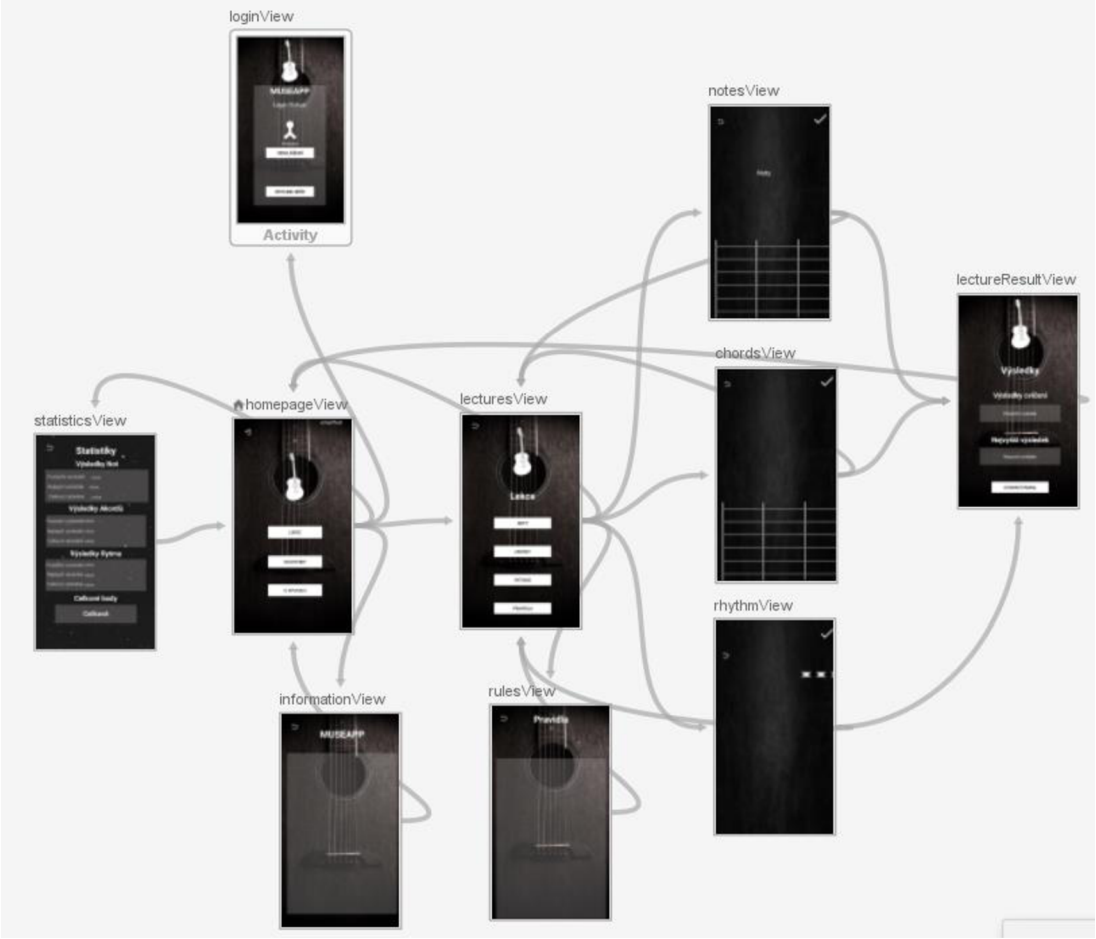
4.3.1 Obecná konfigurace

Konfigurace prostředí krátce shrnuje stav obecných konfigurací a programovacího prostředí, které byly použity k implementaci aplikace. Jedná se o:

- **Programovací jazyk:** Kotlin 203-1.5.20-release-289-AS7717.8
- **Jazyky pro grafické rozhraní** – XML, CSS
- **Vývojové prostředí:** Android Studio Arctic Fox 2020.3.1 Patch 4
- **Emulátor:** Pixel 2 XL API 26
- **Java Virtual Machine** – 1.8
- **Android Gradle:** 7.0.4, Gradle JDK 1.8
- **Compile SDK:** API 31
- **Databáze** – Cloud Firebase a Room database (více viz výše)
- **Dependencies** – appcompat, constraintlayout, core-ktx, espresso-core, firebase, junit, kotlin-stdlib, lifecycle-extensions (viewmodel), navigation, play-services, recyclerview, room
- **Testovací zařízení** – Xiaomi Mi A3, Pixel 2 XL a další
- **Grafické prvky aplikace** – guitar 1 by papapishu (Papapishu, 2007), gamebackground (@benzoix, 2022), generalbackground (wallpapersafari.com, 2020), další volně dostupné prvky poskytnuté prostředím Android Studio

4.3.2 Navigation

Jak již bylo zmíněno, k pohybu skrz jednotlivé části aplikace je použita Jetpack Navigation. Ta se skládá ze tří základních součástí a těmi jsou Navigation graph (viz. Obrázek 19), NavHost a NavController.



Obrázek 25 - Navigation graph hudební aplikace

V navigačním grafu jsou zobrazeny jednotlivé obrazovky (fragmenty nebo aktivity) aplikace. Jak lze vidět, hudební aplikace se skládá z deseti obrazovek, které odpovídají navigačnímu grafu a prototypu. Jako názorná ukázka slouží zobrazení přechodu mezi *homepageView* a *informationView*, která pouze popisuje kód, který se za vizualizací navigation graphu ukrývá.

```
<fragment
    android:id="@+id/informationView"
    android:name="com.example.guitar_music_app.helpers.InformationView"
    android:label="information_fragment"
    tools:layout="@layout/information_fragment" >

    <action
        android:id="@+id/action_informationView_to_homepageView"
        app:destination="@id/homepageView" />
</fragment>
```

Zdrojový kód 1 - Implementace „O aplikaci“ v navigačním grafu

Implementace „O aplikaci“ (viz. Zdrojový kód 1) se skládá ze dvou hlavních částí. Jedná se o popis samotného fragmentu a přípravy rozložení fragmentu (připravení jeho grafického rozhraní). Druhou součástí je „action“, která popisuje možnosti navigace z daného fragmentu. V případě „O aplikace“ se jedná pouze o návrat na úvodní stránku.

NavHost je fragment, který je nad aktivitou obsahující funkcionalitu. NavHost zobrazuje cílové pozice navigačního grafu (viz. Zdrojový kód 2).

```
<fragment
    android:id="@+id/fragment_navigation"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="true"
    app:navGraph="@navigation/navigation_graph"/>
```

Zdrojový kód 2 - NavHost aplikace v xml result třídy

Poslední součástí je NavController, což je nástroj, který slouží k ovládání navigace z aplikačního kódu. NavController se inicializuje v *onCreate* funkci aktivity (viz Zdrojový kód 3). NavController poskytuje funkce, které umožňují navigaci mezi jednotlivými obrazovkami aplikace.

```
private lateinit var nav: NavController

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.result_activity)
    nav = Navigation.findNavController(this, R.id.fragment_navigation)
}
```

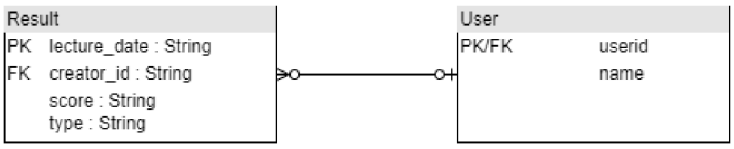
Zdrojový kód 3 - Implementace NavControlleru v result aktivitě

4.3.3 Databáze

Aplikace používá dvě databáze: Room database a Cloud Firebase. První slouží pro ukládání výsledků v lokálním prostředí, a tudíž podporuje režim bez připojení k internetu. Cloud Firebase je cloudová databáze, a tudíž funguje jen při připojení k internetu. Slouží

k ukládání uživatelů a jejich výsledků. Její hlavní výhodou je škálovatelnost a zabezpečení díky poskytovateli z třetí strany. Ačkoliv Firebase poskytuje i vlastní lokální databázi, po analýze byla zvolena možnost Room Database, tudíž aplikace není závislá pouze na jednom poskytovateli.

Samotná vytvořená databáze je jednoduchá (viz Obrázek 20). Jedná se pouze o vztah mezi uživatelem a výsledky. Uživatel může mít jeden nebo více výsledků. Výsledek nemusí mít žádného uživatele (použití room databáze).



Obrázek 26 - ERD hudební aplikace

V ERD diagramu lze také vidět jednotlivé prvky databáze. Výsledek se skládá z data vytvoření (primární klíč, který je definovaný na milisekundy, tudíž je odstraněna možnost duplicity), cizí klíč je uživatelovo ID, které je generováno pomocí firebase a jestliže výsledek nemá uživatele, jedná se o prázdnou hodnotu. Mimo jiné má výsledek také skóre (určující hodnotu výsledku lekce) a typ (který určuje, ze které lekce bylo výsledku dosaženo). Uživatel se skládá z uživatelského ID a jména. V tomto případě je jako oba klíče použita kombinace (spojení userid a name), tudíž možnost duplicity je nulová. Uživatel nemusí mít žádný výsledek, ale může jich mít i několik. Výsledek nemusí mít (podle použití databáze) žádného nebo jednoho uživatele.

Výsledky obou databází jsou převáděny do a z unifikovaného formátu tak, aby aplikace mohla pracovat s oběma databázemi stejně (viz. Zdrojový kód 4). Převod probíhá vkládáním hodnot jednoho formátu do druhého (formáty hodnot jsou totožné).

```
internal val FirebaseResult.toResult: Result
    get() = Result(
        this.creationDate ?: "",
        this.score,
        this.type,
        User(this.creator ?: "")
    )
```

Zdrojový kód 4 - Převod do unifikovaných hodnot z Firebase výsledku

4.3.3.1 Implementace Room databáze

Room databáze slouží výhradně k lokálnímu ukládání dat, a proto neimplementuje uživatele. Skládá se tedy pouze z tabulky výsledků. Tabulka je naplněna pomocí výsledků (jejich podoba lze vidět v Obrázku 20).

```
private const val DATABASE = "results"

@Database(
    entities = [RoomResult::class],
    version = 1,
    exportSchema = false
)
abstract class RoomResultDatabase : RoomDatabase() {

    abstract fun roomResultDao(): ResultDao

    companion object {

        @Volatile
        private var instance: RoomResultDatabase? = null

        fun getInstance(context: Context): RoomResultDatabase {
            return instance
                ?: synchronized(this) {
                    instance
                        ?: buildDatabase(context).also { instance = it }
                }
        }

        private fun buildDatabase(context: Context): RoomResultDatabase {
            return Room.databaseBuilder(context,
RoomResultDatabase::class.java, DATABASE)
                .build()
        }
    }
}
```

Zdrojový kód 5 - Room databáze

Ve schématu Zdrojový kód 5 je definována databáze *RoomResult* (výsledky). Databáze je definována abstraktní třídou *RoomResultDatabase*, která rozšiřuje *RoomDatabase*. Knihovna Room z *RoomResultDatabase* vygeneruje kód potřebný k vytvoření databáze a poskytne DAO (data access object) pro přístup k datům. K databázi přistupujeme pomocí metody *getInstance*, která vrátí singleton databáze (popřípadě ho vytvoří, jestliže neexistuje).

Pro přístup k datům jsou využívány metody DAO (data access object). Metody jsou definované pomocí SQL příkazů (viz. Zdrojový kód 6). Room umožňuje validaci příkazu SQL podle existujících objektů.

```
@Dao
interface ResultDao {
    @Query("SELECT * FROM results")
    suspend fun getResults(): List<RoomResult>

    @Query("SELECT * FROM results WHERE lecture_date= :lectureDate")
    suspend fun getResultById(lectureDate: String): RoomResult

    @Delete
    suspend fun  deleteResult(result: RoomResult)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun  insertOrUpdateResult(result: RoomResult): Long
}
```

Zdrojový kód 6 - Data Access Object výsledků

Po vytvoření příkazů je automaticky generována implementace (*ResultDao_Impl*), která rozepisuje jednotlivá volání v příslušném programovacím jazyce (kotlin). Mimo jiné je také na základě definování tabulky generována její implementace (*RoomResultDatabase_Impl*).

4.3.3.2 Implementace Firebase databáze

4.3.3.2.1 Výsledky

První část zobrazuje práci s výsledky ve Firebase Firestore (viz. Příloha 13). Vytvoření cloudové databáze vyžaduje menší množství kódu než definice Room database. V tomto případě je databáze automaticky vytvořena na základě vkládaných prvků. Vložený prvek se jmenuje *FirebaseResult* a jeho struktura je stejná jako výsledek v Room. Firebase pro práci s databází nepoužívá SQL notaci. Přístup k datům probíhá pomocí volání funkcí přímo v jazyce Kotlin (viz. Zdrojový kód 7)

```
private suspend fun getRemoteResults(user: User): GeneralResult<Exception,
List<Result>> {
    return try {
        val task = awaitTaskResult(
            remote.collection(COLLECTION_NAME)
                .whereEqualTo("creator", user.userid)
                .get()
        )

        resultToResultList(task)
    } catch (exception: Exception) {
        GeneralResult.build { throw exception }
    }
}
```

Zdrojový kód 7 - Firebase příkaz pro získání výsledků uživatele

Na Zdrojovém kódu 7 lze vidět, jak vypadá volání do Firebase databáze, které zobrazí všechny výsledky pro jednoho uživatele. Metoda je asynchronní (suspend) tudíž probíhá na

neblokujícím vlákně a nezastavuje funkcionalitu (stejně jako ostatní volání do databáze). Výsledek funkce rozšiřuje *GeneralResult*, což je třída, která zaštiťuje různé výsledky činností v aplikaci (nejen spojené pouze s lekcemi) a skládá se z hodnoty při úspěchu, hodnoty při chybě a build funkci, která určuje, o který výsledek se jedná. Funkce volá *awaitTaskResult* metodu, která slouží k pozastavení vlákna, dokud nebude výsledek proveden, a tak není možné provést další databázové volání, dokud není daná funkce hotová. *Remote* je inicializovaná Firebase databáze a *collection()* odkazuje na specifické místo v databázi (v tomto případě celá databáze). *WhereEqualTo()* první umožňuje zadat který parametr bude porovnáván a hodnoty, pro které porovnávání chceme provést. *Get()* celý příkaz provede. *resultToResultList* provede pro každý zápis v knihovně převod do mutable listu a hodnoty tak budou v zobrazitelném formátu. Tato funkce je identická volání funkce *getResults()* pro lokální databáze (viz Zdrojový kód 6).

Podobná operace je provedena i pro získání specifického výsledku, smazání výsledku a úpravy výsledku. Jediným větším rozdílem v těchto metodách je, že místo *whereEqualsTo()* metody je volána metoda *document()*, která odkazuje na určité místo v tabulce.

4.3.3.2.2 Uživatel

Pro uživatelské operace je používána FirebaseAuth, která je vstupem Firebase Authentication SDK. Po získání její instance může používat metody pro práci s účtem. V případě Hudební aplikace pracuje s účtem třída *FirebaseUserRepo*, která rozšiřuje interface *UserRepository*. *UserRepository* má tři funkce metody: *getCurrentUser()* pro získání nynějšího uživatele, *signOutCurrentUser()* pro odhlášení a *signInGoogleUser()*, která umožňuje přihlášení. Ve *FirebaseUserRepository* přepisujeme vytvořené třídy. Pro představu je zobrazena metoda *getCurrentUser()* (viz. Zdrojový kód 8). Na podobném principu, jako má ona fungují i zbývající metody.

```
override suspend fun getCurrentUser(): GeneralResult<Exception, User?> {
    val firebaseUser = auth.currentUser

    return if (firebaseUser == null) {
        GeneralResult.build { null }
    } else {
            GeneralResult.build {
                User(
                    firebaseUser.uid,
                    firebaseUser.displayName ?: ""
                )
            }
    }
}
```

Zdrojový kód 8 - Metoda FirebaseAuth pro získání uživatele

V třídě *getCurrentUser()* (viz. Figure 8) vidíme, že je nejprve z auth (instance *FirebaseAuth)* získán nynější uživatel pomocí poskytnutých funkcí. Následně je zkontrolováno, jestli není hodnota nulová a jestliže není, jsou vráceny základní informace uživatele (id a jméno).

4.3.3.3 Propojení databází a rozhodovací proces

Základní třídou (interface) propojení databází je *ResultRepository*, která specifikuje čtyři metody sloužící jako volání do databáze výsledku. Těmi jsou *getResultById()* - získání výsledku pomocí id, *getResults()* – získání všech výsledků, *deleteResult()* – smazání výsledku, *updateResult()* – upravení výsledku. Tento interface je implementován pomocí *ResultRepoImpl* třídy, která iniciuje jak obě Firebase Firestore a Authorization, tak Room Database. V případě Firebase je přímo implementována logika volání databáze (viz. Figure 26) a následně je proveden převod na unifikované jednotky nebo zpět. V případě Room zde dochází pouze k převodu na unifikované jednotky a zpět. V neposlední řadě je zde také specifikována funkce, která převádí výsledky z Firebase databáze do listu výsledků.

Nejdůležitějším prvkem implementace *ResultRepository* je rozhodovací proces, který určuje, která z databází bude použita. Její ukázku lze vidět opět na příkladu přepisu metody *getResults()* – získání všech výsledků (viz. Zdrojový kód 9).

```
override suspend fun getResults(): GeneralResult<Exception, List<Result>> {
    val user = getActiveUser()
    return if (user != null) getRemoteResults(user)
    else getLocalResults()
}
```

Zdrojový kód 9 - Metoda getResults() - výběr databáze

Metoda *getResults()* (viz. Zdrojový kód 9) nejprve získá přihlášeného uživatele z FirebaseAuth, jestliže výsledek není nulový, volá funkci *getRemoteResults()* s parametrem user, která získá všechny firebase výsledky daného uživatele. Jestliže hodnota user nulová je, je volána *getLocalResults()*, která slouží k získání lokálních výsledků. Na stejném principu opět fungují i ostatní volání databáze.

4.3.4 Obecné prvky

Některé prvky jsou duplikovány ve více třídách. Mezi ně patří například třídy poskytující databáze, ViewModel nebo akce, které z aplikace mohou vzniknout. Ačkoliv se tyto třídy mohou v některých případech lišit, popisováním každé z nich by se v práci ocitla nechtěná duplicita. Proto bude zmíněn vždy jeden případ, díky kterému bude pochopitelné, co se děje v kterékoliv další třídě.

4.3.4.1 ViewModelFactory

ViewModelFactory slouží ve zkratce k vytvoření instance ViewModelu. Jelikož hudební aplikace má dependence (části kódu, které jsou nutné pro správnou funkcionalitu části jiné), byl nejprve vytvořen *ViewModelProvider.Factory*, který je připojen k ViewModel konstruktoru. *ViewModelProvider* zajišťuje vytvoření instance ViewModelu s bezargumentovým konstruktorem. Jestliže ale chceme ViewModel s několik argumenty, je nutné použít Factory, kterou lze poslat do *ViewModelProvider* když je potřeba vytvořit instanci ViewModelu. Jako ukázka je použita ViewModelFactory stránky lekce Chords (viz. Zdrojový kód 10).

```
class ChordsViewModelFactory(
    private val resultRepo: ResultRepository
) : ViewModelProvider.NewInstanceFactory() {

    @Suppress("UNCHECKED_CAST")
    override fun <T : ViewModel> create(modelClass: Class<T>): T {

        return ChordsViewModel(resultRepo, Dispatchers.Main) as T
    }
}
```

Zdrojový kód 10 - ChordsViewModelFactory pro instanci ChordsViewModel

Ve Zdrojovém kódu 10 je nejprve vytvořena instance *UserRepository*, která slouží k získání volání spojených s databází uživatelů (možné vidět v sekci Databáze – Uživatel). Metoda *create()* vytváří instanci třídy ViewModelu. T je generický typ označující různé

akce, které mohou přicházet. Tato metoda vrátí *LoginViewModel* s parametry *UserRepository* a *Dispatchers.Main*, jež slouží k vytvoření coroutine na hlavním vlákně, který slouží k operacím s UI.

4.3.4.2 Injector

Druhým obecným prvkem je Injector, který poskytuje tzv. dependency injection. Injector v programování slouží k poskytnutí odkazů na ostatní třídy (dependence) ve formě parametru. Aplikace tyto dependence může poskytnout třídě při startu nebo je používat jen v některých funkcích. Tento proces je nazýván dependency injection a je možné díky němu vzít dependence a poskytnout je třídě, místo aby je třída získala jako instance.

V případě hudební aplikace slouží Injector k poskytnutí odkazů na databáze (konkrétně Repository) a ViewModelFactory. Jako ukázka je použit Injector, který opět souvisí s cvičením akordy (viz. Zdrojový kód 11).

```
class ChordsInjector(application: Application): AndroidViewModel(applica-
tion) {
    private fun getResultRepository(): ResultRepository {
        FirebaseApp.initializeApp(getApplication())
        return ResultRepoImpl(
            local = RoomResultDatabase.getInstance(getApplication()).room-
ResultDao())
    }

    fun provideLectureViewModelFactory(): ChordsViewModelFactory =
        ChordsViewModelFactory(
            getResultRepository()
        )
}
```

Zdrojový kód 11 - ChordsInjector

LoginInjector rozšiřuje AndroidViewModel, což je forma ViewModelu, která drží instance aplikačního kontextu. Nejprve inicializuje FirebaseApp, která je vstupním bodem do Firebase SDK. Následně vytváří metodu, která odkazuje na uživatelské Repository (tedy FirebaseUserRepository) a jako poslední poskytuje dependency na UserViewModelFactory (viz Zdrojový kód 10), který zpřístupňuje UserRepository.

4.3.4.3 Event

Posledním společným prvkem je Event. Event je reprezentace různých událostí, které můžou být propagovány z uživatelského rozhraní. Event je označován jako T. Vlastní event by měl mít každý ViewModel.

```
sealed class LoginEvent<out T> {
    object OnStart : LoginEvent<Nothing>()
    object OnAuthButtonClick : LoginEvent<Nothing>()
    data class OnGoogleSignInResult<out LoginResult>(val result:
    LoginResult) : LoginEvent<LoginResult>()

}
```

Zdrojový kód 12 - LoginEvent

LoginEvent (viz. Zdrojový kód 12) je sealed třída, což znamená, že se jedná o fixní hierarchii a nejde tvořit její podtřídy. Třída se skládá ze dvou objektů a jedné datové třídy. Tyto prvky ukazují, co bude propagováno z *LoginViewModelu*. Jedná se o události při startu, při odhlášení a při přihlášení přes Google. Co dané prvky provádí je definováno ve ViewModelu.

4.3.5 Jednotlivé funkcionality

Aplikace následuje MVVM návrhový vzor, který ve zkratce rozděluje funkcionality do View, ViewModel a Model. První dvě složky poskytují grafické rozhraní a prezentační logiku. Model poskytuje aplikační vrstvu (business logiku). V případě hudební aplikace je návrhový vzor použit v případech, kdy aplikace potřebuje poskytovat složitější funkcionalitu než pouze přechod mezi prvky navigace, ačkoliv i v takových případech by MVVM model mohl být používán.

V následující části jsou popsány jednotlivé fragmenty a aktivity aplikace v pořadí podle funkcí, stejně jako je zobrazeno v prototypu. Popsána bude obecná funkcionalita jednotlivých stránek aplikace a jejich důležitý a unikátní kód, který popisuje, jakým způsobem bylo funkcionality dosaženo.

4.3.5.1 Úvodní stránka

Úvodní stránka (viz. Příloha 1) je první stránka, ve které se uživatel ocitne. Rozhoduje se na ní, jestli se přihlásí nebo bude používat verzi bez připojení k internetu. Jestliže si zařízení pamatuje uživatelovo přihlášení, je tato stránka vynechána, uživatel je automaticky přihlášen a zpět se na ní lze dostat pomocí odhlášení uživatele z domovské stránky.

Pro přihlašování byla vybrána možnost přihlašování se pomocí Google účtu (viz. Příloha 2). Tato možnost byla zvolena, protože řeší hned několik problémů spojených s přihlašováním. Těmi jsou bezpečnost nebo například GDPR. Služba přihlašování je tímto

způsobem přenesena na jiného dodavatele, jehož zázemí je pro tuto službu více připravené (dosáhnout dokonalého zabezpečení vlastním způsobem je náročné a přesahuje rozsah diplomové práce, především protože se nejedná o primární cíl aplikace). Nevýhodou tohoto přístupu je, že ne všichni uživatelé musí mít Google účet, ačkoliv v moderních Android zařízení se přihlášení přes Google účet uživatel již téměř nevyhne. Jestliže uživatel není vlastníkem Google účtu, stále se může přihlásit v offline módu, pro který žádný účet zakládat nemusí.

4.3.5.1.1 LoginView

Třída, zajišťující View pro úvodní stránku se jmenuje LoginView a rozšiřuje android třídu *Fragment()*. LoginView vytvoří instanci svého ViewModelu (viz kapitola LoginViewModel). Následně pomocí funkce *onCreateView()* vytvoří grafické rozhraní daného fragmentu (pomocí inflate metody na xml soubor daného fragmentu). Zajímavější část přichází v *onStart()* metodě, kde je inicializovaný viewModel pomocí *ViewModelProvideru* (viz. Zdrojový kód 13).

```
LoginInjector(requireActivity().application).provideUserViewModelFac-
tory())[LoginViewModel::class.java]
setUpClickListeners()
observeViewModel()
```

Zdrojový kód 13 - Instancializace ViewModelu ve View

ViewModelProvider přijímá jako parametry vlastníka, kterým je v tomto případě daný fragment a Factory, která je poskytnuta pomocí *LoginInjector.provideUserViewModelFactory()* (viz. kapitoly Factory a Injector). Pomocí Factory spustí příslušný ViewModel.

Další zajímavá část View je *observeViewModel()* metoda, která umožňuje reagovat na LiveData z ViewModelu (více viz. Teoretická část práce). Observe zkoumá změny, které ve LiveData probíhají a reagují na ně ve View. Ukázka snadné observe metody z LoginView je například reakce na změnu přihlašovacího textu ve ViewModelu (viz. Zdrojový kód 14).

```
viewModel.signInStatusText.observe(
    viewLifecycleOwner,
    {
        loginStatusTextView.text = it
    }
)
```

Zdrojový kód 14 - Změna přihlašovacího textu pomocí observe

Ve Zdrojovém kódu 14 je nad *signInStatusText* (LiveData) ve viewModel je volána funkce observe, která přijímá parametr vlastníka, kterým je v tomto případě LifecycleOwner, reprezentující životní cyklus View. Druhou částí je observer, což je součást, která se při observe mění. Tou je v tomto případě text *loginStatusTextview*, do kterého je přidána hodnota „it“, což je hodnota MutableLiveData objektu z ViewModelu.

Druhou ukázkou je *observe()* na přihlášení (viz. Zdrojový kód 15).

```
viewModel.signedIn.observe(
    viewLifecycleOwner,
    {
        if (it) {
            startResultActivity()
            requireActivity().onBackPressedDispatcher.addCallback(this) {
                startResultActivity()
            }
        }
        else if (loginStatusTextView.text  == SIGNED_IN) {
            startSignInFlow()
        }
    }
)
```

Zdrojový kód 15 - Observe na přihlášení

Observe ve Zdrojovém kódu 15 reaguje na změnu *viewModel.signedIn* (boolean). Ta je upravena pomocí přihlášení ve ViewModelu a při změně na true znamená přesun na result aktivitu pomocí metody *startResultActivity()*. Jestliže uživatel přihlášený není a přihlašovací text byl právě změněn na SIGNED_IN (textová konstanta), je zahájen *startSignInflow()* (viz. Zdrojový kód 16).

```
private fun startSignInFlow() {
    val gso =
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestIdToken(getString(R.string.default_web_client_id))
        .build()

    val googleSignInClient = GoogleSignIn.getClient(requireActivity(), gso)

    val signInIntent = googleSignInClient.signInIntent
    startActivityForResult(signInIntent, RC_SIGN_IN)
}
```

Zdrojový kód 16 - StartSignInFlow() metoda pro přihlášení uživatele

Funkce *startSignInFlow()* spouští standardizovaný Google sign in. Pomocí *GoogleSignIn.getClient* následně přijme uživatelské informace a jestliže jsou správné, započne přihlašování.

Obdobné volání funkce observe platí také pro ostatní části a souvisejí většinou se změnou textu, animacemi nebo automatickým přihlašováním v případě přihlášeného

uživatele. Poslední zmíněnou částí v LoginView je *onActivityResult()* metoda (viz. Zdrojový kód 17). Tato metoda slouží k získání výsledků aktivity, jakmile skončí.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: In-
tent?) {
    super.onActivityResult(requestCode, resultCode, data)

    val task = GoogleSignIn.getSignedInAccountFromIntent(data)
    var token: String? = null

    try {
        val account: GoogleSignInAccount? = task.getResult(ApiExcep-
tion::class.java)

        if (account != null) token = account.idToken

    } catch (exception: Exception) {
        Log.d("Přihlášení", exception.toString())
    }

    viewModel.handleEvent(
        LoginEvent.OnGoogleSignInResult(
            UserLoginResult(
                requestCode,
                token
            )
        )
    )
}
```

Zdrojový kód 17 - OnActivityResult() metoda

Funkce ve Zdrojovém kódu 17 nejprve získá informaci o přihlášeném uživateli. Jestliže tato data nejsou nulová, nastaví je jako token. V opačném případě hodí chybovou hlášku. Následně je spuštěna funkce *OnGoogleSignInResult* specifikovaná v LoginEvent, která nastaví hodnoty *UserLoginResult* (data wrapper pro výsledky přihlašování) číslo požadavku a uživatelský token.

4.3.5.1.2 LoginViewModel

LoginViewModel, který rozšiřuje BaseViewModel, vytváří instanci *UserRepository* a *CoroutineContextu*. Následně je vytvořeno několik MutableLiveData, která jsou spojena s prvky uživatelského rozhraní (UI binding) a funkce, které MutableLiveData vyplňují podle eventu, který je registrován. Jedná se například o chybové provedení, úspěšné přihlášení nebo úspěšné odhlášení.

Nutnou metodou, kterou třída dědí z *BaseViewModelu* je *handleEvent*. Ta reaguje na uživatelské akce a provádí podle nich příslušné funkce. Metoda *handleEvent* (viz. Zdrojový kód 18) má většinou podobnou strukturu a přímo reaguje na metody implementované v *LoginEventu* (viz. kapitola Event).

```
override fun handleEvent(event: LoginEvent<UserLoginResult>) {
    showLoadingState()
    when (event) {
        is LoginEvent.OnGoogleSignInResult -> onSignInResult(event.result)
        is LoginEvent.OnAuthButtonClick -> onAuthButtonClick()
        is LoginEvent.OnStart -> {getUser()
        }
    }
}
```

Zdrojový kód 18 - Login handleEvent()

V metodě ve Zdrojového kódu 18 se jako první spustí metoda *showLoadingState()*, která nastaví hodnoty úvodní stránky. Následně je pomocí switche (v kotlinu when) zachycena akce v momentu provedení. První akce probíhá při startu a volá metodu *getUser()* (viz. Zdrojový kód 19).

```
private fun getUser() = launch {
    val result = userRepo.getCurrentUser()
    when (result) {
        is GeneralResult.Value -> {
            userState.value = result.value
            if (result.value == null) showSignedOutState()
            else showSignedInState()
        }
        is GeneralResult.Error -> showErrorState()
    }
}
```

Zdrojový kód 19 - Metoda getUser() pro získání přihlášeného uživatele

Metoda ve Zdrojovém kódu 19 slouží k získání hodnot nynějšího přihlášeného uživatele. Nejprve získá z *UserRepository* hodnotu uživatele a následně reaguje na její výsledek. Jestliže akce proběhne úspěšně, nastaví hodnotu MutableLiveData na hodnotu uživatele. Jestliže je hodnota nulová, žádný uživatel přihlášený není a jsou zobrazeny odpovídající informace. Jestliže je uživatel přihlášený, jsou zobrazeny informace potvrzující nález uživatele. Jestliže uživatel nebyl nalezen, ukáže se chybová hláška. To může nastat například pokud uživatel v době pokusu o přihlášení není připojen k internetu.

OnAuthButtonClick() nastává, jestliže uživatel klikne na tlačítko přihlášení. Tato metoda je složená z jednoduchého if cyklu. Jestliže uživatel není přihlášen, *authAttempt* MutableLiveData jsou změněna a v *LoginView* je v reakci na to spuštěna metoda startSignInFlow() (viz. kapitola LoginView). Jestliže je uživatel na úvodní stránce a uživatel již přihlášen je, je možné se pomocí totožného tlačítka odhlásit. Odhlašovací funkce je obdobná té pro *getUser()* (viz. Figure 38).

Poslední částí je *OnGoogleSignInResult()* (viz. Zdrojový kód 20). Ten reaguje na přihlášení v uživatelské stránky.

```
private fun onSignInResult(result: UserLoginResult) = launch {
    if (result.requestCode == RC_SIGN_IN && result.userToken != null) {
        val createGoogleUserResult = userRepo.signInGoogleUser(
            result.userToken
        )
        if (createGoogleUserResult is GeneralResult.Value) getUser()
        else showErrorState()

        signedIn.value = true
    } else {
        showErrorState()
    }
}
```

Zdrojový kód 20 - OnSignInResult() metoda po přihlášení

Metoda *onSignInResult()* zapne novou coroutine a na ní spouští svou funkci. Jestliže je requestcode *UserLoginResult* roven přihlašovací hodnotě a uživatelova data nejsou nulová, je v *UserRepository* započato přihlašování uživatele do Google účtu (viz. kapitola Firebase – uživatel) s hodnotou userTokenu. Jestliže systém zkontroluje, že uživatel účet má, uživatel je přihlášen. V posledním případě je zobrazena chyba. Následně je nastaveno signedIn na true, což umožňuje automatické přesunutí na úvodní stránku po zapnutí aplikace již přihlášeného uživatele.

4.3.5.2 Domovská stránka a stránka lekcí

Tyto dvě stránky (viz. Příloha 3 a 5) obsahují málo pokročilé funkcionality, takže pro jejich implementaci nebyl vytvořen ViewModel a mají tak pouze View. V aplikaci slouží pouze jako rozcestí při průchodu aplikací. Téměř jediná funkcionalita, kterou stránky mají, je možnost kliknutí na tlačítko pro přesun do jiného fragmentu. Zajímavostí je, že domovská stránka slouží jako báze pro navigation.

Užitečnou netriviální funkcionalitou je zobrazení, jestli je uživatel přihlášen nebo ne (viz. Příloha 3). To lze vidět z textu na domovské stránce, do kterého je zobrazeno jméno a příjmení uživatele, které je vyčteno z Google účtu. Jestliže je hodnota uživatele nulová (offline mód), je hodnota textu nastavena na „Bez přihlášení“. Druhou užitečnou funkcionalitou je možnost odhlášení, která po potvrzení převede uživatele na úvodní stránku a změní jeho informace do úvodního stavu.

4.3.5.3 Vysvětlující stránky O aplikaci a Pravidla

Stejně jako předchozí stránky, ani tyto nemají pokročilou funkcionalitu a nevyžadují přísnou implementaci MVVM návrhového vzoru. Stránka O aplikaci (viz. Příloha 4) zobrazuje uživateli jednoduché informace o aplikaci, které jsou nahrané v textu. Mimo jiné také zobrazuje snímek, který ukazuje, jak aplikaci používat.

Stránka Pravidla (viz. Příloha 6) funguje na obdobné bázi. Tato stránka slouží k popisu jednotlivých cvičení a k pochopení, jak je používat. I tato stránka obsahuje pomocné snímky, které určují, jak cvičení používat. Jelikož text popisující cvičení je relativně detailní, popis všech cvičení by se nevešel na jednu stránku. Kvůli tomu jsou data měněna pomocí android spinneru, který umožňuje vybrat které cvičení má být popsáno a vysvětleno.

4.3.5.4 Stránka Noty (vybrnkávání)

Prvním popisovaným cvičením je cvičení Noty (viz. Příloha 7), které slouží k vybrnkávání pravou nebo levou rukou. Obrazovka se skládá z několika sekcí. Zaprvé obsahuje několik tlačítek, které slouží především k posunu na a z fragmentu. Dále obsahuje tabulku výsledků a název noty, která má být zahrána. Nejdůležitějším prvkem jsou čtyři virtuální vrchní pražce, které simulují kytaru. Každý z těchto pražců se skládá z šesti dotykových polí. Jakmile se těchto polí uživatel dotkne, aplikace zahraje příslušný tón. Jedno z těchto polí je označeno šedou barvou. To odpovídá také názvu noty, která má být zahrána. Jakmile se uživatel daného pole dotkne, šedé pole se přesune a uživatel má zahrát další notu.

S touto funkcionalitu je spojeno několik tříd. Hlavními ovládacími třídami jsou *NotesView* a *NotesViewModel*. Dalšími důležitými třídami jsou *NotesInjector* a *NotesViewModelFactory*. Krom toho jsou použity všechny prvky pro ukládání výsledků.

4.3.5.4.1 NotesView

Stejně jako u ostatních MVVM návrhových vzorů, i zde View vytváří instanci ViewModel a jeho inicializace probíhá na stejném principu jako lze vidět na Zdrojovém kódu 13 a v *onCreateView()* je také vytvořena grafika pomocí inflate.

První unikátní a důležitá část pro samotnou funkcionalitu not se nachází v metodě *onViewCreated(),* ve které je specifikovaná funkcionalita spojená s dotykem virtuální kytary (viz. Zdrojový kód 21)

```
views.forEach { (viewId, note) ->
    view.findViewById<View>(viewId)?.setOnTouchListener { view, event ->
        viewModel.noteTouched(
            note,
            touched = event.action != MotionEvent.ACTION_UP
        )
        view.performClick()
        true
    }
}
```

Zdrojový kód 21 - ForEach cyklus pro registrování dotyků virtuální kytary

Zdrojový kód 21 popisuje forEach cyklus, procházející všechny views. Views kombinuje (pomocí mapování) dvojice jednotlivých dotykových součástí virtuální kytary (view) s notami, které jsou opět definovány dvojicí tónů (raw resource – mp3 zvuk) a integeru tónu. Pro každou view nastaví *setOnTouchListener*, který při doteku dané view spustí z *NotesViewModel* funkci spojenou s podržením místa. *NotesViewModel* následně volá funkci *noteTouched()* s parametry noty a touched. Nota je opět spojená s dotčeným místem a touched je boolean, který označuje, že plocha byla dotčena (v tomto případě nebyla puštěna, aby bylo bráno i podržení). *View.performClick()* je nutná funkce, která ukončuje akci a *setOnTouchListener()* je bez ní chybový.

Následně je v této funkci volána funkce *soundPoolInitialize()*, která slouží k vytvoření instance knihovny SoundPool, která k přehrává zvuky. Zvuk může být pomocí SoundPool hrát i vícekrát naráz. Mimo to také volá metodu observeViewModel(), která slouží k vytvoření observe funkcí. V neposlední řadě také obsahuje *handleEvent()* metodu z ViewModel, která je vysvětlena v kapitole NotesViewModel.

V *observeViewModel()* se nachází observe (viz. Zdrojový kód 22), který zajišťuje hlavní funkcionalitu spojenou s grafickým rozhraním.

```
viewModel.noteState.observe(
        viewLifecycleOwner
    ) { noteState ->
        views.forEach { (viewId, note) ->
            if (noteState.notesTouched.any { it.note == note }) {
                playSound(note)
                if (note != noteState.targetNote) {

view?.findViewById<View>(viewId)?.setBackgroundColor(Color.YELLOW)
                } else {

view?.findViewById<View>(viewId)?.setBackgroundColor(Color.GREEN)
                    noteText.setTextColor(Color.GREEN)
                }
            } else if (noteState.targetNote == note) {

view?.findViewById<View>(viewId)?.setBackgroundColor(Color.GRAY)
            } else {

view?.findViewById<View>(viewId)?.setBackgroundColor(Color.TRANSPARENT)
            }
        }
        if (noteState.targetAccomplished) {
            noteText.setTextColor(Color.GREEN)
            displayToast()
            vibrate(millisecond = 5)
        } else {
            noteText.setTextColor(Color.RED)
            noteText.text = noteState.targetNote.toString()
        }
    }
```

Zdrojový kód 22 - ViewModel.NoteState observe funkcionalita

Jak funguje observe je zmíněno již v kapitole LoginView. Observe ve Zdrojovém kódu 22 reaguje na MutableLiveData noteState, které jsou typu UiState, což je data třída, která se skládá ze setu dotčených not Set<ButtonState> (vlastní hodnoty not), noty, která je cílem a je typu Note a výsledku cíle (Boolean). Jestliže se noteState jakkoliv změní, observe pro každou z views, která byla dotčená zahraje příslušný tón pomocí funkce *playSound()* funkce. Jestliže dotčená nota neodpovídá cílové notě, nastaví se její pozadí na žlutou. V opačném případě na zelenou. Jestliže se hodnota jedné z not rovná hodnotě targetNote (cílové noty), je nastavena na šedou barvu, a tudíž označena jako další hraná nota. Všechny ostatní noty jsou transparentní. Jestliže bylo dosaženo cíle (dotčena cílová nota). Text noty se změní na zelenou barvu, ukáže se hláška o správném provedení a mobil 5 milisekund vibruje. Jestliže tomu tak není, text je červený a zobrazuje hodnotu cílové noty.

Další observy slouží především k naplnění textů „výsledky cvičení“. Pro představu, obdobnou funkcionalitu lze vidět ve Zdrojový kód 14. Jedním zajímavým, ač krátkým observem je observe, který při ukončení cvičení uloží data, nicméně i jeho funkcionalita funguje na podobném principu jako zbytek observe funkcí.

V neposlední řadě funkce obsahuje metodu *setUpClickListener()*, která pouze umožňuje posun mezi fragmenty pomocí tlačítek.

4.3.5.4.2 NotesViewModel

NotesViewModel obsahuje mezikrok mezi modelem a view. Obsahuje instanci *ResultRepository*, která slouží pro uložení dat a rozšiřuje *BaseViewModel* s hodnotou *LectureEvent*. Třída kvůli tomu opět musí implementovat metodu *handleEvent*, která v tomto případě slouží především k inicializaci view a uložení dat.

Důležitou složkou je data class UiState, která má i svá mutableLiveData noteState. To je detailněji popsáno v sekci *NotesView*. Hlavní řídící složkou třídy je metoda *noteTouched* (viz. Zdrojový kód 23) s parametry Note a touched(boolean). Tato je volána po dotčení jakékoliv části virtuální kytary.

```
fun noteTouched(note: Note, touched: Boolean) {
    val currentState = noteState.value!!
    val updatesTouchedNotes = currentState.notesTouched.toMutableSet()
    val exists = updatesTouchedNotes.any { it.note == note }
    if (touched && !exists) {
        updatesTouchedNotes.add(ButtonState(note))
    } else if (!touched && exists) {
        updatesTouchedNotes.remove(ButtonState(note))
    } else {
        return
    }
    if (updatesTouchedNotes.isEmpty() && currentState.targetAccomplished) {
        noteState.value = UiState(randomNote(), false, emptySet())
    } else if (!currentState.targetAccomplished && updatesTouchedNotes.any
{ it.note == currentState.targetNote }) {
        addToResult()
        intNotesNumber += 1
        notesNumber.value = intNotesNumber
        noteState.value =
            currentState.copy(targetAccomplished = true, notesTouched = up-
datesTouchedNotes)
    } else {
        noteState.value = currentState.copy(notesTouched = updatesTouched-
Notes)
    }
}
```

Zdrojový kód 23 - Metoda noteTouched pro odpověď na kliknutí

Ve funkci *noteTouched* (viz. Zdrojový kód 23) je nejprve zkopírován stav mutableLiveData noteState, nad kterým se následně pracuje (protože mutableLiveData by nikdy neměly být měněny přímo, ale vždy jen zkopírováním na konci funkce). Následně je vytvořený z dotčených not mutable set se jménem updatesTouchedNotes. Jestliže má tento set alespoň jednu notu, Boolean exists je nastaven na true. Jestliže nota ještě v setu není a uživatel klikl na příslušné pole, pak je odpovídající nota přidána do setu. Jestliže pole není dotčené, ale noty v setu jsou, tak jsou odebrány. Pro vysvětlení – jelikož *onTouch* obsahuje velkou škálu doteků, které se mohou stát, kterýkoliv z nich tuto metodu může spustit. Tato metoda by ale měla pouze rozdělit doteky do dvou typů, kterými je puštění tlačítka a zbytek.

Jestliže uživatel opustí tlačítko, jeho hodnota je smazána z listu. Jestliže klikne, tak je přidán. Jestliže není touched a neexistuje, metoda se ukončí. Dále se provede rozhodování založené na tom, že jestliže je set dotčených not prázdný a byla dotčena správná nota (noteState.targetAccomplished == true), tak se jako cílová nota nastaví náhodná nota (pomocí funkce *randomNote()*, která slouží pouze k nalezení nové noty), targetAccomplished je false a set je prázdný. Jestliže cíl nebyl dosažen, ale jedna z dotčených not byla správná, *addToResult()* vytvoří nový výsledek a přidá body za úspěšné splnění. Zvýší se hodnota kliknutých not a do noteState je nahrána nynějšího hodnota prvotně upravované kopie currentState, konkrétně se změněnou hodnotou dosáhnutého cíle a dotčených not. V opačném případě je změněn pouze set dotčených not notesTouched.

Pro zajímavost je zobrazena funkce addToResult (viz. Zdrojový kód 24), která mění výsledek, který je následně uložen.

```
private fun addToResult() {
    intResult += 5
    resultState.value =
        Result(getCalendarTime(), intResult.toString(), "notes", null)
}
```

Zdrojový kód 24 - Funkce k přidání do výsledku

Metoda ve Zdrojovém kódu 24 do mutableLiveData resultState s hodnotou Result vloží data, která se rovnají funkci *getCalendarTime()*, která vytváří unikátní ID pomocí nynějšího data v nynější časové zóně, dále hodnoty výsledku, typu a uživatele.

Tento výsledek je při ukončení uložen do databáze pomocí funkce *updateResult* (viz.Zdrojový kód 25)

```
private fun updateResult(contents: String) = launch {
    val updateGeneralResult = resultRepo.updateResult(
        result.value!!
            .copy(score = contents)
    )

    when (updateGeneralResult) {
        is GeneralResult.Value -> updatedState.value = true
        is GeneralResult.Error -> updatedState.value = false
    }
}
```

Zdrojový kód 25 - Přidání hodnoty do databáze

Ve Zdrojovém kódu 25 je vytvořena pomocná hodnota updateGeneralResult nad kterou je volána funkce *updateResult* z *ResultRepository* s hodnotami mutableLiveData

result, kde je nové skóre. Jestliže updateGeneralResult má hodnotu, stav je upraven a znamená to, že data jsou připravena na odeslání. V opačném případě se pokračuje.

4.3.5.5 Stránka Akordy

Druhým cvičením jsou Akordy (viz. Příloha 8 a 9). Akordy slouží k procvičení tonických kombinací hraných not, kterým se říká akordy. Ty jsou předem specifikované a dají se zahrát stiskem správné kombinace na (virtuálních) pražcích. Stejně jako Noty, stránka Akordů se skládá ze dvaceti čtyř dotykových polí, které simulují struny na pražcích. Krom toho také obsahuje text na jméno akordu, statistické hodnoty a tlačítka na ukončení cvičení. Stránka umožňuje dvě možnosti lekce, které lze přepínat pomocí posouvacího tlačítka. Změna v nich je pouze v podpoře, kterou uživatel dostává. Zkušenější hráč může hrát bez pomoci, začátečník dostává nápovědu.

Jelikož funkcionalita Akordů je v některých bodech stejná s funkcionalitou Not, i zde bude detailněji popsána pouze originální funkcionalita a důležité prvky, které jsou s Akordy stejné budou pouze zmíněny.

4.3.5.5.1 ChordsView

ChordsView je ve své podstatě stejná s NotesView. Rozdílnou je pouze hlavní observe (viz. Zdrojový kód 26), který je spuštěn v reakci na dotek tlačítek.

```
viewModel.state.observe(viewLifecycleOwner, { state ->
    views.forEach { (viewId, note) ->
        if (state.buttonsTouched.any { it.note == note }) {
            lifecycleScope.launch { playSound(note) }
            val color = if (state.isChordValid) {
                Color.GREEN
            } else {
                Color.YELLOW
            }
            view?.findViewById<View>(viewId)?.setBackgroundColor(color)
        } else if (state.assistant && state.chord.notes.contains(note)) {

view?.findViewById<View>(viewId)?.setBackgroundColor(Color.LTGRAY)
        } else {

view?.findViewById<View>(viewId)?.setBackgroundColor(Color.TRANSPARENT)
        }
    }
    noteText.text = state.chord.toString()
    if (state.isChordValid) {
        noteText.setTextColor(Color.GREEN)
        displayToast()
        vibrate(millisecond = 1)
    } else {
        noteText.setTextColor(Color.RED)
    }
})
```

Zdrojový kód 26 - Observe hlavní funkce ChordsViewModel

Mimo základů z observe popsaných dříve, prochází observe ve Zdrojovém kódu 26 volaný nad mutableDataLive state opět všechny views (dvojice not a dotykových ploch). Jestliže má funkce nějaké dotknuté plochy, zahraje se na pozadí tón. Následně je vytvořena barva (color), která se nastavuje podle toho, jestli byl zahrán akord zeleně a v opačném případě žlutě. Tato barva je následně přidělena polím, kterých se uživatel dotkl. Mimo jiné je zde také inicializován asistent, který pomáhá při hře. Jestliže je zapnutý a má přiřazené noty, jsou pole s těmito notami nastaveny na šedou barvu. V opačném případě jsou transparentní.

V druhé části, která se již nestará o práci s poli, se nejprve nastavuje akord podle mutableLiveData do textové podoby. Jestliže je akord zahrán správně, barva textu se změní na zelenou, ukáže se potvrzovací textové pole a mobil zavibruje. Jestliže správně zahrán není, text akordu zůstává červený.

Kromě *observeViewModel()* má ChordsView také *onViewCreated* metodu, která inicializuje ViewModel, posuvné tlačítko a dotykové plochy. Mimo to má také metody na klikání tlačítek, hudební output, vibrace a vytvoření toastu (zobrazeného textu na ploše). Tyto metody nebudou popsány, neboť jsou detailněji popsané v kapitole NoteView.

4.3.5.5.2 ChordsViewModel

Stejně jako ChordsView, i ChordsViewModel je podobný NotesViewModelu. Je v něm vytvořeno několik mutableLiveData, která řídí změny v ChordsView. Hlavními z nich je state hodnoty data class State, která se skládá z hodnoty akordu, booleanu isChordValid, jenž určuje, jestli byl zahrán akord, buttonsTouched, což je set stavů tlačítek, ve kterém jsou nahrané všechny dotčené plochy, chordplayed, který umožňuje změnu hraného akordu a assistant, který určuje použití asistenta. Hlavní funkcí je *buttonTouched* (viz. Zdrojový kód 27), která je volána při doteku jedné z herních ploch.

```
fun buttonTouched(note: Note, touched: Boolean) {
    val currentState = state.value!!
    val updatesTouchedButtons = currentState.buttonsTouched.toMutableSet()
    val exists = updatesTouchedButtons.any { it.note == note }
    if (touched && !exists) {
        updatesTouchedButtons.add(ButtonState(note))
    } else if (!touched && exists) {
        updatesTouchedButtons.remove(ButtonState(note))
    } else {
        return
    }
    val isChordValid = isChordValid(currentState.chord,
updatesTouchedButtons)
    var newChord: Chord? = null
    val chordPlayed = if (isChordValid) {
        true
    } else if (updatesTouchedButtons.isEmpty() && currentState.chordPlayed)
{
        newChord = randomChord()
        false
    } else {
        currentState.chordPlayed
    }
    state.value = currentState.copy(
        isChordValid = isChordValid,
        buttonsTouched = updatesTouchedButtons,
        chordPlayed = chordPlayed,
        chord = newChord ?: currentState.chord
    )
}
```

Zdrojový kód 27 - Metoda buttonTouched volaná po stisknutí pole

Stejně jako v hlavní metodě v kapitole NotesViewModel, i ve Zdrojovém kódu 27 je nejprve vytvořena kopie hodnot mutableLiveData a mutable set zmáčknutých ploch. Jestliže má mutable set alespoň jedno tlačítko, označuje se boolean exists jako true. Jestliže dané dotčené pole ještě v setu není, přidá se do mutable setu tlačítek. Jestliže není zmáčknuto a existuje, odebere se. V posledním případě, ve kterém není zmáčknuto a neexistuje se funkce ukončuje. Tato část metody odpovídá kapitole NotesViewModel.

V dalším kroku je zkoumána boolean funkce isChordValid (viz. Zdrojový kód 28), která udává validitu akordu pomocí currentState hodnoty akordu a stisknutých polí.

```
private fun isChordValid(chord: Chord, buttons: Set<ButtonState>): Boolean
{
    val isValid = buttons.size == chord.notes.size && buttons.map { it.note
}.containsAll(chord.notes)
    if (isValid) {
        addToResult()
        intChordNumber += 1
        chordsNumber.value = intChordNumber
    }
    return isValid
}
```

Zdrojový kód 28 - Metoda isChordValid

V metodě ve Zdrojovém kódu 28 se nejprve nastavuje boolean isValid, který určuje, jestli se délka setu not (data class *ButtonState*) rovná délce not v akordech a jestli všechna tlačítka ze setu not mají také noty z akordů. Jestliže je isValid pravda, přidá se pomocí funkce *addToResult()* (viz. Zdrojový kód 24) body a posune se počet správně zahraných akordů o jeden. Jako poslední je vrácena hodnota isValid.

V metodě *buttonsTouched* (viz. Zdrojový kód 27) se dostane tato metoda hodnoty not akordu a stisknutých tlačítek. Jestliže se tyto hodnoty rovnají, znamená to, že akord je validní. Dále je vytvořena proměnná newChord s nulovou hodnotou a hodnota boolean chordPlayed, která určuje, jestli byl zahrán akord pomocí funkce *isChordValid*. Jestliže akord správně zahrán není, nebylo kliknuto na tlačítka a chordPlayed hodnota data class *state* se rovná true je zvolen nový akord jako příští, který má být zahraný a chordPlayed je nastavený na false. V posledním případě je vráceno currentState.chordPlayed. V závěrečné části funkce jsou do state.value nahrány hodnoty z currentState. Jestliže funkce proběhla se správným zahráním akordu, je hodnota akordu nastavena na newChord, jinak zůstává stejná.

Pro vysvětlení je ještě dodáno, jak funguje asistent (viz. Zdrojový kód 29)

```
fun assistantSet(isChecked: Boolean) {
    val assistant = state.value?.assistant
    if (assistant == false && isChecked) {
        state.value = state.value?.copy(assistant = true)
    } else if (assistant == true && !isChecked) {
        state.value = state.value?.copy(assistant = false)
    }
}
```

Zdrojový kód 29 - Metoda assistantSet

Metoda ve Zdrojovém kódu 29 pouze překopíruje hodnotu assistantu ze mutableLiveData state a následně zkontroluje, jestli je nepravda a jestli byl posunut switch (isChecked). Jestliže tomu tak je, je jeho hodnota nastavena na pravdivou, v opačném případě je nepravdou.

Ukládání do databáze a ostatní funkce fungují na stejném principu jako v kapitole NotesViewModel.

4.3.5.6 Stránka Rytmus

Poslední stránkou cvičení je rytmus (viz. Příloha 10 a 11). Rytmus je založen na hlavní ploše, která snímá fling, což je rychlé přejetí prstem. Toto přejetí prstem simuluje pohyb pravé ruky na kytaře. Aplikace snímá dva pohyby a tím je pohyb nahoru a dolu. Stejně jako při reálném hraní na kytaru, i v hudební aplikaci se jedná o akce, které tvoří rytmus při hře na kytaru. Lekce Rytmus umožňuje vybrat ze seznamu rytmus, který chce uživatel hrát. Rytmy jsou reprezentovány šipkami, jejichž směr uživatel musí kopírovat. Jestliže je akce provedena správně, je změněna barva šipky a při špatném je lekce resetována. Zároveň také plocha, která zachytává fling, označuje, jakým směrem byl proveden a jestli byl validní.

Ačkoliv funkcionalita Rytmu se od předchozích lekcí liší, v některých bodech může být podobná nebo stejná. Proto bude zmíněna především originální funkcionalita zajišťující fungování hlavního účelu cvičení. Opakující se funkcionalita bude pouze odkazovat na předchozí části.

Než bude zmíněn princip View a ViewModel, je nutné popsat třídu, bez které se funkcionalita neobejde. Tou je třída *MyGestureListener*, která rozšiřuje *GestureDetector*, jenž slouží k všeobecnému snímání pohybu prstů po ploše. Pomocí rozšíření lze upravit metody pohybu a změnit je tak, aby odpovídaly účelům aplikace. Jelikož je rozšiřována

subsekce *SimpleOnGestureListener()*, lze upravit pouze jednu z těchto metod a tou je v případě hudební aplikace metoda *onFling()* (viz. Zdrojový kód 30).

```
override fun onFling(
    e1: MotionEvent?,
    e2: MotionEvent?,
    velocityX: Float,
    velocityY: Float
): Boolean {
    var result = false
    var direction: RhythmViewModel.UiState.Direction? = null
    try {
        if (e1 != null && e2 != null) {
            if (abs(e1.x - e2.x) > swipeMaxOffPath) {
                result = false
            }
            if (e1.y - e2.y > swipeMinDistance
                && abs(velocityX) > swipeThresholdVelocity
            ) {
                slides += 1
                direction = RhythmViewModel.UiState.Direction.UP
                result = true
            }
            else if (e2.y - e1.y > swipeMinDistance
                && abs(velocityX) > swipeThresholdVelocity
            ) {
                slides += 1
                direction = RhythmViewModel.UiState.Direction.DOWN
                result = false
            }
        }
    }
    catch (e: Exception) {
        println("Didn't work")
    }
    direction?.let {
        viewModel.onFling(it)
    } ?: viewModel.onIncorrect()
    viewModel.slidesNumber.value = slides
    return result
}
```

Zdrojový kód 30 - OnFling() metoda

OnFling() metoda (viz. Zdrojový kód 30) je typu boolean a má čtyři parametry, které reprezentují akci, která začala pohyb, akci, která ji ukončila (e1 a e2) a rychlost pohybu. Metoda nejprve kontroluje, jestli pohyb není nulový. Jestliže ne, jsou tři možnosti: Jestliže je odečtení pohybů po ose X vyšší, než předem definována konstanta toho, co je ještě bráno jako správný pohyb, vrací metoda false. To znamená, že jestliže se uživatel pokusí o pohyb zprava do leva nebo naopak, virtuální kytara nebude hrát. Jestliže je začátek pohybu na ose Y výše než konec, překonává minimální vzdálenost a zároveň je rychlost po ose X vyšší než konstantní minimální rychlost, přidá se jeden slide (mutableLiveData), směr pohybu (direction) je označen jako nahoru a metoda se rovná true. Jestliže je začátek na ose Y níže než konec na ose Y a zároveň, překonává minimální vzdálenost a požadavek na rychlost, přidá se slide. Směr je označen jako dolů a výsledek je false. V poslední řadě je podle

výsledku result nastaven onFling ve ViewModel a počet slidů se zapíše do vlastních liveData.

4.3.5.6.1 RhythmView

Mimo instancí, které jsou vytvářeny i ve View ostatních lekcí je v RhythmView instancován *GestureDetector* (kvůli *onFling*). Dále je v metodě *onViewCreated()* vytvořen spinner pro výběr akordu. Pole pro hraní je nastaveno na šedou barvu a je inicializován *GestureListene*r.

Hlavní logiku třídy řídí opět metoda *observeViewModel(),* která se sleduje změny v RhythmViewModelu. Většina observe jsou opět podobná předchozím a slouží především k změnám textu, přidávání do databáze nebo zobrazování zpráv uživateli. Proto je zmíněn pouze hlavní observe (viz. Zdrojový kód 31), který řídí funkcionalitu.

```
viewModel.uiState.observe(viewLifecycleOwner) { uiState ->
    updateRhythmArrows(uiState.flings)
    val lastOrNull =
        uiState.flings.lastOrNull { it.state !=
RhythmViewModel.UiState.FlingState.START }
    val stateOfLast =
        lastOrNull?.state
    if (stateOfLast == RhythmViewModel.UiState.FlingState.VALID) {
        if (lastOrNull.direction == RhythmViewModel.UiState.Direction.UP) {
            stringField.setBackgroundColor(
                ContextCompat.getColor(
                    requireContext(),
                    R.color.GREEN
                )
            )
            playSound(initialTone)
        } else {
            stringField.setBackgroundColor(
                ContextCompat.getColor(
                    requireContext(),
                    R.color.BLUE
                )
            )
            playSound(initialTone)
        }
    } else if (uiState.errorMessage != null) {
        stringField.setBackgroundColor(
            ContextCompat.getColor(
                requireContext(),
                R.color.RED
            )
        )
    } else {
        stringField.setBackgroundColor(
            ContextCompat.getColor(
                requireContext(),
                android.R.color.transparent
            )
        )
    }
}
```

Zdrojový kód 31 - Observe uiState v RhythmView

Mimo již dříve vysvětlené funkcionality volá observe ve Zdrojovém kódu 31 *updateRhythmArrow*, který pro každý potenciální fling v nastaveném rytmu nejprve nastaví viditelnost (jelikož stavy šipek mohou být nahoru a dolů, pro uživatele se takto vytřídí ucelený rytmus). Následně vybere stav flingu (enumerace; START, VALID, INVALID) a podle stavu vybere, jak se daná šipka zabarví. Jestliže se jedná o začátek, šipky jsou transparentní, jestliže fling odpovídá šipce, je označena zelenou. Jestliže neodpovídá, je označena červenou.

V observe (viz. Zdrojový kód 31) je následně získána hodnota posledního flingu (jestliže není jeho hodnota START) a následně je získán jeho stav (VALID nebo INVALID). Jestliže je stav validní a směr flingu byl nahoru, je hlavní plocha nastavena na zelenou a je zahrán tón (tón je vybrán náhodně, nicméně dokud není jeden rytmus dohrán, nemění se). Jestliže byl směr dolů, je plocha modrá. Jestliže fling validní nebyl (a tím pádem chybová hláška není nulová), změní se hlavní plocha na červenou.

Mimo *observeViewModel()* také třída obsahuje metody k vytváření potvrzovacích a chybových správ pro uživatele, nastavování funkcionality tlačítek nebo metody pro hraní tónů. Třída je také založena na eventech, nicméně ty se nijak neliší od již zmíněných a slouží tedy především k vytvoření úvodního zobrazení a ukládání do databáze.

4.3.5.6.2 RhythmViewModel

I RhythmViewModel obsahuje prvky předchozích ViewModel. I zde je vytvořeno několik MutableLiveData, která slouží k aktivnímu ukládání hodnot. Nejdůležitějším z nich je uiState, který obsahuje data class tvořenou z rhythmType (typ rytmu), flings (list flingů; fling je dataclass, které se skládá ze směru a flingState, který byl popsán dříve), errorMessage, successMessage (oba jsou stringy držící informace o provedení) a boolean tryAgain, který označuje chybné provedení. Hlavní řídící metodou třídy je metoda *onFling()* (viz. Zdrojový kód 32).

```
fun onFling(direction: UiState.Direction) {
    val currentState = _uiState.value
    var flingFound = false
    var isValidFling = true
    val mappedFlings = currentState!!.flings.map {
        if (flingFound || it.state != UiState.FlingState.START) {
            it
        } else {
            flingFound = true
            if (it.direction == direction) {
                it.copy(state = UiState.FlingState.VALID)
            } else {
                isValidFling = false
                it.copy(state = UiState.FlingState.INVALID)
            }
        }
    }
    when {
        currentState.flings.all { it.state == UiState.FlingState.VALID } ->
{
            _uiState.value = UiState(currentState.rhythmType,
successMessage = "Správný rytmus")
            addToResult()
        }
        isValidFling -> {
            _uiState.value = currentState.copy(
                flings = mappedFlings,
                successMessage = null,
                errorMessage = null,
                tryAgain = false
            )
        }
        else -> {
            _uiState.value = UiState(currentState.rhythmType, errorMessage
= "Byl zahrán špatný rytmus")

        }
    }
}
```

Zdrojový kód 32 - OnFling() metoda ve ViewModel

Metoda *onFling()* ve Zdrojovém kódu 32 je volána z *MyGestureListener* a nejprve zkopíruje stav uiState a vytvoří boolean proměnné flingFound a isValidFling, které slouží k označení, jestli byl proveden fling a v jaké podobě. Vytvoří také proměnnou mappedFlings, která slouží k držení listu flingů podle následujícího klíče. Pokud je flingFound pravda a stav uiState je START, vrátí se příslušný fling. V opačném případě je flingFound nastaven na true a jestliže určený směr se rovná zadanému tvaru, stav flingu je označen za validní. V opačném případě je isValidFling nastaveno na false a stav je INVALID.

Jestliže jsou všechny flings ve zkopírovaném stavu validní, je do mutableLiveData uiState zkopírován rytmus a je nastavena potvrzovací hláška. Jestliže je fling správný, jsou do uiState přidány všechny uložené flings, zprávy jsou nulové a tryAgain je false. To znamená, že všechno probíhá podle plánu, nicméně uživatel ještě rytmus nedokončil.

Poslední případ znamená, že uživatel udělal chybu a v tom případě je zobrazena chybová hláška.

Unikátní jsou také metody *changeRhythm()* a *onIncorrect()* (viz. Zdrojový kód 33).

```
fun changeRhythm(rhythmType: RhythmType) {
    _uiState.value = UiState(rhythmType)
}
fun onIncorrect() {
    _uiState.value = _uiState.value?.copy(tryAgain = true)
}
```

Zdrojový kód 33 - ChangeRhythm() a onIncorrect() metody

Metoda *changeRhythm()* je volána ve View při změně hodnoty rytmu ve spinneru. Do uiState nastaví hodnotu typu rytmu na zadanou hodnotu. *onIncorrect()* je volána v případě, že onFling neměla zvolenou direction (byl proveden chybný pohyb). Ta pouze nastaví, že uživatel by měl zahrát znovu.

4.3.5.7 Statistiky

Posledními popisovanými stránkami jsou statistické stránky. Ty jsou dvě, první jsou výsledky lekcí (viz. Příloha 12) a druhou jsou obecné statistiky (viz. Příloha 13). Funkce obou těchto stránek je pouze informativní a motivující, a slouží k poskytnutí jak krátkodobých, tak dlouhodobých statistik a výsledků. K tomu je využívána databáze, což znamená, že mimo View a ViewModel obsahují obě tyto stránka i Injector (viz. kapitola Injector) a ViewModelFactory (viz. kapitola Factory).

Stránka s výsledky cvičení se uživateli zobrazí po dokončení lekce (kliknutí na ukončující vlajku ve hře) a obsahuje výsledek dohraného cvičení v porovnání s nejvyšším výsledkem daného cvičení. Mimo to také obsahuje možnost vrátit se na úvodní stránku. Jelikož stránka je pro všechny cvičení totožná, dochází k výběrovému procesu pro ukázání správného posledního cvičení. Ve ViewModel jsou nastavena mutableLiveData s druhem odehraného cvičení, nad kterými je následně ve View observe, který změní příslušné texty.

Stránka se statistikami obsahuje informací více. Informace se zde dělí do čtyř kategorií podle jednotlivých her + celkových výsledků. Výsledky her se skládají z posledního výsledku, nejlepšího výsledku a celkového výsledku. Celkový výsledek je součet celkových výsledků předchozích funkcí. Důležitým ošetřením je volání do databáze, která ještě dané hodnoty nemá.

Jelikož funkcionalita dosazování výsledků je ve své podstatě duplicitní, bude zobrazena pouze jednou. Pro zobrazení je použita část funkce *getResults()* z *StatisticsViewModel*, která slouží k získání výsledků a následné práci s nimi.

```
                try {
                        val notesResults =
                            resultsResult.value.filter { result: Result ->
result.type == "notes" }
                        val currentNotesState = notesResultState.value
                        val notesLastResult = notesResults.last().score
                        val notesBestResult =
                            notesResults.map { it.score.toInt()
}.sorted().lastOrNull().toString()

                        val notesCompleteResult =
                            notesResults.sumOf { result: Result ->
result.score.toInt() }.toString()
                        notesResultState.value = currentNotesState?.copy(
                            notesLastValue = notesLastResult,
                            notesBestValue = notesBestResult,
                            notesCompleteValue = notesCompleteResult,
                            notesErrorState = 1
                        )
                    } catch (e: Exception) {
                        notesResultState.value?.notesErrorState = 0
                    }
```

Zdrojový kód 34 - Try/catch blok pro vložení dat z databáze

Kód ze Zdrojového kódu 34 slouží k zapsání výsledků cvičení Noty. Try/catch ošetření je zvoleno právě kvůli výše zmíněným problémům při ještě nevytvořených hodnotách v databázi. Nejprve je vytvořena hodnota notesResults, do které jsou nahrány všechny výsledky typu noty. První získanou hodnotou je poslední hodnota not, která je volána metodou *last()*, jež získá poslední vytvořený prvek. Následně je pomocí seřazeného listu vybrán poslední prvek, který je nahrán jako nejlepší výsledek. Jako celkový výsledek je vybrána suma skóre. Posledním krokem je kopírování dat do mutableLiveData s notesErrorState 1, který určuje, že nahrání proběhlo úspěšně.

Podobně probíhá i vytvoření ostatních filtrovaných hodnot. MutableLiveData těchto hodnot jsou sledována a následně nahrána ve formě textu do textových polí. Výsledkem jsou aktuální hodnoty, které zobrazují funkcionalitu.

4.4 Testovací fáze

Testovací fáze slouží k zajištění kvality aplikace. Testování kódu proběhlo dvěma způsoby. První z nich bylo testování funkcionality a odpovídající kvality během vývoje. Druhým je funkční testování, které slouží k otestování, jestli aplikace v dostatečné kvalitě zajišťuje dodržení funkčních požadavků. Slouží především k ověření všech povinných částí.

Pro toto testování bylo vytvořeno několik test případů, které jsou postaveny na předem specifikovaných funkčních požadavcích a jejich průchod probíhá podle úspěšného i alternativního scénáře. Tyto testy zkoušely po přečtení požadavků 3 osoby, každá s jinou úrovní informovanosti o aplikaci.

4.4.1 UX testování a testování funkčních požadavků

4.4.1.1 Práce s účtem

Jelikož implementace je provedena jedním způsobem, testování obou způsobů by bylo redundantní.

Tabulka 11 - Testování práce s účtem

Typ testu:	Verifikační
Čas:	2 minuty
Testovací data:	Přihlášení: gmail uživatele Registrace: testovacidata2352@gmail.com Heslo: 23TeStoVaciUdaje53 Nové heslo: 1. čzunove 2. 23noveHeslo234
Provedení testu (průchodu):	Úspěch kromě změny hesla, kde bylo schválně zadáno nesprávné heslo
Poznámky:	I přes provedení mírně pozměněným způsobem se povedlo dosáhnout daného výsledku a uživatel se přihlásil, založil si účet a změnil si heslo.

4.4.1.2 Zobrazení a změna osobních informací

Jelikož funkcionalita je outsourcována, testování sice je provedeno, nicméně netestuje vytvořenou aplikaci.

4.4.1.3 Cvičení

Test byl proveden pro všechna tři cvičení

Tabulka 12 - Testování cvičení

Typ testu:	Verifikační
Čas:	1 minuta zapnutí cvičení + 2 minuty používání
Testovací data:	Uživatel je přihlášen, uživatel není přihlášen
Provedení testu (průchodu):	Úspěch
Poznámky:	Bez předchozího pročtení informací má neinformovaný uživatel problémy se snadným pochopením cvičení a musí se vrátit. Informovaný a středně informovaný uživatel je schopen se za normálních podmínek dostat pod 1 minutu do hry, hra odpovídá předepsaným nefunkčním požadavkům. Dvě minuty hraní všech her bez problému. Výsledky jsou po ukončení korektně zobrazeny.

4.4.1.4 Zobrazení statistik

Zobrazení statistik po odehrání náhodného počtu cvičení, s i bez přihlášení.

Tabulka 13 - Testování zobrazení statistik

Typ testu:	Verifikační
Čas:	30 sekund
Testovací data:	Uživatel je přihlášen, uživatel není přihlášen, uživatel nemá odehraná cvičení
Provedení testu (průchodu):	Úspěch

Poznámky:	Statistiky byly korektně zobrazeny ve všech testovaných případech. Uživatel po přihlášení z jiného zařízení viděl svoje statistiky ze zařízení předchozích. Při vyšších výsledcích se skóre zobrazuje rozházeně.

4.4.1.5 Zobrazení pravidel/Zobrazení O aplikaci

Tabulka 14 - Testování zobrazení pravidel a „O aplikaci“

Typ testu:	Verifikační
Čas:	30 sekund cesta, 2 minuty na prostudování
Testovací data:	-
Provedení testu (průchodu):	Úspěch
Poznámky:	Pravidla byla vždy správně zobrazena. Uživatelé se v systému zorientovali relativně snadno a do daného časového limitu byly s pravidly srozuměni.

4.4.1.6 Odhlášení

Odhlášení není plně zajištěno Googlem, a proto má vlastní sekci.

Tabulka 15 - Testování odhlášení

Typ testu:	Verifikační
Čas:	Z hlavní stránky 15 sekund.
Testovací data:	Přihlášený uživatel i nepřihlášený uživatel
Provedení testu (průchodu):	Upravit
Poznámky:	Uživatel se vždy úspěšně odhlásil a dostal se na úvodní stránku. Uživatel bez přihlášení také musel potvrdit odhlášení – Změnit.

4.4.2 Testování nefunkčních požadavků

4.4.2.1 Použitelnost

Specifikovaný cíl: Uživateli aplikace by nemělo vybrání možnosti trvat déle než 5 sekund.

Výsledek: Uživatelé neměli s průchodem aplikací problémy, nejdéle trvala možnost ukončení cvičení a odhlášení, neboť jejich označení není písemně označeno a není tak tolik průkazné. Problémy, jestliže je zařízení menší, než určuje technické omezení. Rozhodnutí trvala méně než 5 sekund.

4.4.2.2 Výkon

Specifikovaný cíl: Přepínání ve cvičení by nemělo trvat déle než 1 sekundu. Přepínání mezi obrazovkami by nemělo trvat déle než 1 sekundu.

Výsledek: Přepínání ve hře za všech podmínek a na všech zařízeních trvá méně než jednu sekundu (odpověď je okamžitá), takže cvičení funguje čistě. Přepínání mezi obrazovkami závisí na zařízení, některé (viz. emulátor) trvají na hranici jedné sekundy. Většina novějších zařízení reaguje okamžitě.

4.4.2.3 Zabezpečení a soukromí

Specifikovaný cíl: Mimo první vrstvu zabezpečení (emailová adresa a heslo) by žádná zadaná data neměla zasahovat do uživatelského soukromí. Systém by neměl mít možnost zjistit a přísně tak dodržovat GDPR. Zaměření zabezpečení bude tedy mířit především na ochranu přihlašovacích údajů. Jiné informace při případných únikách dat neposkytnout žádné podstatné informace.

Výsledek: Uživatel poskytuje pouze emailovou adresu a heslo, které jsou drženy v databázi Google. I při úniku informací z aplikace nemůže útočník nic získat. Ostatní osobní data byla omezena, aby bylo přísně dodrženo GDPR.

4.4.2.4 Uložení dat

Specifikovaný cíl: Aplikace musí ukládat všechna data do databáze, včetně hodnocení ze cvičení uživatelů. Aplikace by měla mít k dispozici offline i online databázi a zpřístupnit

tím oba režimy používání. Ukládání do databáze by nemělo, jakkoliv zdržovat běh aplikace (mělo by trvat pod 0,5 sekundy).

Výsledek: Aplikace má dvě přístupné databáze. Práce s databází je zajištěna pomocí coroutines, takže probíhá v pozadí, aplikace není zdržena.

4.4.2.5 Škálovatelnost

Specifikovaný cíl: Systém by měl být připraven na potenciální nárůst uživatelů.

Výsledek: Nárůst uživatelů je zajištěn cloudovou databází, do které jsou uložena data přihlášených uživatelů. Škálovatelnost v offline verzi je zbytečná.

4.4.2.6 Jazyková podpora

Není zavedena.

5 Výsledky a diskuze

Pomocí vědeckých metod a dodržováním metodologie životního cyklu vývoje mobilních aplikací byla vytvořena multifunkční aplikace na OS Android, která slouží k základům výuky hry na kytaru. Tato aplikace přináší nezávislou podobu hry na kytaru, která je zároveň užitečná, ale nevyžaduje nutné vlastnictví kytary. Znalosti z aplikace jsou přenositelné do vlastní hry. Mimo samotné funkcionality související s hrou na kytaru má aplikace ostatní prvky moderních aplikací, jako jsou databáze (cloud i lokální) nebo přihlašování (přes Google účet). Aplikace byla vytvořena za pomoci moderních programovacích metod a přístupů v jazyce Kotlin. Mezi nimi stojí za zmínku především MVVM model, Firebase Authorization a Firestore, Room Database, Android Navigation atd. V následující tabulce jsou viditelné některé statistiky aplikace:

Tabulka 16 - Obecné statistiky aplikace

Počet stránek (views):	10
Počet aktivit:	2
Počet tříd:	48 + 2 pomocné
Počet řádků:	3111
Počet package:	14
Počet XML souborů:	39
Počet řádků XML souborů:	3236
Počet zvukových souborů:	25

5.1 Porovnání

Jelikož není mnoho volně dostupných aplikací, které by sdílely své kódy, obecné statistiky ani rychlost aplikace nelze s žádnou populární aplikací porovnat. Proto jsou především popsány nové funkcionality, které má hudební aplikace oproti ostatním. Toto porovnání vychází z analýzy trhu.

Jak již bylo zmíněno v analýze, většina nynějších aplikací pomáhajících s hrou na kytaru má dvě podoby: Herní podobu a podobu při které mobil slouží pouze jako výukový prvek, nicméně uživatel stále musí hudební nástroj vlastnit. Z tohoto vycházelo, že vývoj mobilních aplikací se příliš nespoléhá na samostatnou vzdělávací činnost, což bylo určeno jako hlavní cíl hudební aplikace.

Mezi hlavní výhody hudební aplikace oproti ostatním lze zařadit samostatná použitelnost, grafická jednoduchost, vzdělávací funkce a adekvátní použitelnost.

Mezi výhody ostatních aplikací nad hudební aplikací lze zařadit možnost využití placených zvukových souborů a jiných autorských děl, zařazení opakujících se populárních funkcionalit (například písní podle akordů), multiplatformní běh nebo lepší protestování a optimalizace.

5.2 Přínos

Vytvořená hudební aplikace byla na základě požadavků vytvořených za pomocí analýzy trhu, zdrojů pro vývoj moderních mobilních aplikací a zdrojů souvisejících s efektivní výukou hry na kytaru vytvořena tak, aby co nejlépe požadavky splňovala. Z toho vznikla aplikace simulující kytaru, která cvičí tři základní prvky hry na kytaru, kterými jsou vybrnkávání, akordy a rytmus. Pomocí těchto prvků může uživatel získat základní znalosti hry na kytaru, které mu nejen mohou poskytnout základy do budoucna, ale například i poradit, jestli mu hraní na kytaru vyhovuje.

K tomu, aby byla hudební aplikace označitelná za přínosnou, musí být splněny především nefunkční požadavky. Nejdůležitějšími z nich je zvolena použitelnost a výkon. Použitelnost byla dodržena jednoduchým designem, který zároveň splňuje požadavky velikosti (zajištěno pomocí nástrojů v Android Studiu) a většina jeho částí je jednoznačně popsána. Část výkonu se týká především samotných cvičení. Výkon byl dosažen především optimalizací kódu a prací s vlákny, které slouží k neblokujícímu hlavnímu vláknu. Výsledkem toho je, že cvičení doopravdy funkcí připomínají hru na hudební nástroj.

5.3 Nápady a úpravy v budoucnosti

Aplikace se potýká s několika problémy, které s nynějšími zařízeními nelze vyřešit. Hlavním příkladem, je:

- **Nemožnost hraní pokrytím celého pražce** – Nynější zařízení neumožňuje dotek, který by umožnil efektivně zmáčknout celý pražec. To znamená, že některé akordy nemohou být hrány (například akord F dur). Nicméně tyto akordy lze částečně brát za pokročilé, takže se bez nich aplikace obejde.

Mimo technické nedostatky, se kterými nelze v tomto případě nic dělat, je možné aplikaci nadále zlepšovat tak, aby její uživatelnost byla zamířena i na širší skupinu uživatelů. Mezi prvky, které by se daly přidat lze počítat:

- **Propojení s učitelem** – Aplikace by šla používat nejen jako samo vzdělávací pomůcka, ale i jako nástroj pro samostatné učitele (například na zadávání domácích úkolů, které by šly následně kontrolovat).
- **Více hudebních nástrojů** – Některé hudební nástroje samozřejmě implementovat nepůjdou. Neustále je nutné brát v potaz technická omezení. Určitě by šla přidat většina strunných nástrojů, například ukulele, banjo, mandolína.
- **Kytara není univerzální** – V této podobě poskytuje aplikace unifikovaně naměřený model kytary. Nicméně různé kytary mají různě široké hrdlo, takže některým uživatelům nemusí vyhovovat. Přidáním možnost parametrizace kytary nebo výběru ze seznamu kytar, který by následně změnil tvar virtuální kytary, by se tento problém vyřešil.
- **Velikost zařízení** – V nynější podobě se aplikace snaží pokrývat co nejširší škálu zařízení. Nicméně jestliže je zařízení příliš malé, ne všechny prvky se na plochu vejdou bez překryvu. Určením priority prvků na ploše by šlo odbourávat některé nedůležité prvky na menších zařízení tak, aby uživatel viděl jen to nejdůležitější. Toto má samozřejmě své limity, jakmile se kytara nevejde na plochu, nikdy nebude použitelná. Stejným problémem je tablet, jakmile je zařízení příliš velké, nemůže efektivně simulovat krk kytary.
- **Multiplatformita** – Nyní je zařízení pouze na OS Android, vytvořením aplikace na iOS a další operační systémy se uživatelnost zvýší.
- **Optimalizace funkcí** – Některé funkce se dají optimalizovat. Například cvičení Rytmus může být užitečnější, jestliže umožní hrát rytmy efektivnější a nekonečně. V nynější podobně po odehrání rytmu musí uživatel potvrdit konec a začít hrát od začátku. Zároveň se nyní hraje jeden tón a uživatelsky by bylo příjemnější, kdyby se dala hrát píseň (3 akordy na píseň stačí).

K samotnému vytvoření aplikace nevedlo pouze programování, ale i všechny ostatní fáze metodologie životního cyklu vývoje mobilních aplikací. Jako jednoznačně hlavní nedostatek lze brát:

- **Nedostatek metodologií pro samostatný vývoj** – Analýzou zdrojů vyšlo najevo, že není příliš metodologii, které by byly užitečné anebo přímo aplikovatelné pro vývoj aplikací jednou osobou. To znamená, že některé části se spoléhají na metody, které nejsou efektivní a často se k cíli dostávají oklikou. Příkladem je specifikace požadavků. Většina specifikací vzniká ve větší skupině jako brainstorming za pomocí stakeholderů. Náhradou pro jednou osobu je zdlouhavá série rozhovorů nebo dotazníků, pro kterou v případě hudební aplikace není velká cílová skupina, a tak je i ta málo efektivní. Použití požadavků specifikovaných na základě analýzy zdrojů, jako bylo provedeno v této práci, také nelze brát jako nejefektivnější cestu.
- **Nynější vývojové metodologie nejsou vytvořené pro vědecké práce** – Jak lze vidět i na případě této aplikace, některé složky životního cyklu by při správném zacházení zůstaly prázdné (viz. Nasazení a údržba). To proto, že technická práce jako taková má striktně stanovený konec a následný dlouhodobý stav (údržba) už nedává smysl. Vytvořením metodologie, která by měla dobrovolné části nebo byla použitelná přímo pro tyto případy, by se problém vyřešil.

6 Závěr

V první části byly představeny základy hry na kytaru, které pomáhají pochopení principu a slouží jako teoretický základ začátků. Následně byla popsána metodologie Životní cyklus vývoje mobilních aplikací, kterou se bude vývoj mobilní aplikace ubírat. Jako další byl popsán operační systém Android, byly zobrazeny technologie, které jsou pro vývoj aplikace použity a jejich alternativy a vysvětlení, proč nebyly vybrány. Mezi detailně popsané technologie se řadí Kotlin, Android Jetpack, MVVM vzor, asynchronní programování nebo Firebase. V neposlední řadě byla provedena analýza trhu, ve které byly analyzovány volně dostupné aplikace a výsledky analýzy vytvořily základy pro jedinečnost vyvíjené aplikace.

Tyto znalosti byly aplikovány při vývoji mobilní aplikace v praktické části, ve které byl nejdříve provedený výzkum, v němž byly specifikovány cíle, určena cílová skupina a požadavky a v neposlední řadě analyzovány regulační, technická a business omezení. Za pomoci znalostí z výzkumné části byl zpracován základní návrh aplikace včetně prototypu a software architektury. Na základě toho byla ve vývojové fázi vytvořena hudební aplikace, která splňuje všechny požadavky a obsahuje předem specifikovanou funckionalitu. Tato aplikace následně prošla testovací fází, ve které bylo vyzkoušeno splnění funkčních i nefunkčních požadavků základním testováním. Místo poslední fáze, nasazení/údržby, bylo z důvodů kontextu aplikace vytvořeno shrnutí výsledků a provedena diskuse.

Výsledkem vývoje mobilní aplikace je hudební aplikace v jazyce Kotlin, sloužící k samo výuce hry na kytaru. Hudební aplikace je vytvořena tak, aby obsahovala základní prvky moderních aplikací. Hlavním výukovým prvkem aplikace jsou tři cvičení, které mají za účel výuku základních hudebních dovedností, kterými jsou vybrnkávání(noty), akordy a rytmus. Provedením těchto cvičení jsou získány výsledky, které jsou následně uloženy do databáze podle způsobu uložení (cloud pro online nebo lokální pro offline).

Z diskuse vyplývá, že aplikace, oproti ostatním hudebním aplikacím, poskytuje uživateli větší samostatnost a užitečnost. Zároveň je narozdíl od nich určena čistě pro výukové účely. V diskuzi je také zmíněno několik úprav, které by bylo v budoucnosti možné provést, aby byla aplikace užitečnější. V neposlední řadě také několik technických omezení, které je nutné při vývoji obdobných aplikací brát v potaz.

7 Seznam použitých zdrojů

Abdelhadi Ali (2017) *The Firebase Blog: Introducing Firebase Performance Monitoring*, *Introducing Firebase Performance Monitoring*. Available at: https://firebase.googleblog.com/2017/05/introducing-firebase-performance.html?hl=th&m=1 (Accessed: March 13, 2022).

Adamovic Milan (2019) *Android Jetpack — Guide to the Room Persistence Library* , *Android Jetpack — Guide to the Room Persistence Library*. Available at: https://medium.com/dev4-dev/android-jetpack-guide-to-the-room-persistence-library-6c2a492faef8 (Accessed: March 13, 2022).

Adefioye Temidayo (2018) *Android Jetpack:Empower your UI with Android Data Binding*, *Android Jetpack:Empower your UI with Android Data Binding*. Available at: https://temidjoy.medium.com/android-jetpack-empower-your-ui-with-android-data-binding-94a657cb6be1 (Accessed: March 13, 2022).

Agarwal Tarun (2020) *Android Operating System : Introduction, Features & Its Applications*, *What is an Android Operating System & Its Features*. Available at: https://www.elprocus.com/what-is-android-introduction-features-applications/ (Accessed: March 13, 2022).

Altexsoft (2021) *Functional and Non-functional Requirements: Specification and Types | AltexSoft*, *Functional and Nonfunctional Requirements: Specification and Types*. Available at: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/ (Accessed: March 13, 2022).

Android Developer Relations (2022) *Background work with WorkManager - Kotlin*, *Background work with WorkManager - Kotlin*. Available at: https://developer.android.com/codelabs/android-workmanager#0 (Accessed: March 13, 2022).

Android Developers (2021a) *Data Binding Library*, *Data Bidning*. Available at: https://developer.android.com/topic/libraries/data-binding (Accessed: March 13, 2022).

Android Developers (2021b) *Design for Android*, *Design for Android*. Available at: https://developer.android.com/design (Accessed: March 15, 2022).

Android Developers (2021c) *Handling Lifecycles with Lifecycle-Aware Components | Android Developers*, *Handling Lifecycles with Lifecycle-Aware Components | Android Developers*. Available at: https://developer.android.com/topic/libraries/architecture/lifecycle (Accessed: March 13, 2022).

Android Developers (2021d) *LiveData Overview*, *LiveData*. Available at: https://developer.android.com/topic/libraries/architecture/livedata (Accessed: March 13, 2022).

Android Developers (2021e) *Platform Architecture | Android Developers*, *Platform Architecture*. Available at: https://developer.android.com/guide/platform (Accessed: March 13, 2022).

Android Developers (2022a) *Get started with Jetpack Compose* , *Get started with Jetpack Compose* . Available at: https://developer.android.com/jetpack/compose/documentation (Accessed: March 13, 2022).

Android developers (2022) *Kotlin coroutines on Android | Android Developers*, *Kotlin coroutines on Android*. Available at: https://developer.android.com/kotlin/coroutines (Accessed: March 13, 2022).

Android Developers (2022b) *Meet Android Studio | Android Developers*. Available at: https://developer.android.com/studio/intro (Accessed: March 13, 2022).

Android Developers (2022c) *Meet Android Studio | Android Developers*, *Meet Android Studio*. Available at: https://developer.android.com/studio/intro (Accessed: March 13, 2022).

Android Developers (2022d) *Notifications Overview*, *Notifications Overview*. Available at: https://developer.android.com/guide/topics/ui/notifiers/notifications (Accessed: March 13, 2022).

Android Developers (2022e) *Room | Android Developers*, *Room*. Available at: https://developer.android.com/jetpack/androidx/releases/room (Accessed: March 13, 2022).

Android Developers (2022f) *Run apps on the Android Emulator | Android Developers*, *Run apps on the Android Emulator*. Available at:

https://developer.android.com/studio/run/emulator?gclid=CjwKCAjwyIKJBhBPEiwAu7zll-T7O1p1yRXUOeeaYMqYH1c8Lp6-SwB1VAyq49jW5leTY4JncP0P8RoC1RAQAvD_BwE&gclsrc=aw.ds (Accessed: March 13, 2022).

Android Developers (2022g) *Test apps on Android*, *Test apps on Android*. Available at: https://developer.android.com/training/testing/ (Accessed: March 13, 2022).

Ari Dler (2019) *An introduction to observables in Reactive Programming*, *An introduction to observables in Reactive Programming*. Available at: https://www.freecodecamp.org/news/an-introduction-to-observables-in-reactive-programming-1cfd3e23bb94/ (Accessed: March 13, 2022).

Bamberg, Schonning Nick and Willee Hamish (2022) *Introducing asynchronous JavaScript*, *Introducing asynchronous JavaScript*. Available at: https://github.com/mdn/content/blob/main/files/en-us/learn/javascript/asynchronous/introducing/index.md (Accessed: March 13, 2022).

Banes Chris, A.D.R. (2014) *Android Developers Blog: AppCompat v21 — Material Design for Pre-Lollipop Devices!* Available at: https://android-developers.googleblog.com/2014/10/appcompat-v21-material-design-for-pre.html (Accessed: March 13, 2022).

Batschinski George (2022a) *Firebase vs other service providers*, *Firebase vs other service providers*. Available at: https://blog.back4app.com/firebase-vs/#What_are_the_best_solutions_vs_Firebase (Accessed: March 13, 2022).

Batschinski George (2022b) *What is Firebase? Secrets unlocked (pros, cons, pricing, etc)*, *What is Firebase? Secrets unlocked (pros, cons, pricing, etc)*. Available at: https://blog.back4app.com/firebase/ (Accessed: March 13, 2022).

@benzoix (2022) *Free Photo | Old black background. grunge texture. dark wallpaper. blackboard, chalkboard, room wall.* Available at: https://www.freepik.com/free-photo/old-black-background-grunge-texture-dark-wallpaper-blackboard-chalkboard-room-wall_11712558.htm (Accessed: March 13, 2022).

Biazar Reza (2018) *Man wearing blue and red collared shirt photo – Free Image on Unsplash*. Available at: https://unsplash.com/photos/eSjmZW97cH8 (Accessed: March 13, 2022).

Brambilla, M. and Fraternali, P. (2015) "Implementation of applications specified with IFML," *Interaction Flow Modeling Language*, pp. 279–334. doi:10.1016/B978-0-12-800108-0.00010-2.

Breathnach Cillian (2021) *Best software for guitarists in 2021: 10 best apps to learn how to play guitar*, *BEST SOFTWARE FOR GUITARISTS IN 2021: 10 BEST APPS TO LEARN GUITAR*. Available at: https://guitar.com/guides/buyers-guide/best-guitar-learning-apps/ (Accessed: March 13, 2022).

Brown Simon (2011) *The C4 model for visualising software architecture*, *The C4 model for visualising software architecture*. Available at: https://c4model.com/ (Accessed: March 13, 2022).

Burns Bill (2020) *What Is Multithreading and Multithreaded Applications | TotalView*, *What Is Multithreading: A Guide to Multithreaded Applications*. Available at: https://totalview.io/blog/multithreading-multithreaded-applications (Accessed: March 13, 2022).

Canda, R., Azam, Z. and Sariman, H. (2016) *The Development of FiTest for institution of higher learning using Mobile Application Development Lifecycle Model (MADLC): From Identification Phase to Prototyping Phase*. Available at: https://www.researchgate.net/publication/317058187.

Casteren, W. van and van Casteren, W. (2017) "The Waterfall Model and the Agile Methodologies : A comparison by project characteristics The Waterfall Model and the Agile Methodologies : A comparison by project characteristics Academic Competences in the Bachelor 2 assignment: Write a scientific article on 2 Software Development Models." doi:10.13140/RG.2.2.36825.72805.

Cechetto Thiago (2018) *Android Architecture Components and Jetpack Development | ArcTouch*. Available at: https://arctouch.com/blog/android-architecture-components-jetpack/ (Accessed: March 13, 2022).

Chen James (2021) *Android Operating System Definition*, *Android Operating System*. Available at: https://www.investopedia.com/terms/a/android-operating-system.asp (Accessed: March 13, 2022).

Childers Chad (2021) *Study: Who Make Up the 16 Million Learning Guitar In Last 2 Years*, *Study: Who Make Up the 16 Million Learning Guitar In Last 2 Years*. Available at: https://loudwire.com/study-who-are-16-million-americans-learned-play-guitar-last-two-years/ (Accessed: March 13, 2022).

Chopra Hitesh (2021) *ViewModel magic revealed!!!*, *ViewModel*. Available at: https://proandroiddev.com/viewmodel-magic-revealed-330476b5ab27 (Accessed: March 13, 2022).

Cid Joaquin (2020) *Build a Role-based API with Firebase Authentication*. Available at: https://www.toptal.com/firebase/role-based-firebase-authentication (Accessed: March 13, 2022).

Clegg Dai (2020) *What is MoSCoW Prioritization? | Overview of the MoSCoW Method*, *MoSCoW Prioritization*. Available at: https://www.productplan.com/glossary/moscow-prioritization/ (Accessed: March 13, 2022).

@cookie_studio (2022) *Free Photo*, *Free Photo | Portrait of happy, smiling asian hipster man, young guy in red hoodie smiling cheerful, looking camera enthusiastic, express positive mood, being delighted or satisfied, white wall.* Available at: https://www.freepik.com/free-photo/portrait-happy-smiling-asian-hipster-man-young-guy-red-hoodie-smiling-cheerful-looking-camera-enthusiastic-express-positive-mood-being-delighted-satisfied-white-wall_18405608.htm (Accessed: March 13, 2022).

Dam Rikke Friis and Siang Teo Yu (2021) *Personas – A Simple Introduction | Interaction Design Foundation (IxDF)*, *Personas – A Simple Introduction*. Available at: https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them (Accessed: March 13, 2022).

David Ortinau *et al.* (2021) *Mobile software development lifecycle - Xamarin | Microsoft Docs*, *Mobile software development lifecycle*. Available at: https://docs.microsoft.com/en-us/xamarin/cross-platform/get-started/introduction-to-mobile-sdlc (Accessed: March 13, 2022).

Dewang Nautiyal (2021) *Components of an Android Application*, *Components of an Android Application*. Available at: https://www.geeksforgeeks.org/components-android-application/ (Accessed: March 13, 2022).

Dušan Janovský (2001) *Tóny na kytaře a hledání akordů*, *Kytarové akordy a stupnice*. Available at: https://dusan.pc-slany.cz/hudba/kytara_tony.htm (Accessed: March 13, 2022).

Firebase Documentation (2022a) *Cloud Firestore* , *Cloud Firestore*. Available at: https://firebase.google.com/docs/firestore (Accessed: March 13, 2022).

Firebase Documentation (2022b) *Firebase Crashlytics*, *Firebase Crashlytics*. Available at: https://firebase.google.com/docs/crashlytics (Accessed: March 13, 2022).

Firebase Documentation (2022c) *Firebase Realtime Database* , *Firebase Realtime Database* . Available at: https://firebase.google.com/docs/database (Accessed: March 13, 2022).

Firebase Documentation (2022d) *Firebase Remote Config*, *Firebase Remote Config*. Available at: https://firebase.google.com/docs/remote-config (Accessed: March 13, 2022).

Firebase Documentation (2022e) *Google Analytics*, *Google Analytics*. Available at: https://firebase.google.com/docs/analytics (Accessed: March 13, 2022).

Framer B.V. (2022) *Framer*, *Framer*. Available at: https://framer.com/projects/ (Accessed: March 15, 2022).

Gavrilova Julia (2020) *What is the application development life cycle?*, *Application Development Life Cycle: A to Z*. Available at: https://magora-systems.com/application-development-life-cycle/ (Accessed: March 13, 2022).

Google Console Help (2022) *Intellectual Property*, *Intellectual Property*. Available at: https://support.google.com/googleplay/android-developer/answer/9888072?hl=en (Accessed: March 13, 2022).

Google Firebase (2017) *GuitarApp – Cloud Firestore – Firebase console*, *Cloud Firestore*. Available at: https://console.firebase.google.com (Accessed: March 15, 2022).

GORAN JEVTIC (2019) *What is SDLC? Phases of Software Development & Models*, *What is SDLC? Phases of Software Development, Models, & Best Practices*. Available at: https://phoenixnap.com/blog/software-development-life-cycle (Accessed: March 13, 2022).

Hardy Scott (2019) *Asynchronous vs Synchronous Programming - DEV Community*, *Asynchronous vs Synchronous Programming*. Available at: https://dev.to/hardy613/asynchronous-vs-synchronous-programming-23ed (Accessed: March 13, 2022).

Heller Martin (2020) *What is Kotlin? The Java alternative explained | InfoWorld*, *What is Kotlin? The Java alternative explained*. Available at: https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html (Accessed: March 13, 2022).

Hidden Brains Blog (2022) *Top Android Application Development Trends for 2022*, *Top Android Application Development Trends for 2022*. Available at: https://www.hiddenbrains.com/blog/top-android-application-development-trends.html (Accessed: March 13, 2022).

http://www.kytara.net/ (2010) *Akustická kytara - country, bluegrass - Kytara a její části*. Available at: http://www.kytara.net/anatomie/kytara-a-jeji-casti (Accessed: March 13, 2022).

Hub Guitar (2018) *Introduction to Music Theory | Hub Guitar*, *Introduction to Music Theory*. Available at: https://hubguitar.com/music-theory/introduction-to-music-theory (Accessed: March 13, 2022).

Invonto (2021) *Mobile App Development Process: A Step-by-Step Guide | Invonto*, *Mobile App Development Process: Step-by-Step Guide*. Available at: https://www.invonto.com/insights/mobile-app-development-process/ (Accessed: March 13, 2022).

JetBrains (2021) *Kotlin Language Documentation 1.6.10*. Available at: https://kotlinlang.org/docs/home.html (Accessed: March 13, 2022).

JoyTunes (2022) *Simply Guitar by JoyTunes*. Available at: https://apps.apple.com/us/app/simply-guitar-by-joytunes/id1476695335 (Accessed: March 13, 2022).

Kaseb Kayvan (2020) *Android Jetpack Architecture Components*, *Android Jetpack Architecture Components*. Available at: https://medium.com/kayvan-kaseb/android-jetpack-architecture-components-119c9c973d7e (Accessed: March 13, 2022).

Kaur, A. and Kaur, K. (2015) “Suitability of Existing Software Development Life Cycle (SDLC) in Context of Mobile Application Development Life Cycle (MADLC),” *International Journal of Computer Applications*, 116(19), pp. 1–6. doi:10.5120/20441-2785.

Kaur, A. and Research Scholar, P.D. (2015) *Suitability of existing Software development Life Cycle (SDLC) in context of Mobile Application Development Life Cycle (MADLC)*, *International Journal of Computer Applications*.

Kaur, H. and Sharma, Dr.A. (2014) “Non-Functional Requirements Research: Survey,” *International Journal of Science and Engineering Applications*, 3(6), pp. 172–182. doi:10.7753/ijsea0306.1003.

Kavi, K. (1998) *Multithreading Implementations*. Available at: https://www.researchgate.net/publication/2734966.

Kay Ryan Michael (2020) *How to Use Model-View-ViewModel on Android Like a Pro*, *How to Use Model-View-ViewModel on Android Like a Pro*. Available at: https://www.freecodecamp.org/news/model-view-viewmodel-android-tutorial/ (Accessed: March 13, 2022).

Kolb Apps (2021) *Real Guitar: být kytaristou – Aplikace na Google Play*, *Real Guitar: být kytaristou*. Available at: https://play.google.com/store/apps/details?id=br.com.rodrigokolb.realguitar (Accessed: March 13, 2022).

Kousen, K. (2020) *Kotlin cookbook : a problem-focused approach*. 1st edn. Edited by McQuade Zan and Ortman Tyler. Sebastopol: O’Reilly Media. doi:9781492046677.

Kuntal Sunil (2017) *SQLite Made Easy : Room Persistence Library*, *SQLite Made Easy : Room Persistence Library*. Available at: https://medium.com/mindorks/sqlite-made-easy-room-persistence-library-ecd1a5bb0a2c (Accessed: March 13, 2022).

Lardinois Frederic (2019) *Kotlin is now Google’s preferred language for Android app development | TechCrunch*. Available at: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/?guce_referrer=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAANm5Ys0uAqzjngUo2T1mMqmVu1FZMwhPjI2jEVfrBmn9rRBtGP_YnUZM87tNuQFWWvqRzae1WrWRCypoh7LkThM6r8C-

6ZB3EuSKhofz2r2qPrqADAl2i_BSH5oyICq-_0-fIp0fwFfMJPPy4kMauIGpBz8p4OSoW6HsIIMVU_4T&guccounter=2 (Accessed: March 13, 2022).

Lew Dan (2021) *RxJava vs. Coroutines*, *RxJava vs. Coroutines*. Available at: https://blog.danlew.net/2021/01/28/rxjava-vs-coroutines/ (Accessed: March 13, 2022).

Lyla Fujiwara and Android Developers (2021) *Navigation*, *Navigation*. Available at: https://developer.android.com/guide/navigation (Accessed: March 13, 2022).

Macadamian Technologies (2012) *7 Key Android Concepts | Macadamian*, *7 Key Android Concepts*. Available at: https://www.macadamian.com/learn/7-key-android-concepts/ (Accessed: March 13, 2022).

Mansi Breja (2020) *Software Engineering | Requirements Engineering Process - GeeksforGeeks*, *Software Engineering | Requirements Engineering Process*. Available at: https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/ (Accessed: March 13, 2022).

McKnight, W. (2014) "Agile Practices for Information Management," *Information Management*, pp. 168–178. doi:10.1016/B978-0-12-408056-0.00016-3.

Microsoft Docs (2012) *The MVVM Pattern*, *The MVVM Pattern*. Available at: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN (Accessed: March 13, 2022).

Mishra Rishu (2021a) *Behaviour Components of Android Jetpack*, *Behaviour Components of Android Jetpack*. Available at: https://www.geeksforgeeks.org/behaviour-components-of-android-jetpack/ (Accessed: March 13, 2022).

Mishra Rishu (2021b) *Foundation Components of Android Jetpack - GeeksforGeeks*, *Foundation Components of Android Jetpack*. Available at: https://www.geeksforgeeks.org/foundation-components-of-android-jetpack/ (Accessed: March 13, 2022).

Moore Kevin D. (2018) *Introduction to Android Jetpack*. Available at: https://www.raywenderlich.com/5376-introduction-to-android-jetpack (Accessed: March 13, 2022).

Muntenescu Florina (2020) *What's new in Jetpack*, *What's new in Jetpack*. Available at: https://medium.com/androiddevelopers/whats-new-in-jetpack-1891d205e136 (Accessed: March 13, 2022).

Muntenescu Florina and Android Developers (2022) *Android KTX*. Available at: https://developer.android.com/kotlin/ktx (Accessed: March 13, 2022).

N Inukollu, V. *et al.* (2014) "Factors Influncing Quality of Mobile Apps: Role of Mobile App Development Life Cycle," *International Journal of Software Engineering & Applications*, 5(5), pp. 15–34. doi:10.5121/ijsea.2014.5502.

Ochieng Clement (2021) *Kotlin Android DownloadManager Tutorial and Examples - Android Examples*, *Kotlin Android DownloadManager Tutorial and Examples*. Available at: https://camposha.info/android-examples/android-downloadmanager/#gsc.tab=0 (Accessed: March 13, 2022).

Oliver Trunkett (2020) *SDLC Methodologies: From Waterfall to Agile | Virtasant*, *SDLC Methodologies: From Waterfall to Agile*. Available at: https://www.virtasant.com/blog/sdlc-methodologies (Accessed: March 13, 2022).

Papapishu (2007) *guitar 1 - Openclipart*. Available at: https://openclipart.org/detail/10117/guitar-1 (Accessed: March 13, 2022).

Patel Keval (2016) *What is Reactive Programming? | by Keval Patel | Medium*, *What is Reactive Programming?* Available at: https://medium.com/@kevalpatel2106/what-is-reactive-programming-da37c1611382 (Accessed: March 13, 2022).

Patel Saurabh (2018) *Android Jetpack: Android Slices (Part-1) Introduction* , *Android Jetpack: Android Slices (Part-1) Introduction*. Available at: https://proandroiddev.com/android-jetpack-android-slices-introduction-cf0ce0f3e885 (Accessed: March 13, 2022).

Pedemkar Priya (2020) *MVP vs MVVM*, *MVP vs MVVM*. Available at: https://www.educba.com/mvp-vs-mvvm/ (Accessed: March 13, 2022).

Pickmans Alvaro Ortega (2019) *WPF and MVVM*, *Dynamo Development*. Available at: https://alvpickmans.github.io/DynamoDevelopment-London-Hackathon-2019/ (Accessed: March 13, 2022).

Polyakov Andrey *et al.* (2021) *Coroutines guide | Kotlin*, *Coroutines guide*. Available at: https://kotlinlang.org/docs/coroutines-guide.html (Accessed: March 13, 2022).

@pressphoto (2022) *Free Photo | Stylish elderly woman*. Available at: https://www.freepik.com/free-photo/stylish-elderly-woman_5400760.htm#query=Elder%20lady&position=34&from_view=search (Accessed: March 13, 2022).

PrivacyPolicies.com Legal Writing Team (2021) *GDPR Compliance for Apps*, *GDPR Compliance for Apps*. Available at: https://www.privacypolicies.com/blog/gdpr-compliance-apps/ (Accessed: March 13, 2022).

Raphael JR (2022) *Android versions: A living history from 1.0 to 13 | Computerworld*. Available at: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html (Accessed: March 13, 2022).

ReactiveX contributors (2011) *ReactiveX - Intro*, *ReactiveX*. Available at: https://reactivex.io/intro.html (Accessed: March 13, 2022).

Richardson Leland (2020) *Understanding Jetpack Compose — part 1 of 2* , *Understanding Jetpack Compose — part 1 of 2*. Available at: https://medium.com/androiddevelopers/understanding-jetpack-compose-part-1-of-2-ca316fe39050 (Accessed: March 13, 2022).

Rosencrance Linda (2019) *What is Google Firebase?* , *Google Firebase*. Available at: https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase (Accessed: March 13, 2022).

Ryax technologies (2021) *Reactive programming: what is it? - Ryax Technologies*, *Reactive programming: what is it?* Available at: https://ryax.tech/reactive-programming-what-is-it/ (Accessed: March 13, 2022).

Semy0nov Pavel, Petrakovich Victoria and Polyakov Andrey (2021) *Coroutines | Kotlin*, *Coroutines*. Available at: https://kotlinlang.org/docs/coroutines-overview.html (Accessed: March 13, 2022).

Sharma Prateek (2020) *LiveData Tutorial for Android: Deep Dive | raywenderlich.com*. Available at: https://www.raywenderlich.com/10391019-livedata-tutorial-for-android-deep-dive (Accessed: March 13, 2022).

Sherman, R. (2015) "Project Management," *Business Intelligence Guidebook*, pp. 449–492. doi:10.1016/B978-0-12-411461-6.00018-6.

similarweb.com (2022) *Top Music & Audio Apps Ranking - Most Popular Apps in United States | Similarweb*, *Top Apps Ranking - Music & Audio*. Available at: https://www.similarweb.com/apps/top/google/store-rank/us/music-audio/top-free/AndroidPhone/ (Accessed: March 13, 2022).

Smyth, Neil. (2022) *Android Studio Bumble Bee Essentials - Java Edition Developing Android Apps Using Android Studio 2022. 1. 1 and Java.* Payload Media.

Späth, P. (2018) *Pro android with Kotlin: Developing modern mobile apps*, *Pro Android with Kotlin: Developing Modern Mobile Apps*. Apress Media LLC. doi:10.1007/978-1-4842-3820-2.

SQLite Consortium (2022) *SQLite Home Page*, *What Is SQLite?* Available at: https://sqlite.org/index.html (Accessed: March 13, 2022).

Stoyanov Dennis (2017) *Using Schedulers*, *Using Schedulers*. Available at: http://xgrommx.github.io/rx-book/content/getting_started_with_rxjs/scheduling_and_concurrency.html (Accessed: March 13, 2022).

Strand Peter J., Kouchoukas Robert and Rattner William (2005) *Legal Issues Involved in the Music Industry MUSIC COPYRIGHTS*. Chicago. Available at: www.copyright.gov.

Strawn Jay (2018) *Design Patterns by Tutorials: MVVM , Design Patterns by Tutorials: MVVM*. Available at: https://www.raywenderlich.com/34-design-patterns-by-tutorials-mvvm (Accessed: March 13, 2022).

Team MindOrks (2019a) *Android KTX - Android development with Kotlin*, *Android KTX - Android development with Kotlin*. Available at: https://blog.mindorks.com/android-ktx-android-development-with-kotlin (Accessed: March 13, 2022).

Team MindOrks (2019b) *Implementing Paging Library in Android*, *Implementing Paging Library in Android*. Available at: https://blog.mindorks.com/implementing-paging-library-in-android (Accessed: March 13, 2022).

Thomas, C.G. and Jayanthila Devi, A. (2021) "A Study and Overview of the Mobile App Development Industry," *Google Scholar Citation: IJAEML International Journal of Applied Engineering and Management Letters (IJAEML) A Refereed International Journal of Srinivas University*, 5(1), pp. 2581–7000. doi:10.5281/zenodo.4966320.

Thompson Barbara (2022) *Android Architecture: Application Layers, Framework, Component*. Available at: https://www.guru99.com/android-architecture.html (Accessed: March 13, 2022).

Tim Parsons (2019) *When to Use Waterfall vs. Agile - Macadamian*, *When to Use Waterfall vs. Agile*. Available at: https://www.macadamian.com/learn/when-to-use-waterfall-vs-agile/ (Accessed: March 13, 2022).

Upadhayay Dev (2021) *SDLC Deployment Phase - A Step by Step Guide [2021] - OpenXcell*, *SDLC Deployment Phase – A Step by Step Guide*. Available at: https://www.openxcell.com/blog/sdlc-deployment-phase/ (Accessed: March 13, 2022).

Vogel Eric (2011) *Asynchronous Programming in .NET: I'll Call You Back -- Visual Studio Magazine*, *Asynchronous Programming in .NET: I'll Call You Back*. Available at: https://visualstudiomagazine.com/articles/2011/03/24/wccsp_asynchronous-programming.aspx (Accessed: March 13, 2022).

wallpapersafari.com (2020) *Free download Guitar Dark Music Instrument 4K Wallpaper Best Wallpapers [1080x1920] for your Desktop, Mobile & Tablet | Explore 55+ 4k Music Mobile Wallpapers | 4k Music Mobile Wallpapers, 4K Mobile Wallpaper, PUBG Mobile 4k Wallpapers*. Available at: https://wallpapersafari.com/w/VZmM4p (Accessed: March 13, 2022).

@wayhomestudio (2022) *Free Photo*, *Free Photo | Tender feminine woman with blue eyes, smiles pleasantly, has toothy smile, wears white comfortable sweater, looks directly at camera, isolated on pink background*. Available at: https://www.freepik.com/free-photo/tender-feminine-woman-with-blue-eyes-smiles-pleasantly-has-toothy-smile-wears-white-comfortable-sweater-looks-directly-camera-isolated-pink-

background_12697877.htm#query=Young%20girl&position=46&from_view=search (Accessed: March 13, 2022).

Yousician (2022) *Yousician: Your Music Teacher* , *Yousician: Your Music Teacher*. Available at: https://play.google.com/store/apps/details?id=com.yousician.yousician (Accessed: March 13, 2022).

8 Seznam obrázků, tabulek, grafů a zkratek

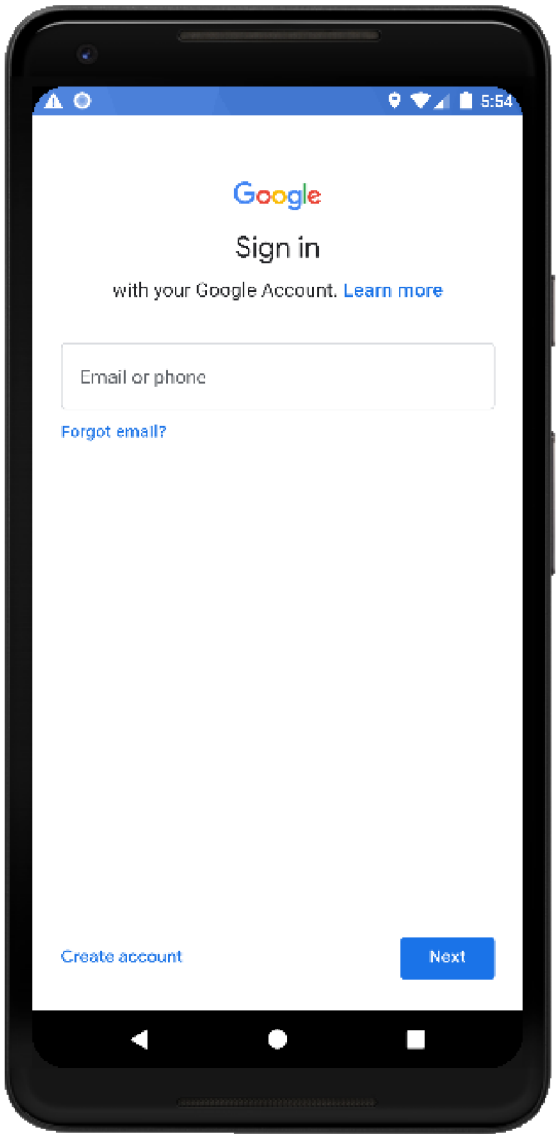
8.1 Seznam obrázků

8.2 Seznam tabulek

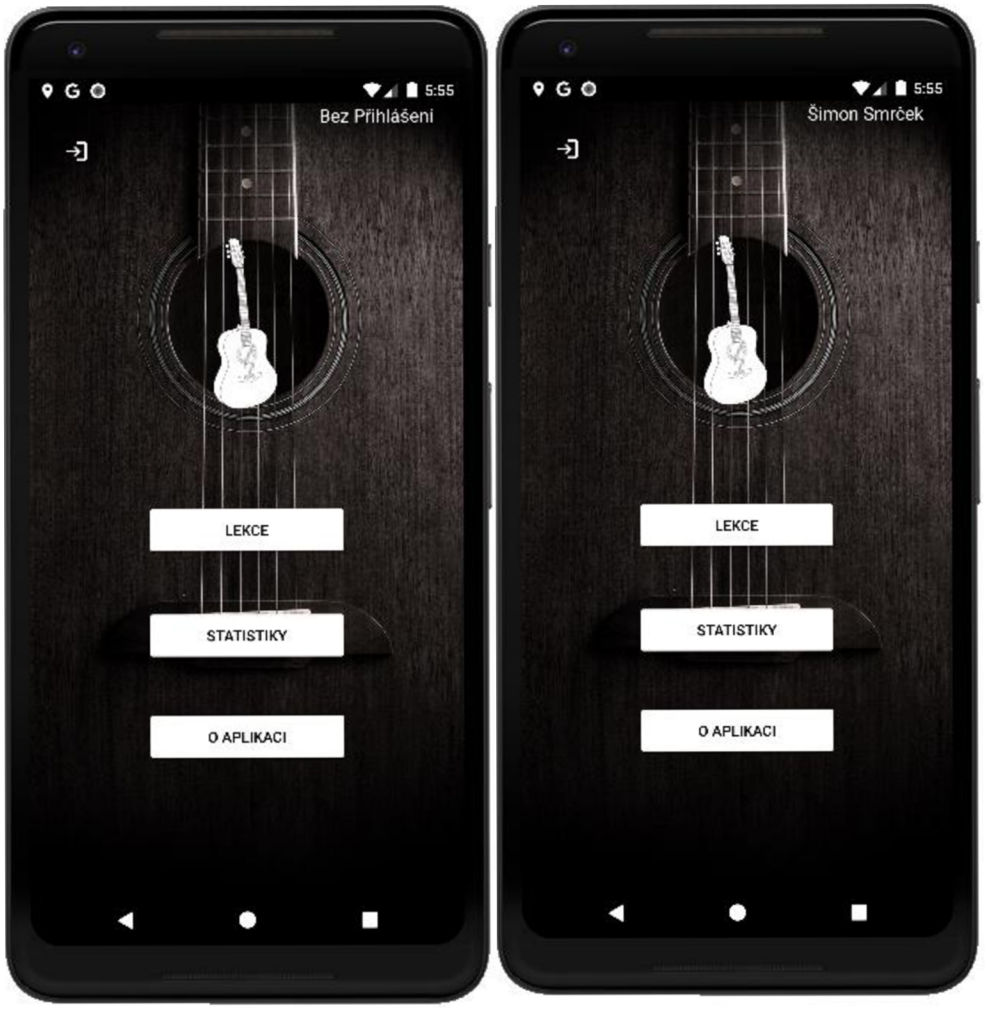
8.3 Seznam zdrojových kódů

Přílohy

Příloha 1 – Úvodní stránka



Příloha 2 – Přihlášení přes Google

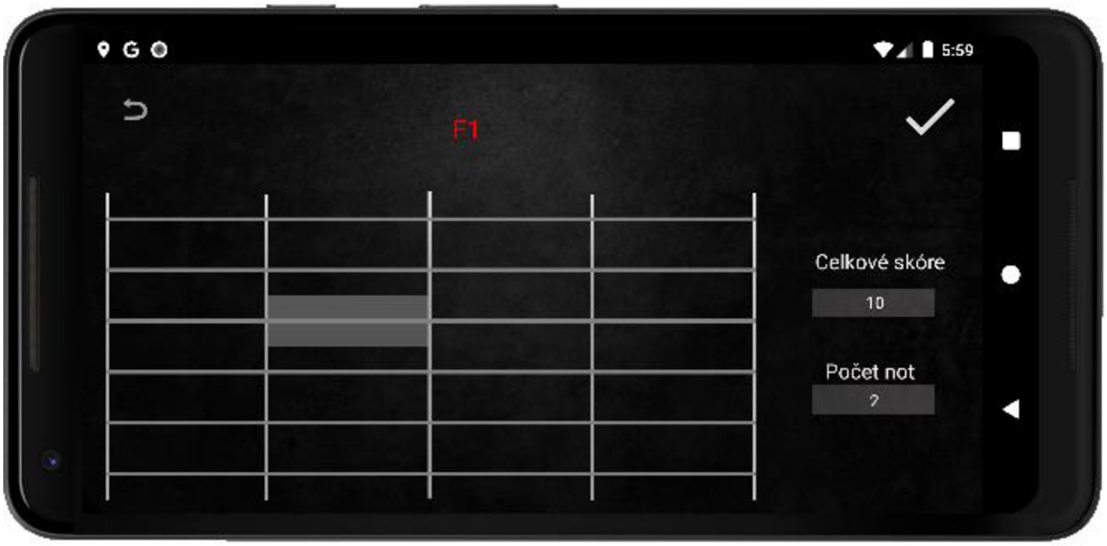


Příloha 3 – Domovská stránka bez přihlášení a s přihlášením

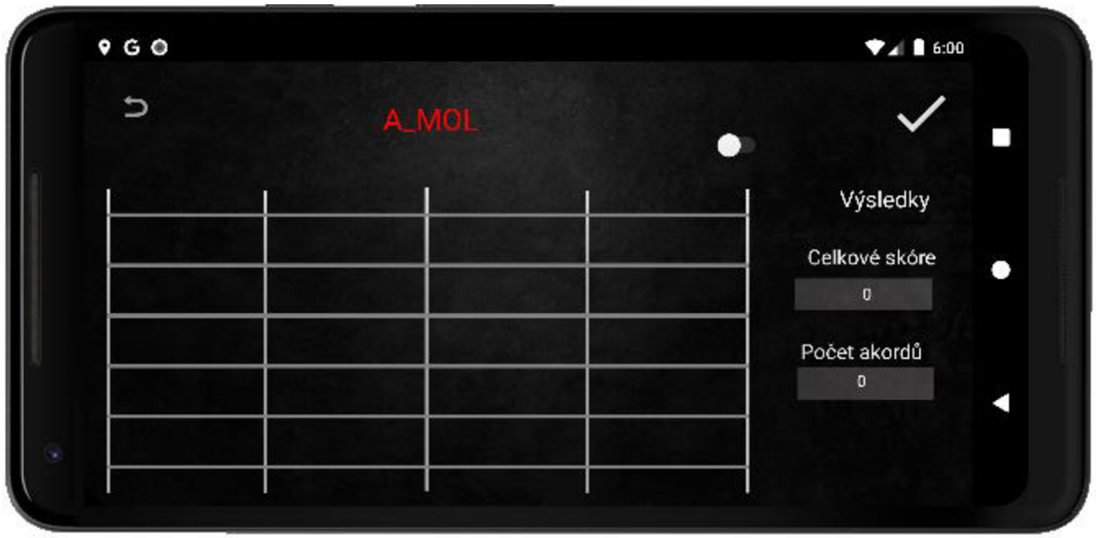
Příloha 4 – Stránka O Aplikaci

Příloha 5 – Stránka Lekce

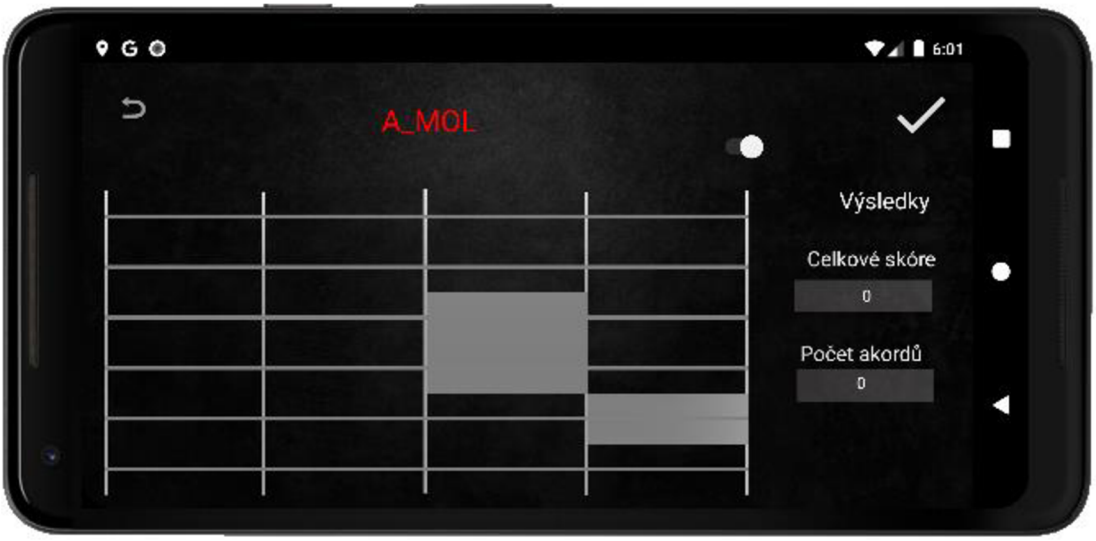
Příloha 6 – Stránka pravidel Noty, Akordy a Rytmus



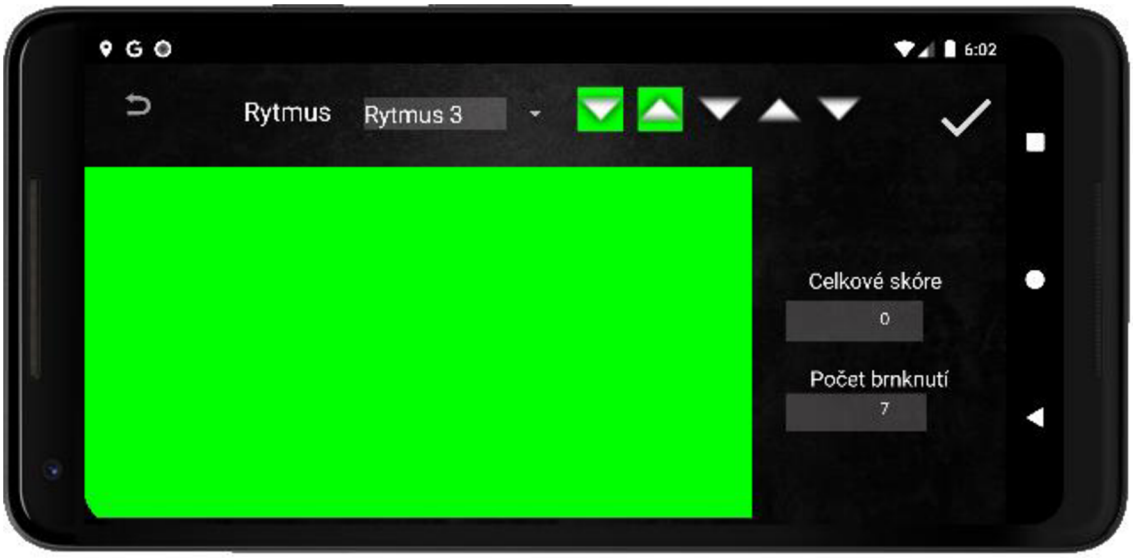
Příloha 7 – Cvičení Noty



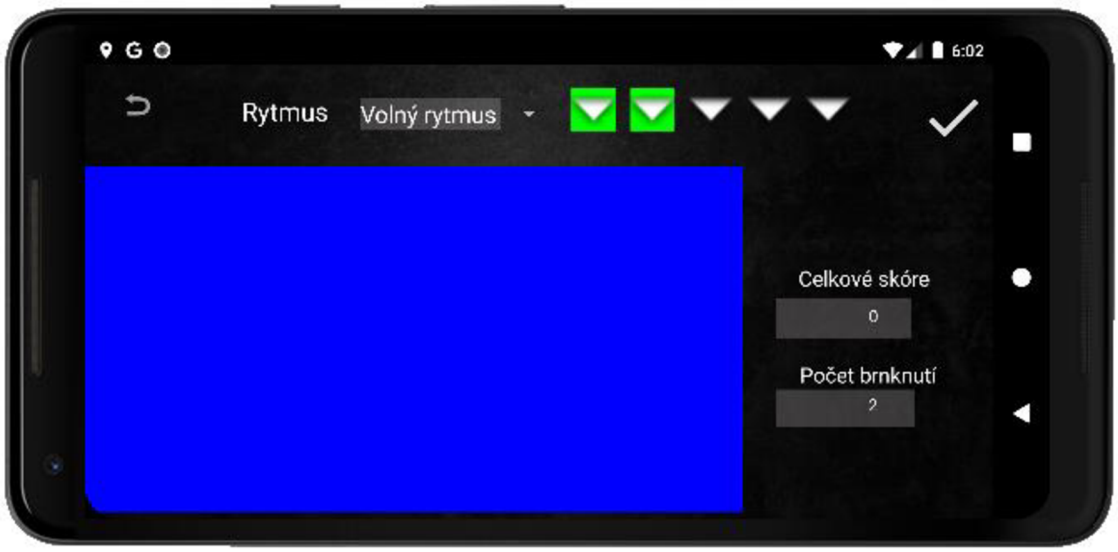
Příloha 8 – Cvičení Akordy bez asistenta



Příloha 9 – Cvičení Akordy s asistentem



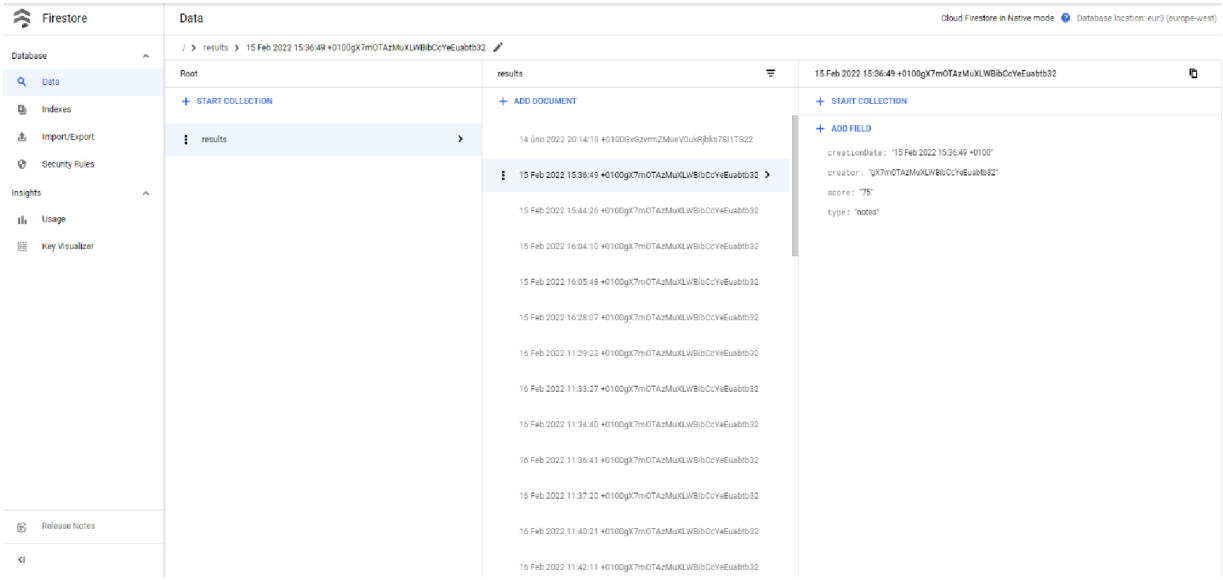
Příloha 10 – Cvičení Rytmus, pohyb nahoru



Příloha 11 – Cvičení Rytmus, pohyb dolu

Příloha 12 – Výsledky cvičení

Příloha 13 – Stránka Statistiky



Příloha 14 – Firecloud Firebase uložení výsledků (Google Firebase, 2017)