FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	iOS aplikace k výuce hry na kytaru
Student:	Petr Šmejkal
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem této bakalářské práce je vytvoření mobilní aplikace pro iOS určené k výuce hry na kytaru. Výsledná aplikace bude obsahovat jednotlivé lekce, které budou realizovány pomocí videa, obrázků a textu. Obsah bude možné přidávat a aktualizovat v rámci provozu aplikace. Součástí aplikace bude i jednoduchá ladička kytary.

Postupujte v souladu s metodami softwarového inženýrství:

- Analyzujte konkurenční iOS aplikace sloužící k výuce hry na kytaru. Současně analyzujte také tradiční přístup k výuce hry na kytaru a sestavte požadavky na aplikaci.
- Seznamte se s vývojovým prostředím určeným pro vývoj iOS aplikací.
- Vyberte vhodné technologie a navrhněte mobilní aplikaci dle požadavků.
- Implementujte a testujte samotnou aplikaci.
- Zhodnoťte výsledek a navrhněte další možný rozvoj.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 12. prosince 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

iOS aplikace k výuce hry na kytaru

Petr Šmejkal

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

13. května 2021

Poděkování

Chtěl bych poděkovat svému vedoucímu, panu Ing. Markovi Suchánkovi, za vřelý přístup a odborné vedení mé bakalářské práce a také za rady poskytnuté při řešení nahodilých překážek v průběhu vytváření této práce. Dále bych chtěl poděkovat své rodině za poskytnutí prostředků a podporu při celé době mého bakalářského studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Šmejkal, Petr. *iOS aplikace k výuce hry na kytaru*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato bakalářská práce ukazuje celý proces vývoje aplikace určené k výuce hry na kytaru pro platformu iOS podle standardních postupů softwarového inženýrství. Na úvod je provedena analýza způsobu výuky na kytaru a konkurenčních aplikací. Na základě této analýzy jsou sestaveny požadavky. Z předchozích poznatků jsou vybrány vhodné technologie a vytvořen návrh aplikace s následnou implementací a testováním. Práce je zakončena zhodnocením výsledků a nastíněním dalšího možného rozvoje. Výsledkem je funkční aplikace připravená k budoucímu rozšíření.

Klíčová slova mobilní aplikace, iOS vývoj, analýza požadavků, Xcode, Swift, UIKit, MVC, Firebase, unit testování, uživatelské testování, kytara

Abstract

This Bachelor's thesis contains the whole process of developing an iOS guitar tutoring app, according to the standard software engineering procedures. At first, an analysis of the method of teaching the guitar is performed as well as an analysis of similar applications. Based on the analysis, requirements for the software are specified and suitable technologies are chosen. Furthermore, the implementation and the testing are described. The thesis concludes with an evaluation of the results and an outline of further possible development. The result is a functional application ready for further extensions.

Keywords mobile applications, iOS developement, requirements engineering, Xcode, Swift, UIKit, MVC, Firebase, unit testing, usability testing, guitar

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Stále se velmi rychle rozvíjející trh s mobilními aplikacemi se již před delší dobou stal součástí životů většiny lidí. Ač výběr mobilních aplikací je nevídaně široký a účel těchto produktů se velmi liší, všechny tyto aplikace mají jedno společné – snaží se poskytnout dostupný produkt, který lze využívat téměř kdekoliv. Některé aplikace výrazně ulehčují každodenní život, jiné například poskytují zábavu ve volném čase a další mohou pomoci při učení nových dovedností.

Právě ona schopnost mobilní aplikace učit novým dovednostem ve spojení se zajímavými technologiemi se stala jednou z hlavních motivací k rozhodnutí pokusit se vytvořit aplikaci k výuce hry na kytaru. Navíc autor sám má zkušenosti s hraním na tento nástroj a v malé míře i s výukou, především pak začátečníků.

Tento produkt má za úkol ukázat, že hraní na hudební nástroj může zkusit opravdu každý – z toho důvodu by bylo dobré, aby tato aplikace nebyla zpoplatněna, což lze považovat za konkurenční výhodu, jelikož drtivá většina podobných aplikací je alespoň částečně zpoplatněná. Dalším kreativním nápadem je integrování jednoduché ladičky kytary.

Práce obsahuje analýzu, návrh, implementaci a testování iOS mobilní aplikace určené k výuce hry na kytaru, která bude vznikat s ohledem na hudební zkušenosti autora a odzkoušené praktiky softwarového inženýrství.

Kapitola 1

Cíl práce

Cílem je vytvoření mobilní aplikace k výuce hry na kytaru pro platformu iOS a to podle tradičního procesu vývoje v softwarovém inženýrství. To znamená analyzovat doménu, sestavit požadavky, vytvořit návrh a na závěr implementovat a testovat aplikaci.

Na samotném začátku je potřeba analyzovat způsob, jak vyučovat hraní na kytaru a také podobné mobilní aplikace, které se v současné době používají. Na základě výsledků analýzy lze sestavit požadavky na aplikaci. Dále je potřeba vytvořit návrh – vybrat vhodnou architekturu, vytvořit databázové schéma a navrhnout uživatelské rozhraní. Závěrečným cílem je samotná implementace za použití vhodných technologií a testování aplikace. Práce bude zakončena zhodnocením celého výsledku a návrhem dalšího možného rozvoje. Zmíněné dílčí cíle odpovídají konkrétním bodům zadání této bakalářské práce.

Kapitola 2

Analýza

Tato kapitola se zabývá analýzou výuky hry na kytaru. Dále jsou analyzovány tři podobné aplikace, zhodnoceny jejich silné a slabé stránky a nakonec zhotovený závěr z těchto poznatků. Poté jsou na základě výsledků analýzy sestaveny funkční a nefunkční požadavky na tuto aplikaci a popsány případy užití. Analýza se později využije při návrhu, který bude vycházet ze sesbíraných požadavků.

2.1 Výuka hry na kytaru

Způsob, výuky hraní na kytaru, může být velmi různý. Každému může vyhovovat odlišný styl učení – záleží např. na aktuálních dovednostech žáka, časovém vytížení, žánru hudby a mnoha dalších aspektech, nicméně lze shrnout některé poznatky, které obecně fungují a pomáhají ulehčit cestu při učení se hraní.

2.1.1 Obvyklý postup výuky

Při výuce hraní na kytaru se jeví jako zásadní dodržet návaznost jednotlivých fází – jinak řečeno, tyto fáze nelze přeskakovat. Hra na kytaru začíná základními cvičeními, na kterých se pak stavějí složitější. Fáze výuky mohou vypadat následovně [12]:

Pochopení stupnic a intervalů

Úplně prvním krokem je pochopení základní hudební teorie – jak fungují stupnice a intervaly, kde se na kytarovém hmatníku nachází který tón (alespoň z těch základních), jak se tyto tóny na kytaře opakují, apod. Tento krok pomůže i při učení akordů, které budou dávat z hlediska hudební teorie (což pomůže i při samotném hraní) větší smysl. [12]

Akordy a jejich variace

Naučení se akordů bývá také jedním ze základních kroků výuky hry na kytaru. Pokud má žák ponětí o tom, jak fungují stupnice a intervaly, bude pro něj jednodušší pochopit i akordy – ty se totiž řídí určitými pravidly. Tato pravidla říkají, jak stavět který akord a jeho variace. Akordy je také vhodné vyučovat různých formách – stejný akord lze obvykle zahrát několika způsoby. [12]

Prstokladová cvičení

Další fáze výuky se zabývá prstokladovými cvičeními – takzvaným vybrnkáváním. To je obecně pro začátečníky složitější, než hraní jednoduchých akordů. Právě již zmíněnými cvičeními lze trénovat tuto schopnost, která bude v budoucnu důležitá pro sólové hraní. [12]

Zvládnutí známých riffů a skladeb

Tato část výuky pomáhá žákovi lépe pochopit logické harmonické postupy – zkrátka to, jaké sekvence tónů a akordů zní dobře a které ne. V této fázi je důležité, aby žák už měl dostatečnou úroveň dovednosti hry. Vhodné je rovněž začít lehkými skladbami a riffy a postupně zvyšovat náročnost. [12]

Improvizace

Improvizace je jedna nejnáročnějších dovedností, kterou by však dobrý kytarista/tka měl/a ovládat. Jistá improvizace pravděpodobně u mnoha žáků přijde sama již v dřívějších fázích, nicméně lze ji při výuce podpořit různými mechanizmy z hudební teorie (např. pentatonické stupnice) a tipy zkušených hráčů. [12]

Důležité je nezapomínat, že všechny zmíněné fáze a postupy nebudou fungovat při dlouhodobém a pravidelném cvičení. Jedná se především o odzkoušené praktiky, které obecně fungují, ale rozhodně z nikoho neudělají skvělého kytaristu za měsíc.

2.1.2 Ostatní aspekty

Při učení se novým dovednostem se chceme, aby se žák cítil snad možná co nejvíce komfortně. To platí i při učení se hraní na kytaru, proto je prostřední, kde se dotyčný učí, velmi důležité. Existuje několik vědecky podložených poznatků o tom, jak by mělo ideální prostředí pro učení se hraní vypadat – např. je dobré trénovat ve světlé místnosti, kde bude klid a žák nebude rušen okolním hlukem. Nebude v přílišné zimě, aby mu netuhly prsty, ale zároveň by se neměl potit horkem, protože tak se určitě nebude cítit v pohodě. [13]

Kvalitní lekce by měly žáka bavit. Aspektu zábavy lze docílit mnoha různými způsoby. Jedním může být například možnost porovnání se s ostatními

žáky, případně soutěže, ve kterých žáci hrají jeden proti druhému. Důležité je rovněž umožnit žákovi, aby si po zvládnutí určité části mohl ověřit své schopnosti a tím znát, jaké úrovně jeho um aktuálně dosahuje. [13]

Další důležitou součástí úspěšné výuky, nejspíše tou nejdůležitější, je udržení zájmu žáka v delším časovém intervalu [14]. Toho lze docílit tím, že výuka bude probíhat pravidelně, v nějakou určitou dobu, kterou považuje žák za ideální. Konkrétní čas během dne se bude lišit podle denního rytmu daného člověka. Dále lze vyzkoušet různé meditační a uvolňovací techniky, které slouží jako podpůrné procesy – uvolněný žák bude pravděpodobně trpělivější a odhodlanější dosáhnout cíle. Kvalitní spánek též přispěje k pozitivnímu přístupu k učení. [13]

2.1.3 Mobilní aplikace jako nástroj výuky

Z předchozích bodů je vidět, že mobilní aplikace může být ideální řešení [13]. Aplikaci lze využívat téměř kdekoliv a kdykoliv, tudíž při používání aplikace doma, ve vhodně vybrané části dne, lze dosáhnout požadovaného komfortu. Součástí aplikace může být mnoho zábavných a motivačních prvků, jako například seznam splněných lekcí, žebříček nejpilnějších uživatelů, různé výzvy nebo prvky her jako součást lekcí. Všechny tyto vlastnosti pomohou uživatele motivovat k dlouhodobému používání aplikace a dosažení cíle. Interaktivita společně s již zmíněnou dostupností dává takové aplikaci výhodu oproti klasické učebnici a v některých ohledech i tradiční hodině s lektorem.

Navíc při vhodném zvolení obsahu výukových materiálů aplikace a správném rozvržení lekcí bude umožněn obvyklý způsob výuky hry na kytaru, který pravděpodobně urychlí celý proces učení.

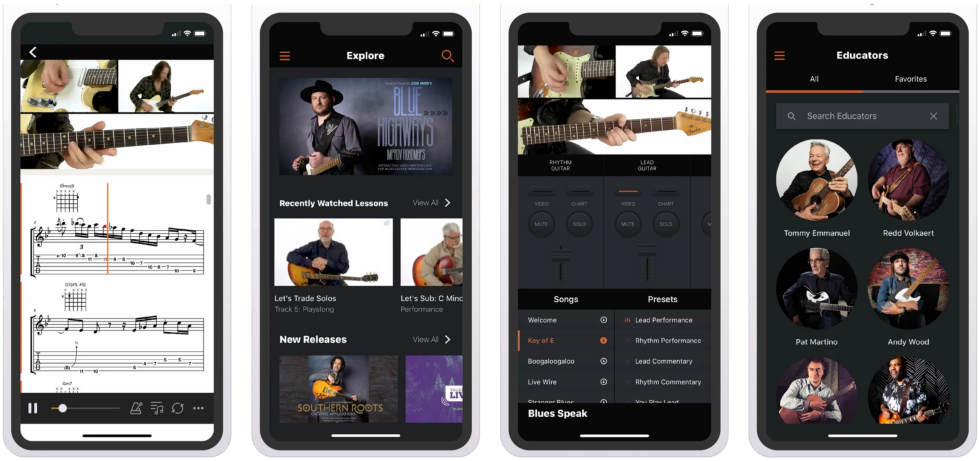
Na základě poznatků z této a dalších sekcí analýzy bude možné později v této kapitole sestavit požadavky vyvíjené aplikace.

2.2 Konkurenční aplikace

Analýza konkurenčních řešení pomůže při sestavování požadavků vyvíjené aplikace a dokáže napovědět, jak získat výhodu oproti konkurenci. Proto byly z *App Store*, online distribuční služby aplikací pro iOS a iPadOS, vybrány tři populární aplikace určené k výuce hry na kytaru. Následuje popis těchto aplikací a zhodnocení jejich silných a slabých stránek.

2.2.1 TrueFire Guitar Lessons

Tato aplikace určitě patří mezi nejpopulárnější aplikace určené k výuce hry na kytaru. Aplikace je dostupná jak pro platformu iOS a Android, tak i pro běžné desktopové operační systémy. V této práce se však bude zajímáno o verzi pro iOS. Samotná aplikace je zdarma, avšak uživatel si později může zakoupit nabízené návody. [15]



Obrázek 2.1: Ukázka obrazovek aplikace Truefire [1]

Truefire nabízí velmi širokou škálu mnoha různých lekcí a interaktivních cvičení, včetně návodů od profesionálních hráčů na kytaru, jako jsou například legendární Tommy Emmanuel a Joe Robinson, což je zcela jistě nespornou výhodou této platformy. Aplikace je určena jak pro začátečníky, tak pro velmi pokročilé hráče na kytaru.

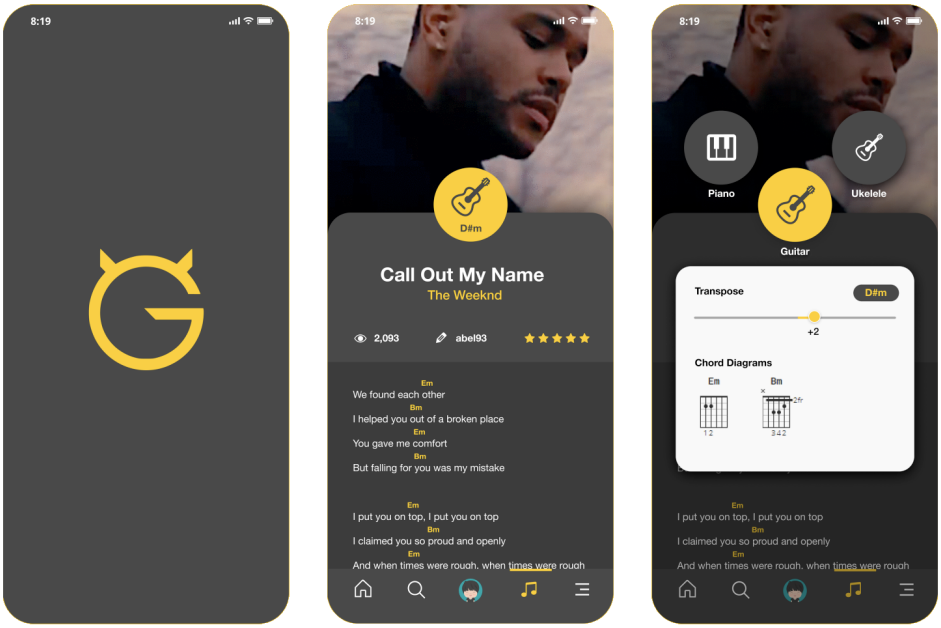
Z uživatelského hlediska patří tato aplikace mezi ty kvalitně zpracované, nabízí mnoho užitečných funkcí jako např. možnost stažení obsahu a prohlížení offline, uložení rozpracované lekce, vizualizace tabů v průběhu přehrávaní lekce a další. Z vlastní zkušenosti testování aplikace autorem této práce a také podle uživatelských recenzí trpí Truefire delší dobou načítání obsahu, občasnými záseky nebo vypínáním aplikace a ojedinělými chybami v UI (překrývání některých elementů). Tyto problémy se však poslední dobou daří postupně eliminovat pomocí aktualizací. [1]

Celkově se tedy jedná o kvalitní aplikaci, jejíž největší výhodou je obsah od známých osobností ze světa hudby. Problémy u Truefire se projevují občasnými chybami a ojedinělým neočekávaným chováním.

Tabulka 2.1: Tabulka silných a slabých stránek aplikace Truefire

Silné stránky	**Slabé stránky**
Velké množství kvalitního obsahu	Placený obsah
Lekce od známých osobností	Občasné záseky a chyby v UI
Určené jak pro začátečníky, tak pro zkušené kytaristy	Delší doba načítání

2.2.2 Ultimate Guitar: Chords & Tabs



Obrázek 2.2: Ukázka obrazovek aplikace Ultimate Guitar [2]

Ulitmate Guitar patří mezi přední aplikace určené k výuce hry na kytaru. Tato platforma se mezi hráči na kytaru nachází už velmi dlouho dobu, a i proto je tak populární. Její verze existuje pro všechny běžné platformy včetně Android a iOS. Jedná se neplacenou aplikaci s placeným obsahem v podobě předplatného.

Ultimate Guitar spojuje kytarové návody s rozsáhlou komunitou kytaristů, která je tvůrcem obsahu této aplikace. V rámci komunity je možné s uživateli různorodě interagovat (např. chatovat). Z toho důvodu, že obsah je vytvářen právě komunitou, se z Ultimate Guitar stal jeden z největší zpěvníku na světě. Na druhou stranu se ze stejného důvodu objevuje celkem značné množství nepřesného obsahu (akordů, tabů), ojediněle i úplně nepoužitelný obsah. [16]

Z hlediska uživatelského prožitku se jedná o velmi kvalitní aplikaci. Její předností je například automatické posouvání obsahu obrazovky při přehrávání, možnost transpozice do vyšších/hlubších tónin a další. Aplikace poskytuje příjemné a přehledné uživatelské rozhraní. Podle uživatelských recenzí a výsledků vlastního testování se jedná o aplikaci s minimem chyb a nedostatků. [16, 17]

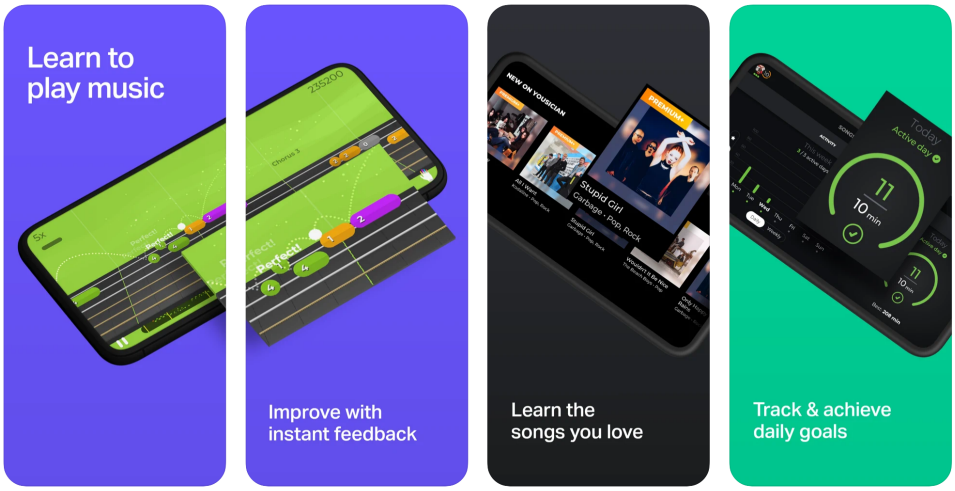
Souhrnně se tato aplikace řadí mezi nejkvalitnější a nejpropracovanější produkty za účelem učení hry na kytaru pro platformu iOS. Nevýhodou může

být občasný nesprávný obsah.

Tabulka 2.2: Tabulka silných a slabých stránek aplikace Ultimate Guitar

Silné stránky	Slabé stránky
Pestrý obsah tvořený komunitou	Placené lekce
Rozsáhlá komunita	Nabízí i chybný obsah
Mnoho funkcí, propracové UI	

2.2.3 Yousician – Your Music Teacher



Obrázek 2.3: Ukázka aplikace Yousician [3]

Alternativou k zmíněným aplikacím může být Yousician, která kromě standardních lekcí nabízí také interaktivní návody s prvky her a propracovaný systém plnění úkolů a dosahování cílů. Tato aplikace je zdarma, avšak poté si uživatel odemyká daný obsah v rámci vybraného programu předplatného.

Podle uživatelských recenzí a vlastního testování se jedná se o kvalitně propracovanou aplikaci s velmi líbivým a zajímavým UI, které v kombinaci s animacemi a chytře zpracovanými lekcemi dává uživateli najevo, že se jedná o prvotřídní produkt. Jediný problém může nastat v některých případech při rozpoznávání zvuku, který není vždy správně zaznamenán, což může ztěžovat průchod lekcemi.

Celkově se jedná o propracovanou aplikaci, která nabízí neobvyklý způsob prezentace lekcí. Chyby se vyskytují spíše výjimečně.

Tabulka 2.3: Tabulka silných a slabých stránek aplikace Yousician

Silné stránky	Slabé stránky
Zajímavé způsoby prezentování lekcí	Placené lekce
Prvotřídní UI	Občasné problémy s rozpoznáváním zvuku

2.2.4 Shrnutí konkurenčních aplikací

Po celkové analýze a zhodnocení silných a slabých stránek předchozích tří populárních aplikací si lze udělat představu o tom, které funkcionality a vizuální prvky je vhodné použít ve vyvíjené aplikaci a které bude pravděpodobně lepší vynechat. Na základě těchto výsledku bude rovněž o poznání jednodušší sestavit funkční a nefunkční požadavky, což je obsahem následující sekce. Závěrem lze tedy říci, že byl úspěšně analyzován jak způsob, kterým se zdá být vhodné prezentovat lekce kytary (jakožto obsah v mobilní aplikaci), tak největší konkurenční aplikace, které posloužily k inspiraci a také poučení se z jejich nedostatků.

2.3 Požadavky na aplikaci

Před samotným návrhem aplikace je nutné provést sběr požadavků tak, aby bylo možné přesněji definovat chování výsledného produktu. Zároveň pak lze odhadnout pracnost projektu a vymezit jeho hranice. Správný požadavek by měl splňovat následující vlastnosti:

- **Jednoznačnost** – Požadavek by měl jasně a konkrétně vystihovat svůj účel a cíl.
- **Splnitelnost** – Požadavek musí být možné realizovat.
- **Ověřitelnost** – požadavek by mělo být možné ověřit pomocí akceptačního testování. [4]

Požadavky se mohou dělit na funkční požadavky a nefunkční požadavky:

- **Funkční požadavky** – Jsou takové základní požadavky, které definují chování systému.
- **Nefunkční požadavky** – Omezují informační systém, mají dopad na zvolení architektury a dodržení standardů. [4]

Požadavky jsou dále ohodnocené podle proritizační metody MoSCoW, tedy: [18]

- **Must Have** – Tyto požadavky musí být splněny.
- **Should Have** – Takové požadavky není nutné splnit, ale měl by být součástí řešení, pokud to není vyloženě nemožné.
- **Could Have** – Požadavky *Could Have* obykle zvyšují komfort a spokojenost zákazníka. Obvykle se realizují v případě, že na ně při vývoji zbude čas.
- **Won't Have** – Požadavky určené pro budoucí rozvoj vyvíjeného software.

Následující obrázek 2.4, diagram požadavků, shromažďuje všechny funkční a nefunkční požadavky. Poté následuje textový popis jednotlivých požadavků a jejich priorit.

Obrázek 2.4: Diagram požadavků, dle [4]

2.3.1 Funkční požadavky

FP1 – Vytvoření uživatelského účtu: Uživatel je vyzván k vyplnění jména, emailu a dále si musí zvolit heslo. *(Must Have)*

FP2 – Zaslání potvrzovacího odkazu emailem: Po dokončení registrace je novému uživateli zaslán email s odkazem na potvrzení svého účtu. Do nepotvrzeného účtu se nelze přihlásit. *(Must Have)*

FP3 – Přihlášení do aplikace: Pro registrované uživatele je možné se přihlásit a vstoupit do aplikace pomocí emailu a hesla. *(Must Have)*

FP4 – Obnovení zapomenutého hesla: Při zapomenutí hesla je možné zadat email na který je zaslán email s odkazem na obnovu hesla. *(Should Have)*

FP5 – Zobrazení lekcí: Uživatel si může zobrazit lekce buď na domovské obrazovce nebo v seznamu lekcí. *(Must Have)*

FP6 – Ladička kytary: K dispozici je jednoduchá ladička kytary. *(Must Have)*

FP7 – Odhlášení: Uživatel se může odhlásit ze svého uživatelského účtu. *(Must Have)*

FP8 – Změna hesla: Uživatel si může změnit přístupové heslo ke svému účtu. *(Should Have)*

FP9 – Vkládání lekcí: Uživatel typu administrátor může vkládat nové lekce (tj. nahrávat text, obrázky a video). *(Must Have)*

FP10 – Mazání lekcí: Uživatel typu administrátor může smazat existující lekci. *(Could Have)*

FP11 – Změna profilového obrázku uživatele: Uživatel si může v nastavení měnit svůj profilový obrázek. *(Could Have)*

FP12 – Kategorie lekcí: Uživatel typu administrátor vytvářet kategorie pro lekce, které uživateli ulehčují orientaci mezi lekcemi. *(Won't Have)*

FP13 – Různé stupnice ladění: Uživatel si může vybrat tóninu, ve které chce, aby fungovala integrovaná ladička. *(Won't Have)*

2.3.2 Nefunkční požadavky

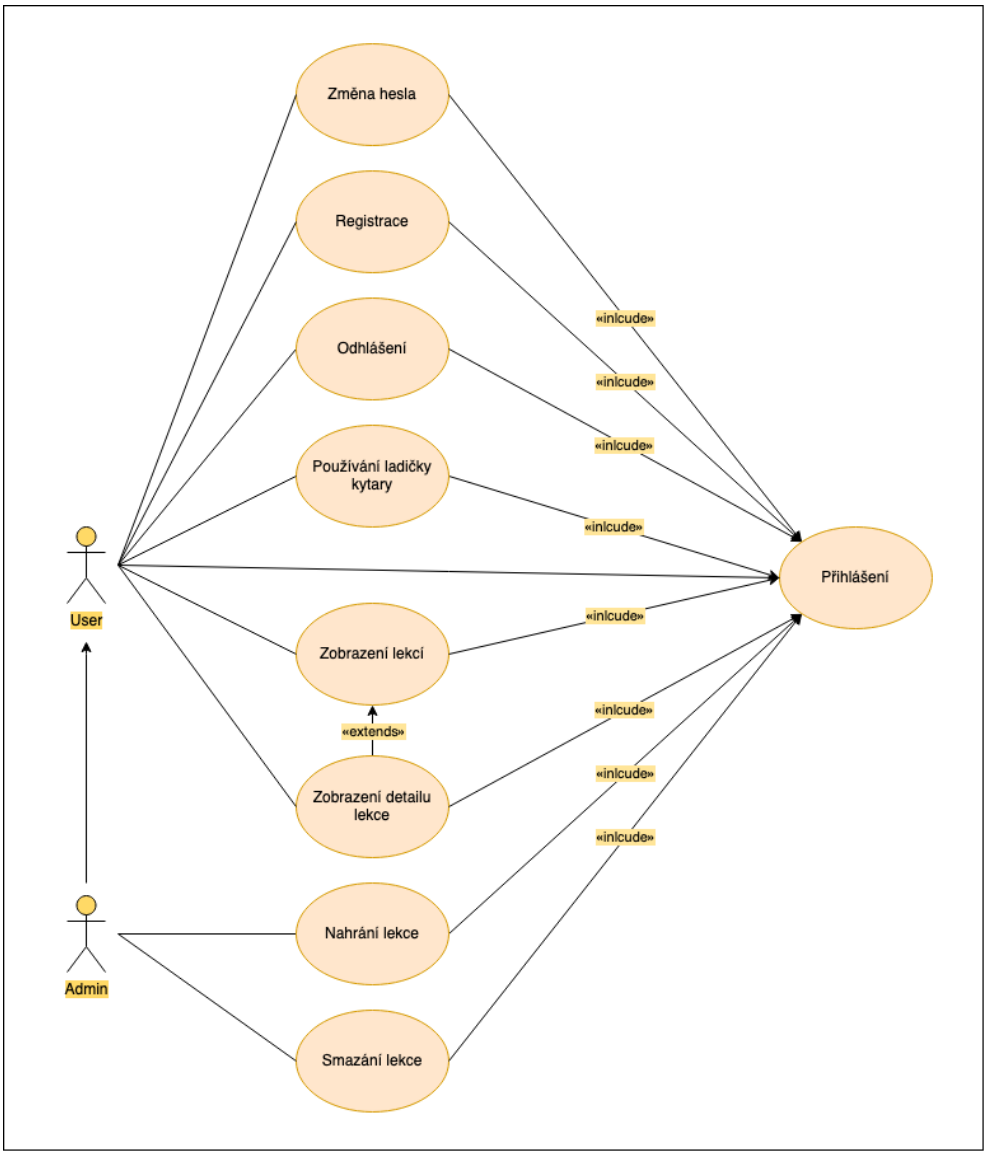
NP1 – iOS Platforma: Platforma výsledného produktu musí být iOS.

NP2 – Jednoduchost: Musí být kladen důraz na dodržení designových norem iOS aplikací.

NP3 – Rozšiřitelnost: Aplikace by měla být jednoduše rozšiřitelná při budoucím vývoji.

2.3.3 Případy užití

Případy užití jsou modelové situace, které prezentují využití funkčních požadavků. Tím lze získat detailnější přehled o funkcionalitě aplikace a jejích konkrétních business procesů. Případy užití musí pokrýt všechny funkční požadavky. Souhrn funkčních požadavků zobrazuje diagram případů užití, obrázek 2.5.



Obrázek 2.5: Diagram případů užití, dle [5]

1. **Registrace**

   Uživateli se zobrazí registrační formulář a je vyzván k vyplnění následujících polí: jméno, email, heslo (splňující bezpečnostní standard) a potvrzení hesla. Poté klikne na tlačítko registrovat. Při korektním vyplnění formuláře se uživateli zobrazí upozornění o odeslaném aktivačním emailu a dodatečných instrukcích. V opačném případě dojde k výzvě opravení chybně vyplněných polí.

2. **Přihlášení**

   Po kliknutí na tlačítko „Login“ ve spodní části obrazovky je uživatel vyzván k zadání svého emailu a hesla do přihlašovacího formuláře. V případě validní kombinace těch údajů je uživatel vpuštěn na domovskou obrazovku aplikace. V opačném případě je uživateli sdělena příčina nesprávného pokusu o přihlášení.

   V případě zapomenutého hesla může uživatel využít tlačítko „Reset Password“, které se nachází ve spodní části přihlašovací obrazovky. Dotyčný je vyzván k zadání svého emailu, na který je mu zaslán email s dodatečnými instrukcemi k obnově hesla (v případě, že účet registrovaný na zadaný email neexistuje, uživatel je upozorněn).

3. **Změna hesla**

   Změnu hesla je možné provést v záložce „Settings“ po přihlášení do aplikace. Po rozkliknutí položky „Change Password“ dojde k otevření formuláře, kde je nutné zadat současné heslo, nové heslo a nakonec nové heslo potvrdit. Nové heslo musí splňovat stejné bezpečnostní standardy jako původní heslo. Při správném vyplnění formuláře je uživatel upozorněn, že jeho heslo bylo změněno. Pokud dojde k nějaké chybě, dojde k upozornění na konkrétní nesrovnalost.

4. **Odhlášení**

   Odhlášení je možné provést pouze pro přihlášené uživatele. Tato položka se nachází v záložce „Settings“ a v daném seznamu je označena jako „Log Out“. Po kliknutí na tuto položku se aplikace uživatele zeptá, jestli se opravdu chce odhlásit. Při kladném potvrzení dojde k odhlášení uživatele a přesměrování na startovací obrazovku aplikace. Uživatel zůstává přihlášen při opětovném zavření a spuštění aplikace do té doby, než se odhlásí.

5. **Zobrazení lekcí**

   Zobrazení lekcí je možné pro přihlášené uživatele. Lekce se nachází v záložce „Tutorials“ na spodní liště aplikace. Zde najdeme seznam všech lekcí, které jsou nabízeny.

6. **Zobrazení detailu lekce**

   Nejprve je nutné zobrazit seznam všech lekcí – tzn. být přihlášen a ve spodní liště aplikace kliknout na položku „Tutorials“. Poté lze kliknout na název vybrané lekce. Tím je uživatel přesněrován na detail této lekce, ve kterém se může nacházet text, video a doplňující obrázky.

7. **Nahrání lekce**

   Nahrávání lekcí je povolené pouze pro administrátorské účty. V případě, že uživatel je přihlášen do aplikace jako administrátor, uvidí v záložce „Settings“ položoku „Upload Tutorial“. Při kliknutí na tuto položku je přesměrován na další obrazovku, kde je vyzván k vyplnění formuláře. Je nutné vyplnit název lekce, text lekce a dále případně nahrát výukové video nebo obrázky. Po vyplnění tohoto formuláře může tlačítkem „Upload“ nahrát novou lekci. O úspěchu publikování nové lekce dojde k upozornění v každém případě – tedy jak při úspěchu, tak při neúspěchu nahrávání.

8. **Smazání lekce**

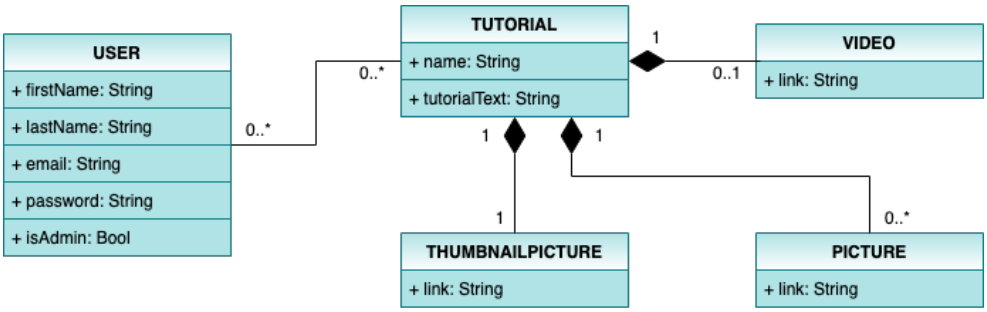
   Přihlášení administrátoři mohou mazat lekce. V záložce „Settings“ lze kliknout na položku „Delete Tutorial“.Po kliknutí na tuto položku seznamu je administrátor přesměrován na další obrazovku, kde musí zvolit lekci k vymazání. Svou volbu potvrdí nejprve kliknutím na tlačítko „Delete“ a poté ještě odsouhlasením další notifikace o mazání obsahu (při které lze ovšem odmítnout, pak vše zůstává v původním stavu).

9. **Používání ladičky kytary**

   V záložce „Tuner“ ve spodním menu se nachází jednoduchá ladička. Při zahrání prázdné struny standardního ladění kytary ladička ukazuje šipkou nahoru, resp. dolu, jak máme upravovat utažení dané struny. Ladičku mohou využívat všichni přihlášení uživatelé.

2.4 Doménový model

Doménový model popisuje entity dané domény a vztahy mezi nimi. Na rozdíl od diagramu tříd, doménový model je implementačně nezávislý. Tento model se obvykle staví na zadání od klienta (v tomto případě na zadání práce), celkově zlepšuje představu o daném problému a podchycení některých detailů. V neposlední řadě pomáhá k sestavení dalších modelů používaných v softwarovém inženýrství (např. databázový model). Konkrétně pro doménový model této bakalářské práce viz následující obrázek 2.6. [6]



Obrázek 2.6: Doménový diagram, dle [6]

2.4.1 User

Entita User reprezentuje jednoho daného uživatele aplikace. Každý uživatel má křestní jméno, příjmení, emailovou adresu a své heslo. Dále je u této entity nutné znát, zda-li se jedná běžného uživatele, nebo administrátora – k tomu slouží atribut typu *Bool* „isAdmin“.

Vazba mezi User a Tutorial (M:N) říká, že uživatel konzumuje obsah lekcí, přesněji 0 až N lekcí, a naopak na každou lekci se může dívat 0 až M uživatelů.

2.4.2 Tutorial

Tutorial znázorňuje kytarovou lekci. Každá lekce má svůj název a text (faktický textový obsah lekce). K lekci navíc patří ThumbnailPicture, Video a Picture. Bez těch by tato entita Tutorial neměla smysl, proto jsou tyto tři entity spojeny s Tutorial vazbou typu kompozice.

2.4.3 Video, Picture, Thumbnail

Každá z těchto entit reprezentuje média (obrázky a video), proto je jejich jediný atribut odkaz na dané médium uložené v databázi. Jak je vidět z násobností vazeb v obrázku 2.6, ke každé lekci patří právě jeden obrázek náhledu, dále libovolný počet výukových obrázků a jedno nebo žádné výukové video.

2.5 Shrnutí analýzy

V této kapitole byla provedena celková analýza podle obvyklých postupů softwarového inženýrství. Od analýzy výuky hraní na kytaru, přes analýzu konkurenčních aplikací a sestavení požadavků (na základě předchozích analýz) po doménový model. Poznatky z této kapitoly budou dále využity při návrhu a výběru technologií. Splněním již zmíněných bodů došlo ke splnění analytické části zadání.

Kapitola 3

Návrh

Po provedení sběru všech požadavků lze začít se samotným návrhem aplikace. Tato kapitola se zabývá popisem nejčastějších architektur a návrhových vzorů, ze kterých bude nakonec vybrána ta nejvhodnější pro vyvíjenou aplikaci. Dále kapitola obsahuje schéma databáze a návrh vzhledu uživatelské rozhraní.

3.1 Architektura

Při výběru architektury bylo hledáno mezi nejpoužívanějšími současnými architekturami a návrhovými vzory, které se používají pro vývoj iOS aplikací. První z nich je velmi populární návrhový vzor *Model-View-Controller* (MVC), dále MVVM (*Model-View-ViewModel*) a nakonec nejnovější z těchto tří – *Clean Architecture*. V následujících sekcích jsou tyto návrhové vzory a architektury rozebrány a v závěrečném bodě pak vybrána ta nejvhodnější možnost pro účely této práce včetně zdůvodnění daného rozhodnutí.

3.1.1 MVC

Model-View-Controller (MVC) je návrhový vzor, který se skládá ze tří komponent – *Model*, *View* a *Controller*. [19] Následuje popis pro každou z nich a poté vysvětlení jejich způsobu komunikace. Sekce je zakončena shrnutím výhod a nevýhod tohoto návrhového vzoru.

Popis jednotlivých komponent

> **Model** se stará o správu dat, což může zahrnovat rozhraní pro perzistentní datové úložiště a datové struktury určené k uchovávání dat v době běhu programu. Součástí *modelu* je také logika umožňující implementaci CRUD operací. [20, str. 176]

Komponenta **View** poskytuje uživateli rozhraní pro interakci s aplikací. [20, str. 176] Jedná se tedy o prezentaci informací pomocí zvoleného UI.

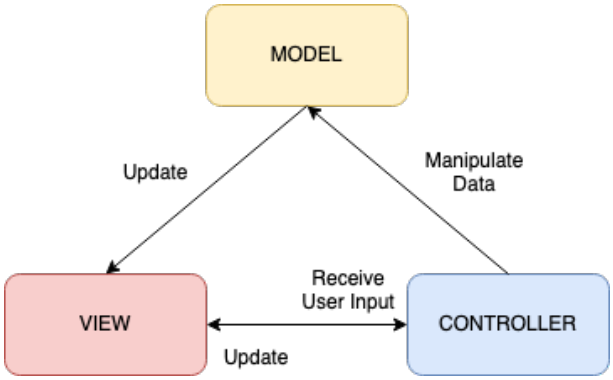
Controller je zodpovědný za business logiku aplikace. Zároveň komunikuje s komponentami Model a View – má za úkol přijímat vstupy od uživatele prostřednictvím *View* a poté na ně náležitě reagovat. [20, str. 176]

Komunikace mezi komponentami

Model odesílá informaci o změně uložených dat na komponentu *View*. Případně může informovat i *Controller* v případě, že je potřeba business logika k řízení daného *View*.

Komponenta *View* přijímá data z *Modelu* (občas i přímo z *Controlleru*), která poté prezentuje uživateli.

Controller reaguje na vstupy z *View* a po aplikování business logiky obnovuje obsah na *View* a v *Modelu*. [7]



Obrázek 3.1: Diagram návrhového vzoru MVC, dle [7]

Výhody a nevýhody

Výhody MVC

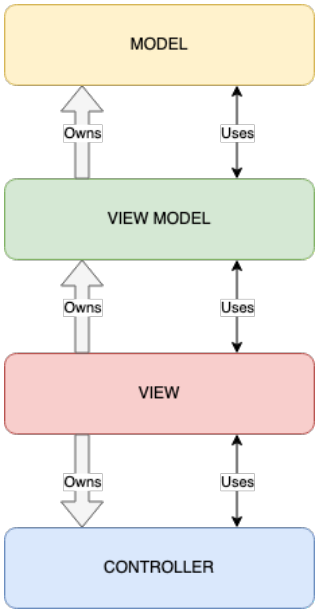
Jednou z hlavních výhod tohoto návrhového vzoru je oddělení uživatelského rozhraní od dat, což umožňuje měnit technologie těchto komponent bez zásahu jedné do druhé. Dále tímto přístupem vzniknou tři komponenty s oddělením zodpovědnosti, tudíž program rozdělený na části, které se z hlediska funkcionality nepřekrývají.

Nevýhody MVC

Tento návrhový vzor může způsobovat problémy ve větších projektech z důvodu velmi rozsáhlých *Controllerů* , které pak obsahují velké množství kódu business logiky. Nicméně u menších a středních projektů se obecně problémy nevyskytují. [21]

3.1.2 MVVM

Návrhový vzor *Model-View-ViewModel* se velmi podobá MVC, avšak řeší problém masivních *Controllerů* přidáním komponenty *ViewModel*. Tato komponenta zodpovídá za správu Modelu a předávání dat *Modelu* do *View* skrze *Controller*. *ViewModel* tedy přebírá úlohu *Controlleru* „přeložení“ dat *Modelu* do podoby, kterou umí prezentovat komponenta *View*. Následující obrázek 3.2 pomůže s lepším pochopením tohoto konceptu. [21]



Obrázek 3.2: Diagram návrhového vzoru MVVM, dle [8]

Výhody a nevýhody

Výhody MVVM

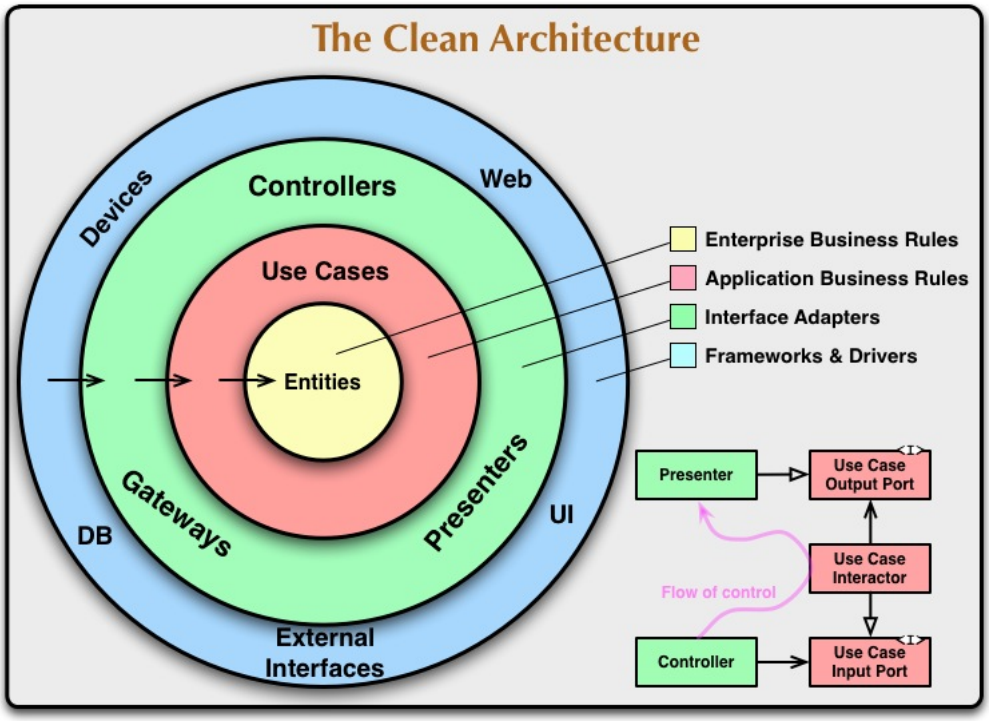
Jak již bylo zmíněno, MVVM má oproti MVC výhodu částečného předvedení kódu z *Controlleru* do *ModelView*. Tím se kód stává přehlednější. Zároveň *ModelView* obvykle umožňuje jednoduché Unit testování, protože nebývá přímo spojen s uživatelským rozhraním dané aplikace.

Nevýhody MVVM

MVVM není vhodný pro jednoduché UI. Navíc implementace každého *ViewModelů* bývá často velmi podobná, takže ve výsledku může nastat situace, kdy v projektu bude velké množství duplicitního kódu. [8]

3.1.3 Clean Architecture

Clean Architecture se stejně jako podobné mnoho jiných architektur snaží oddělit odpovědnosti. Toho lze dosáhnout dělením aplikace na vrstvy. Oddělení vrstev u *Clean Architecture* se řídí tzv. *Dependency Rule*. Samotné vrstvy jsou pak čtyři – *Entities*, *Use Cases*, *Interface Adapters* a *Frameworks and Drivers*. Abstrakce vrstev se zvyšuje směrem do středu kruhu. [9]



Obrázek 3.3: Diagram *Clean Architecture* [9]

The Dependency Rule

Dependency Rule je pravidlo, které říká, že závislosti ve zdrojovém kódu mohou směřovat pouze směrem do středu kruhu v obrázku 3.3. To znamená, že vrstva, která se nachází blíže ke středu nemá žádné informace o vnějších vrstvách. Rovněž datové struktury generované ve vnější vrstvě by neměly být používány vnitřními vrstvami (především pokud se jedná o data frameworků). [9]

Entities

Jedná se o business pravidla. *Entities* je nejdůležitější vrstva, na které stojí všechny ostatní, na kterých je navíc nezávislá. Jedná se o jádro aplikace. Tato vrstva by se neměla v průběhu vývoje měnit. [22, str. 194]

Use Cases

Use Cases reprezentují aplikační logiku aplikace. Obaluje a implementuje všechny případy užití. [22, str. 194]

Interface Adapters

Interface Adapters se chová podobně jako *ViewModel* u MVVM – tedy překládá data mezi vrstvami *Frameworks and Drivers* a *Use Cases*. [22, str. 194]

Frameworks and Drivers

Poslední vrstva *Frameworks and Drivers* se skládá z frameworků, databází a podobných komponent. Tato vrstva se často mění. Navíc se nepředpokládá psaní velkého množství kódu, spíše zde umožňujeme komunikaci již zmíněných komponent (např. databáze) s vnitřními vrstvami *Clean Architecture*. [22, str. 194]

Výhody a nevýhody

Výhody Clean Architecture

Databáze (případně jiné komponenty z vrstvy *Framework and Drivers*) a uživatelské rozhraní jsou zcela nezávislé na zbytku aplikace, proto je lze jednoduše upravovat, měnit a rozšiřovat. Kvůli nízké závislosti mezi vrstvami je umožněno jednoduše testovat jednotlivé moduly.

Nevýhody Clean Architecture

Zvýšená komplexita architektury způsobuje, že i pro jednoduché operace je potřeba napsat velké množství kódu. Proto se *Clean Architecture* využívá spíše u složitějšího softwaru.

3.1.4 Výběr architektury

Po analýze všech zmíněných architektur a návrhových vzorů se jako zcela nejvýhodnější volba jeví stavět vyvíjenou aplikace podle návrhového vzoru MVC a to z několika následujících důvodů:

Velikost aplikace – aplikace není dostatečně komplexní na použití architektury *Clean Architecture*. Navíc vybraná doména neobsahuje velké množství entit, tedy ani výhoda MVVM v tomto ohledu není velmi užitečná.

Doporučení od Apple[1] – dokumentace a vývojářské návody od společnosti Apple doporučují pro podobné projekty vyvíjet podle návrhového vzoru MVC. [23]

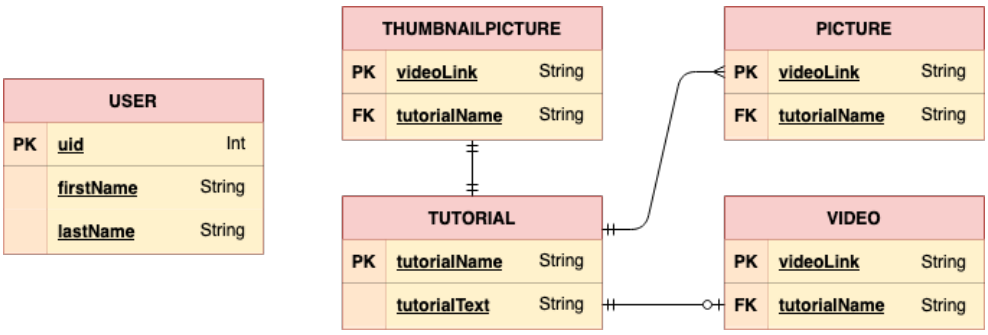
Dostatečná rozšiřitelnost – MVC nabízí dostatečně jednoduchý způsob rozšíření aplikace, což se může hodit při budoucím vývoji.

Jednoduchost – vybraný návrhový vzor je velmi oblíbený a známý a zároveň ze všech tří zmíněných nejjednodušší (pro účel této bakalářské práce). I to je důvod k volbě MVC.

Závěrem lze říci, že byla vybrána vhodná architektura pro vyvíjenou aplikaci a to podle zadní této práce a v souladu se standardními postupy v softwarovém inženýrství.

3.2 Databázové schéma

Na základě doménového modelu a analýzy požadavků je nyní možné sestavit databázové schéma. To definuje fyzické rozvržení systému – tabulky databáze a jejich vztahy pro konkrétní databázový stroj [24, str. 5]. Pro konkrétní databázové schéma viz obrázek 3.4.



Obrázek 3.4: Databázové schéma, dle [10]

V tomto případě se jedná od entitně vztahový model (ang. ERM). I přes to, že databáze, která bude použita (*Firebase*) není sama o sobě relační databází, z programátorského hlediska se s ní bude při vývoji zacházeno jako kdyby relační byla. Primární klíče jsou označeny PK (*Private Key*), cizí klíč pak FK (*Foreign Key*).

[1] celým názvem Apple Inc., dalé vždy „Apple“

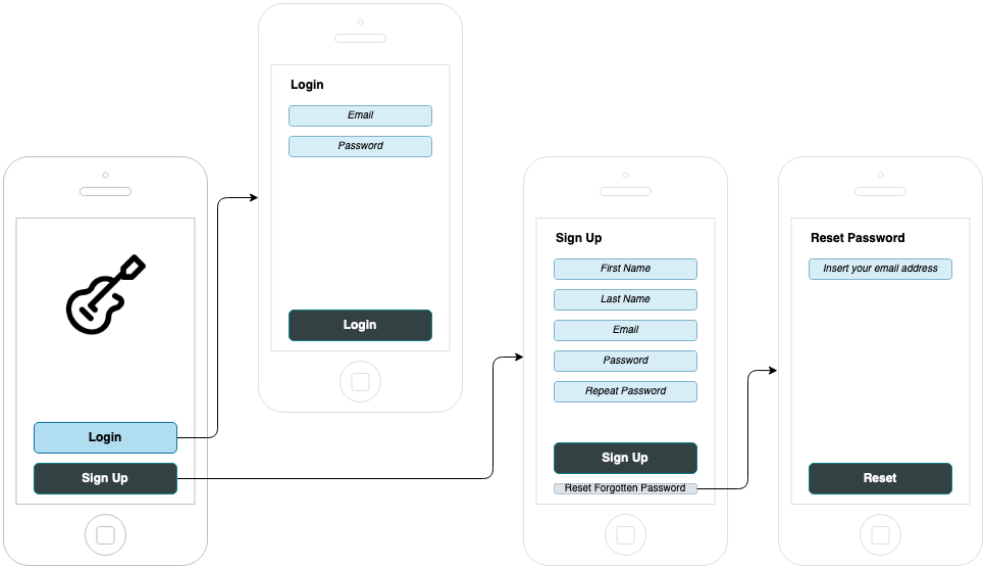
3.3 Uživatelské rozhraní

Poslední část návrhu se věnuje uživatelskému rozhraní. Při tomto procesu bude využit tzv. *iOS Design Language*, tj. soubor pravidel, která Apple prezentuje jako ideální zásady při tvorbě UI aplikací pro iOS [25]. Pro každou důležitou obrazovku aplikace se navrhne *wireframe*[2], což je přibližný vzhled daného uživatelského rozhraní [26]. Následující sekce ukazují přibližný vzhled uživatelského rozhraní jednotlivých částí aplikace a jejich stručný popis.

3.3.1 Autentizace

Design všech obrazovek této sekce se snaží pokud možno co nejvíce čistý a minimalistický. Tlačítka jsou vždy v navzájem kontrastních barvách. Pozadí aplikace zůstává nastavené na systémové pozadí z důvodu podpory *Dark Mode*.

Po spuštění aplikace se uživatel nachází na základní obrazovce autentizačního procesu. Zde si vybírá, zdali se chce přihlásit nebo registrovat. Poté je přesměrován na obrazovku přihlášení (případně obnovy hesla), resp. obrazovku registrace. Všechny obrazovky se snaží dodržet zvolený design. Pro detailnější představu viz obrázek 3.5.



Obrázek 3.5: Wireframe obrazovek autentizace

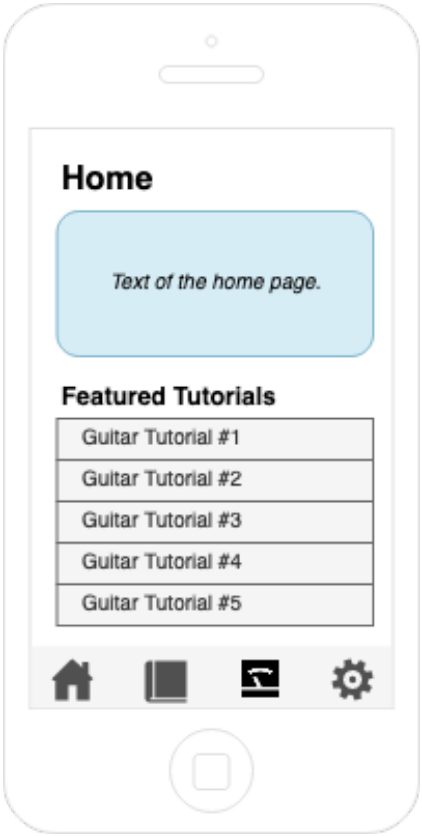
[2]všechny wireframe v této práci jsou tvořeny podle zdroje [26]

3.3.2 Navigační lišta

Po přihlášení do aplikace se uživatel octne na domovské obrazovce aplikace. Mezi jednotlivými hlavními obrazovkami se lze přepínat pomocí navigační lišty, která se vždy nachází ve spodní části obrazovky. Tato lišta obsahu položky Home, Tutorials, Tuner a Settings. Jedna se o nejběžnější způsob navigování v aplikace a zároveň o Applem doporučený způsob návrhu UI [27].

3.3.3 Lekce

Lekce se nachází na dvou různých obrazovkách aplikace. V části *Home*, kde se kromě již zmíněných lekcí nachází i informační text domovské obrazovky, je seznam nejzajímavějších lekcí tak, aby uživatele upoutal ihned po spuštění aplikace.



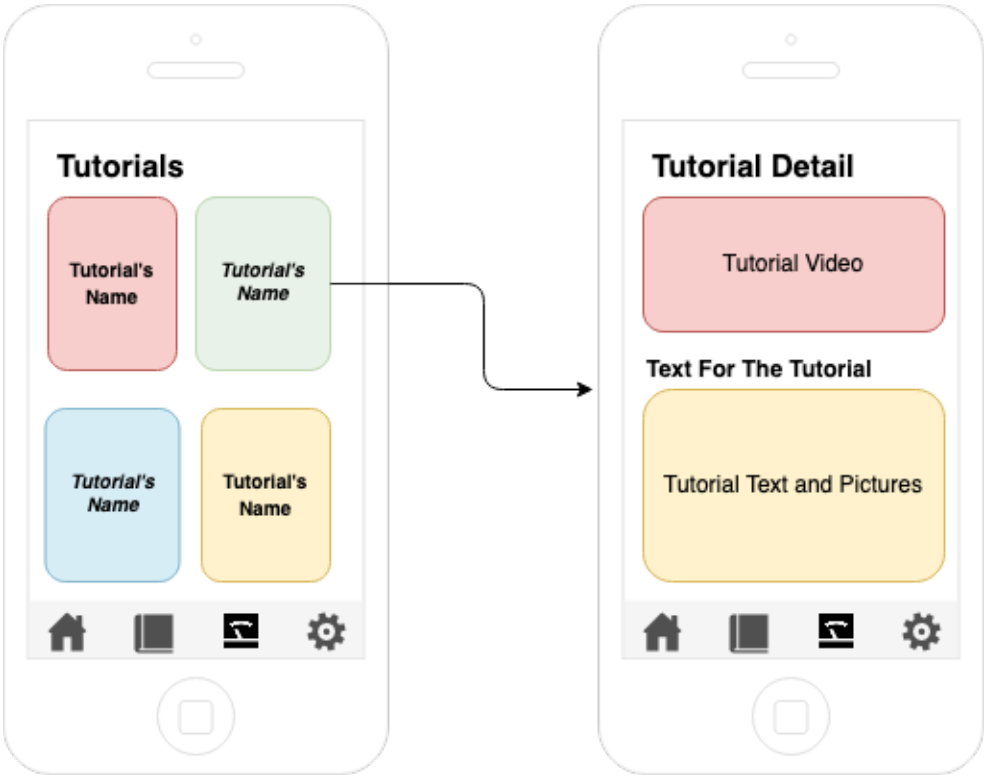
Obrázek 3.6: Wireframe domovské obrazovky

Dále pak v sekci „Tutorials“ lze vidět designově zajímavý seznam všech lekcí. Jedná se o spojení vertikálního a horizontálního seznamu. Jedná a dva seznamy, které jsou v sobě vnořené. V jednom ze seznamů lze posouvat položky

vertikálně a v druhém horizontálně. Celkově pak tento prvek tvoří dlaždicové zobrazení náhledů lekcí.

Při přechodu na detail dané lekce se v hodní části této obrazovky může nacházet video (pokud je pro danou lekce nahrané), poté směrem dolů název lekce a její text s výukovými obrázky.

Z designového hlediska se jedná o jedno z nejčastějších provedení, jak vybírat mezi lekcemi a umožnit jejich prohlížení. Při návrhu bylo tedy možné se inspirovat díky části této práce, kde byly analyzovány konkurenční aplikace, viz 2.2.



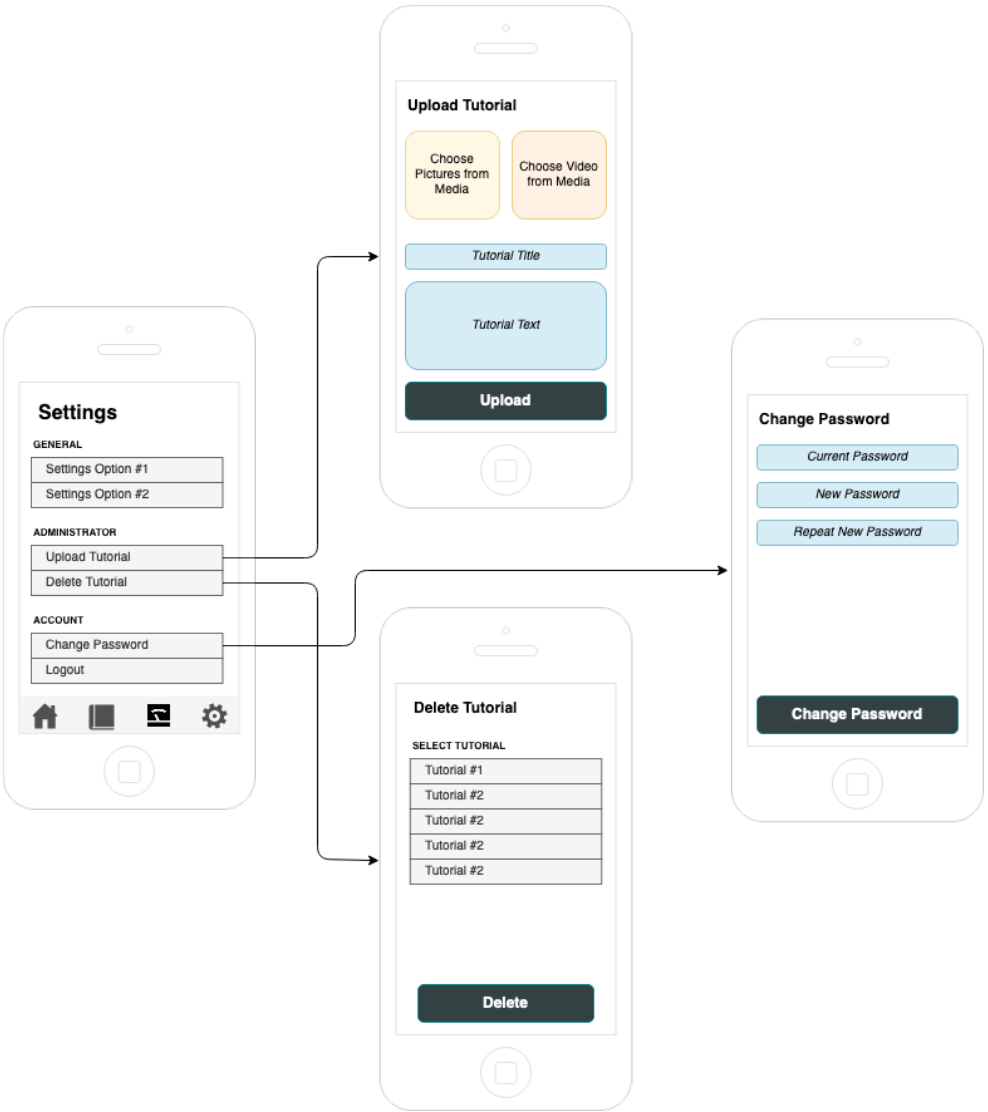
Obrázek 3.7: Wireframe seznamu lekcí a detailu obsahu

3.3.4 Správa uživatelského účtu a lekcí

Pro uživatelské rozhraní nastavení aplikace je použit vertikální seznam rozdělený na skupiny (v tomto případě např. General, Administrator, Account), což je nejčastější řešení a také doporučení od Apple [28]. Díky takovému řešení navíc zůstává zachovaná podpora pro *Dark Mode*.

Pro všechny uživatele se v seznamu nachází položky umožňující změnu hesla a odhlášení. Změna hesla zůstává konzistentní s UI autentizace.

Pro administrátorské účty jsou k dispozici ještě položky pro nahrání, případně smazání lekce. Po přechodu na stránku pro nahrání lekce je administrátor požádán o nahraní videa a výukových obrázku, poté vyplnění názvu lekce a jejího textu. To vše v podobném designovém vzoru jako předešlé obrazovky – kombinace dlaždic a textových polí/tlačítek v kontrastních odstínech. Obrazovka pro smazání dodržuje stejný design.

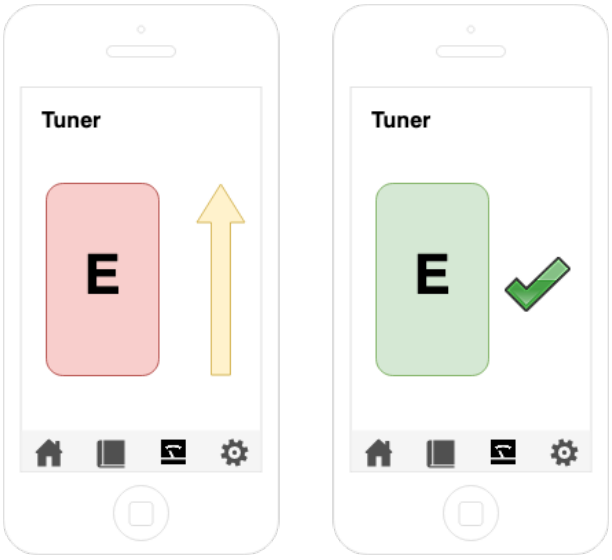


Obrázek 3.8: Wireframe obrazovek umožnující správu aplikace

3.3.5 Ladička

V obrazovce ladičky kytary se nachází jednoduché uživatelské rozhraní – v dlaždici na levé straně se nachází právě laděná struna (její označení podle tónu) a vpravo od ní pak šipka směřující dolu, pokud je potřeba strunu povolit a tím snížit frekvenci hraného tónu, nebo nahoru v opačném případě. Pokud se frekvence tónu, který snímá mikrofon schoduje s referenčním tónem, pak se namísto zmíněné šipky zobrazí zelený symbol značící dobré naladění.

Toto uživatelské rozhraní je velmi jednoduché a přehledné, což se dá určitě považovat za jeho přednost. Navíc se stále daří zachovat předchozí designové prvky tak, aby bylo uživatelské rozhraní co nejvíce konzistentní.



Obrázek 3.9: Wireframe kytarové ladičky

3.4 Shrnutí návrhu

Návrh zahrnuje výběr architektury včetně navržených alternativ, sestrojení databázového schématu a návrh uživatelského rozhraní pomocí vizualizace obrazovek s popisem. Proto lze říci, že se podařilo navrhnout zadanou aplikaci podle běžných postupů softwarového inženýrství a tím splnit body vyplývající ze zadání této práce.

Kapitola 4

Výběr technologií

Kapitola výběr technologií nabízí přehled nejčastějších technologií, které se používají při vývoji mobilních aplikací pro platformu iOS. U každé sekce jsou uvedeny vybrané technologie včetně jejich možných alternativ. Výběr je proveden na základě zhodnocení výhod a nevýhod dané technologie (především vzhledem k účelům software vyvíjeného v rámci této práce) a srovnání s ostatními možnostmi.

4.1 Swift 5

Swift je kompilovaný programovací jazyk vyvíjený společností Apple ve spolupráci s *open-source community*. Jedná se o jeden z jazyků, který lze využít k vyvíjení aplikací pro platformu iOS (a všechny ostatní platformy od Apple). Tento jazyk byl poprvé představen v roce 2014. Do té doby se ke stejnému účelu používal jazyk *Objective-C*, který má stále podporu Apple. [29]

Nelze jednoduše říci, že by jeden z těchto jazyků byl lepší pro iOS vývoj než druhý, nicméně *Swift* je nyní používanějším jazykem. *Objective-C* hraje důležitou roli u starších projektů, které jsou díky podpoře dodnes kompatibilní s Apple platformami. Použití obou těchto jazyků přináší své výhody a nevýhody. Přehledné porovnání nabízí následující tabulka 4.1.

Tabulka 4.1: Tabulka srovnání jazyků *Swift* a *Objective-C*, dle [11]

SWIFT	**OBJECTIVE-C**
VÝHODY	**VÝHODY**
Rozmanitost frameworků – `SwiftUI`, `UIKit`, *Storayboards*, aj.	Velmi stabilní a odzkoušený jazyk
Mízké množství kódu při vývoji (oproti *Objective-C*)	Kvalitní dokumentace
Mnoho mechanizmů moderních jazyků – ochrana neoprávněného přístupu k paměti, apod.	**NEVÝHODY**
Jedná se o velmi rychlý jazyk, co se týče běhu výsledného programu	Málo nových vývojových nástrojů
NEVÝHODY	Snižující se počet vývojářů, kteří používají tento jazyk
Občasné známky nestability jazyka, *Swift* je stále ve vývoji	I pro malý program je potřeba napsat velké množství kódu

Při vývoji aplikace pro tuto bakalářskou práci bude nejvhodnější použít jazyk *Swift*, především pak kvůli frameworkům, které nabízí. Ty velmi usnadní a urychlí celý vývoj. Výchozí volbou se pak stává nejnovější verze tohoto jazyka, tedy *Swift 5*. Zároveň se jedná o současně populárnější a Applem propalovanější volbu [11], což je další důvod k výběru jazyka *Swift*.

4.2 Výběr IDE

V této sekci je vysvětleno, které IDE a proč je vhodné použít pro vývoj aplikace z této práce. Jaké výhody přinese jeho používání a jaké existují alternativní možnosti.

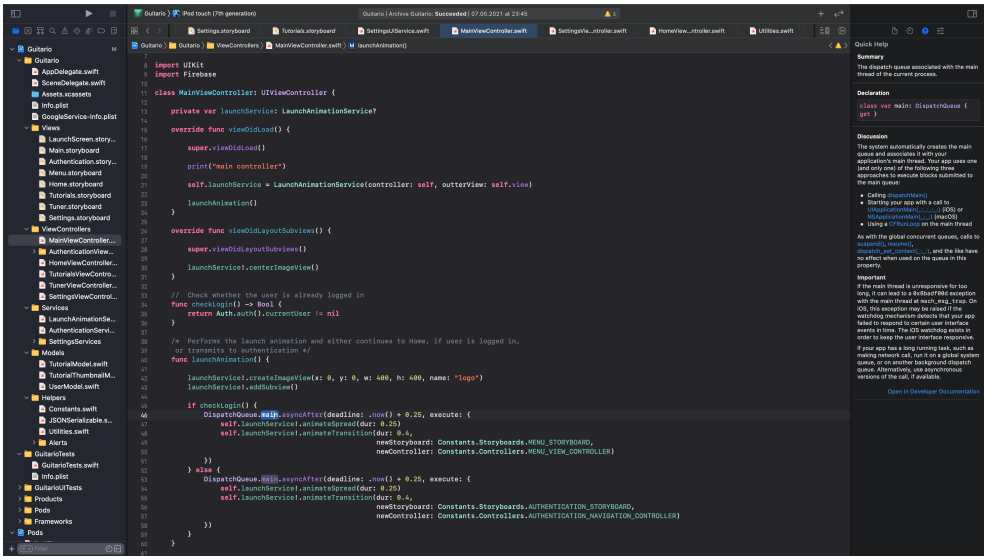
4.2.1 Xcode

Xcode je IDE od společnosti Apple. Jedná se o hlavní softwarové studio pro vývoj aplikací pro operační systémy iOS, iPadOS, watchOS, tvOS a macOS. Také je možné vyvíjet multiplatformní aplikace, nicméně hlavní účel *Xcode* je poskytnout prostředí pro vývoj software pro již zmíněné operační systémy, které vyvíjí právě Apple. Toto IDE existuje pouze ve verzi pro macOS a to jak ve verzi pro procesory postavené na architektuře x86, tak na zařízeních s procesorem typu ARM. [30]

Xcode nabízí širokou škálu připravených šablon, které mohou usnadnit vývoj (např. šablona pro vývoj hry na iOS zařízení). Navíc má v sobě zabudovaný *Interface Builder* pro mobilní aplikace, což je nástroj, který umožňuje rychlou a jednoduchou tvorbu uživatelského rozhraní pomocí tzv. *storyboards*. Zároveň lze použít jeden ze čtyř podporovaných jazyků programovacích jazyků

(*C*, *C++*, *Swift*, *Objective-C*). Jiné jazyky lze též použít, ale většinou to není vhodné. [30]

Jedna z výhod *Xcode* spočívá v podpoře přímo od autora operačních systémů, pro které je vývoj v *Xcode* určen. Z toho důvodu dochází k aktualizacím nástrojů potřebných pro vývoj na nejnovějších verzích těchto OS od Apple vždy velmi rychle. Například vždy při vydání nové verze iOS vychází i aktualizace pro *Xcode*. [31] Nevýhodou může být občasné nečekané restartování nebo zasekávání, avšak to se ze zkušenosti autora této práce děje spíše výjimečně.



Obrázek 4.1: Ukázka *Xcode* IDE

4.2.2 Alternativní řešení

Nejčastějším alternativním řešením je používání IDE *AppCode* od společnost JetBrains. To podporuje vývoj v jazycích *Swift*, *Objective-C* a *C++*. Výhodou je podpora operačních systému Windows a Linux, tedy nejen macOS. Toto IDE však neobsahuje výše zmíněné šablony ani *Interface Builder*. Navíc na jiných operačních systémech (mimo macOS) není možné spouštět simulátory iOS zařízení, takže testování aplikace je možné pouze na fyzickém zařízení. [32, 33]

Při vývoji multi-platformních aplikací, např. ve pomocí frameworku *Flutter*, lze využít např. *Android Studio*, taktéž od JetBrains. V takovém případě může být software vyvíjen v jazyce Dart, který je ovšem ve výsledku přeložen do jazyka *Swift*. Takový vývoj má podobné nevýhody jako vývoj v již zmíněném *AppCode* – obtížnější testování aplikace, chybějící *Interface Builder*, aj.

Zřejmou výhodou je pak výsledná multi-platformní aplikace, jejíž kód lze přeložit do spustitelné aplikace jak pro Android, tak pro iOS. [34]

4.2.3 Shrnutí výběru IDE

V případě této bakalářské práce, kdy se počítá s výslednou aplikací určenou pro platformu iOS a vývojem na operačním systému macOS, se jeví jako optimální výběr *Xcode* a to jak z důvodu výhod podpory ze strany Apple, tak ostatních benefitů popsaných výše. Navíc možnost použití integrovaného *Interface Builderu* může značně zjednodušit a urychlit celý proces implementace.

4.3 Tvorba uživatelského rozhraní

Při tvorbě uživatelského rozhraní pro iOS aplikace lze postupovat několika způsoby. Každý z těchto způsobů přináší své výhody i nevýhody. Různé přístupy lze také kombinovat, nicméně ne vždy se takový postup jeví jako nejlepší řešení.

V následujících třech sekcích se nachází popis každého z přístupů, stejně tak i jejich hlavní výhody a nevýhody.

4.3.1 UIKit

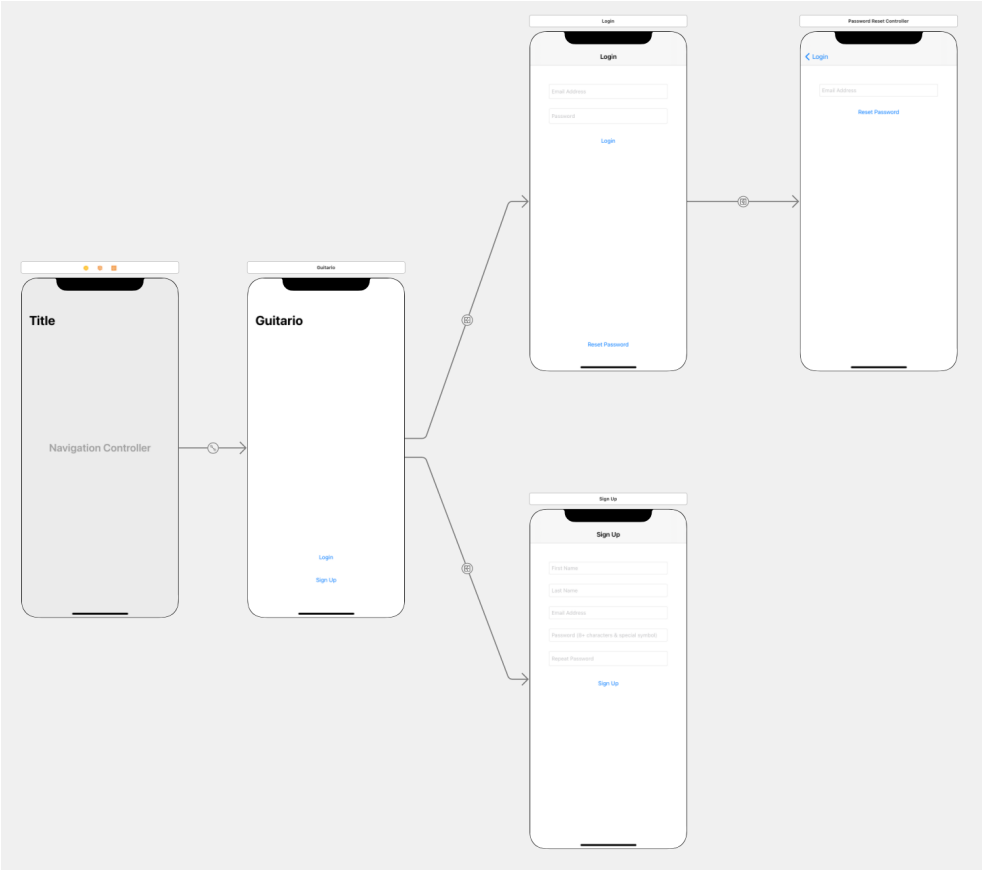
`UIKit` je framework, který slouží k postavení událostně řízeného (*event-driven*) grafického uživatelského rozhraní (GUI) iOS aplikací. Objekty tohoto frameworku reprezentují scény (obrazovky aplikace) a elementy `(UI Objects)`, ze kterých se tyto obrazovky skládají. Při interakci uživatele s těmito elementy může dojít k vyvolání události `(UI Event)` na kterou lze následně adekvátně reagovat (např. při kliknutí na objekt reprezentující tlačítko se změní obrazovka aplikace). [35]

Při využití tříd `UIKit` v kódu se hovoří o tzv. programátorském přístupu k tvorbě UI (*Programmatic UI*). Výhodou tohoto přístupu je rychlost. Zkušený programátor dokáže pomocí tohoto frameworku velmi rychle a efektivně zkonstruovat dobře vypadající uživatelské rozhraní. Tento postup lze bez problémů uplatnit jak v menších, tak ve velkých projektech. [36]

Nevýhodou pak může být určitá náročnost – potřeba znát příslušné objekty a postupy. Tento přístup dále vyžaduje určitou představivost a zkušenost s vývojem UI (zkušenost s vytvářením UI webových stránek může značně zjednodušit pochopení `UIKit`).

4.3.2 Storyboards

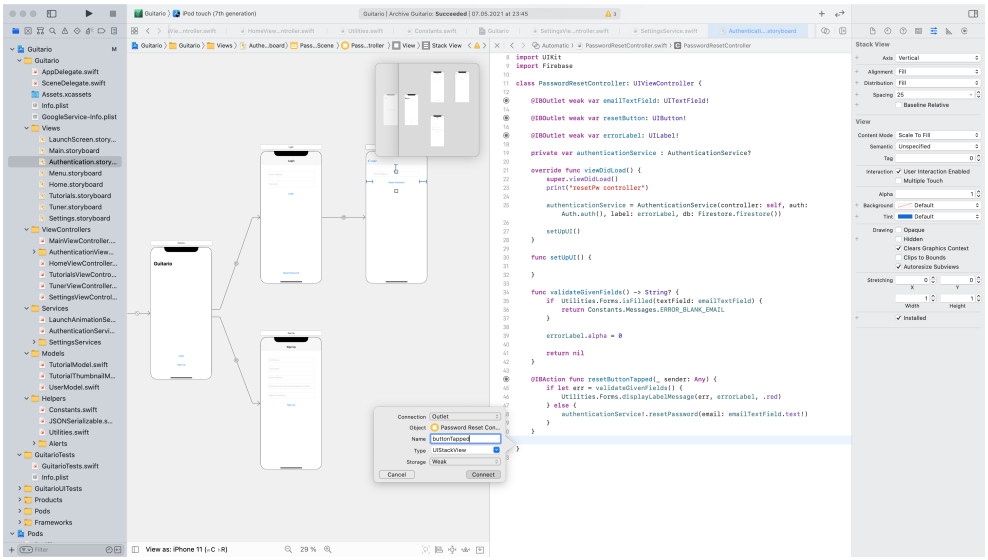
V IDE *Xcode*, jak již bylo zmíněno v sekcí Výběr IDE 4.2, je možné používat integrovaný *Interface Builder*, který umožňuje vytvářet UI pomocí tzv. *storyboards*. Každý *storyboard* reprezentuje jednu danou obrazovku ve výsledné aplikaci. [37] Viz obrázek 4.2.



Obrázek 4.2: Ukázka *storyboards* v *Xcode* IDE

Pomocí *storyboards* lze rozmístit různé prvky UI do dané obrazovky (např. tlačítka, seznamy, apod.) a následně i definovat jejich design (barva, tvar, různé animace, apod.). Tyto elementy lze rovněž jednoduše propojovat s kódem pomocí tzv. `IB Outlets` – viz obrázek 4.3. Tudíž poté je možné definovat akci (kódem programu), která nastane např. při stisknutí zmíněného tlačítka. [38]

Storyboard se skládá ze scény (*Scene*) a UI elementů `(UI Objects)`, ze kterých se scéna skládá. Mezi jednotlivými obrazovkami lze přecházet pomocí tzv. *Segues*, které v obrázku 4.2 znázorňují šipky mezi obrazovkami. Tyto přechody jsou vyvolány při nějaké akci `(IB Actions)`. [39, str. 209–217]



Obrázek 4.3: Ukázka `IB Outlets`

Zřejmá výhoda *storyboards* spočívá v tom, že pro vytvoření UI není potřeba psát žádný kód. Vývojář (případně klient) zároveň okamžitě vidí vzhled obrazovek aplikace, což značně ulehčí představu o výsledku. [39, str. 209–217]

Na druhou stranu tento přístup nemusí vždy být obvykle nejrychlejší – např. při zdlouhavém nastavování atributů některých UI elementů v dlouhých seznamech. V takovém případě je jednodušší použít jiný způsob tvorby UI. U velkých projektů se tento způsob téměř nepoužívá z důvodu komplexity aplikace – pro takový projekt by byly *storyboards* příliš rozsáhlé a nepřehledné. [39, str. 209–217]

Nejčastější způsob vývoje uživatelského rozhraní spočívá v kombinování tvorby UI programátorským způsobem pomocí `UIKit` a právě *storyboards*. *Storyboards* obvykle poskytnou možnost vytvořit rozložení prvků UI a jejich spojení se zdrojovým kódem, nicméně design těchto elementů je pak uskutečněn pomocí funkcí frameworku `UIKit` (je jednodušší zavolat desetkrát stenou funkci na nastavení designu tlačítka, než každé tlačítko upravovat zvlášť v *Iterface Builderu* pomocí *storyboards*).

4.3.3 SwiftUI

Nejnovějším přístupem k tvorbě uživatelského rozhraní je `SwiftUI`. Tento framework byl zveřejněn v posledním čtvrtletí roku 2019 a je podporován pouze na iOS ve verzích 13.0 a novějších. `SwiftUI` má za cíl zjednodušení a zrychlení vývoje iOS aplikací – deklarativní programování UI, kratší kód než u `UIKit`, aj. Jedná se stejně jako u `UIKit` o programátorský přístup k tvoření UI. [40]

Výhodou `SwiftUI` je rychlost tvorby uživatelského rozhraní, přehledný kód, relativně jednoduchý styl programování na naučení. Jelikož se jedná o nový framework, `SwiftUI` řeší některé problémy, na které trpěl `UIKit`. Např. padání aplikací při zapomenuté referenci `@IBOutlet` nebo některé problémy s automatickým rozvrhováním obsahu scény. [41]

`SwiftUI` lze kombinovat s `UIKit`, nicméně není možné jej použít společně se *storyboards*. To není nutně nevýhoda, ale někomu může vizualizovaný *Interface Builder* chybět. Jistou nevýhodou však zůstává podpora pouze na zařízeních s iOS 13.0 a novějším. [41]

4.3.4 Shrnutí tvorby UI

Pro účely této bakalářské práce se jeví jako nejvýhodnější zvolit kombinaci frameworku `UIKit` a integrovaného *Interface Builderu*. Tento projekt se řadí spíše mezi menší (případně středně velké), tudíž by neměl být problém v přehlednosti a komplexitě *storyboards*. Při této kombinaci lze vytvořit rozmístění objektů každé scény pomocí *Interface Builderu* a následně upravit design těchto objektů pomocí `UIKit` a programátorského přístupu.

`SwiftUI` nebyl vybrán z důvodu chybějící okamžité zpětné vazby vzhledu uživatelského rozhraní, které nabízejí *storyboards* a také kvůli nedostatečné podpoře starších iOS zařízení.

4.4 Firebase Backend

Firebase od společnosti Google[3] nabízí velké množství backendových služeb nejen pro platformu iOS. Jejich využití značně urychluje vývoj iOS aplikací a zaručuje dostatečnou míru bezpečnosti. Používaní těchto služeb bývá relativně jednoduché, navíc je k dispozici kvalitní online dokumentace. [42]

V případě iOS stačí nainstalovat knihovnu *Firebase* pomocí *CocoaPods* a poté vždy jen použít *include Firebase* při používání v kódu. Pro detailnější představu, jak používat *Fireabse* přímo v kódu viz následující kapitolu 5.1.1.

Pro potřeby této práce bude nutné používat službu *Firebase Authentication* pro správu uživatelských účtů, dále *Firestore Database* a *Firebase Storage* k ukládání dodatečných informací a výukových médií.

[3] celým názvem Google LLC, dalé vždy „Google“

4.4.1 Firebase Authentication

Firebase Authentication je služba umožňující bezpečné evidování identity uživatelů. Díky této službě může být uživatel autentizován při každém vstupu do aplikace. Tuto autentizaci lze provést mnoha různými způsoby. Nejzákladnějším způsobem je kombinace emailu a hesla. Dalším řešením pak může být ověření pomocí účtů třetích stran, např. od Google, Twitter, Apple, Facebook, aj. *Firebase Authentication* nabízí i registraci (kombinace emailu a hesla) s potvrzovacím emailem. [43]

4.4.2 Cloud Firestore

Tato backendová služba umožňuje uchování souborů v cloudovém úložisti. *Cloud Firestore* je tzv. *NoSQL* databáze. Funguje jako obyčejný adresář s dokumenty, kde lze každý dokument identifikovat pomocí jedinečné url adresy. Dokumenty tvoří kolekce, které pomáhají k lepší organizaci a zpřehlednění obsahu úložiště. Tato databáze podporuje velké množství datových typů. [44]

Cloud Firestore umožňuje přístup k datům a operace nad nimi (např. seřazení) pomocí knihovny, kterou lze jednoduše nainstalovat pomocí *CocoaPods*. [44] Pro účely této bakalářské práce bude *Cloud Firestore* využito k uchovávání výukových videí a obrázků.

4.4.3 Firestore Database

Firestore Database slouží jako synchronizační nástroj pro *Cloud Firestore* a k ukládání dodatečných informací. Data v této databázi jsou uložena ve formátu JSON a synchronizována (*realtime synchronization*) s každým připojeným klientem. [45] *Firestore Database* bude využita k ukládání lekcí, kde každý záznam bude obsahovat informace o dané lekci a url adresy všech potřebných médií uložených v *Cloud Firestore*. Pro lepší ilustraci viz databázové schéma (obrázek 3.4).

4.5 Shrnutí výběru technologií

Za použití poznatků z kapitol analýza (2) a návrh (3) se podařilo vybrat vhodné technologie, které budou dále využity při implementaci a testování vyvíjené aplikace. Uzavřením této části došlo ke splnění bodu „výběr technologií“ ze zadání této bakalářské práce a to podle obvyklých postupů softwarového inženýrství.

Kapitola 5

Implementace

Díky předešlým kapitolám, kde se podařilo správně analyzovat doménu, sestavit požadavky na aplikaci, vytvořit návrh a vybrat vhodné technologie může začít realizační část práce.

Tato kapitola se zabývá procesem spojení vyvíjené aplikace s backendovou službou *Firebase*, poté implementací samotné aplikace podle stanovených funkčních a nefunkčních požadavků a to včetně ukázek významných částí zdrojových kódů.

5.1 Použití Firebase

V této sekci je stručně popsáno, jak probíhá proces konfigurace *Xcode* projektu ke zprovoznění služeb *Firebase*, které byly zvoleny v rámci výběru technologií v kapitole 4. Konfigurace této služby závisí na platformě, pro kterou je produkt vyvíjen. V této práci bude popsán postup pro vyvíjení iOS aplikací pomocí IDE *Xcode*. K dispozici jsou i ukázky významných částí zdrojových kódů.

5.1.1 Počáteční spojení

Jako první krok přichází na řadu vytvoření *Xcode* projektu, který musí být nastaven jako aplikace pro platformu iOS se zvolením jazyka *Swift* a tvorbou uživatelského rozhraní pomocí *storyboards*. Poté je potřeba vytvořit projekt na webu *Firebase*. Tento projekt musí být asociován s daným *Xcode* projektem. Kroky tohoto procesu popisují následující tři odstavce: [46]

Bundle ID

První krok tohoto procesu je nastavení tzv. *bundle ID*, což je jedinečný identifikátor *Xcode* projektu. Tento identifikátor je popsán v hlavním konfiguračním souboru projektu. [46, 47]

Konfigurační soubor pro Firebase

Poté je potřeba vložit do hlavní složky projektu *Firebase* konfigurační soubor `GoogleService-Info.plist`. Tento soubor obsahuje veřejné informace potřebné ke správné identifikaci *Firebase* a lze jej stáhnout z webové stránky příslušného projektu *Firebase*. [46]

Přidání Firebase SDK

Přidání *Firebase* SDK umožní používání *Firebase* knihoven ve zdrojovém kódu. Pro přidání lze použít nástroj *CocoaPods*, což je správce závislostí pro *Swift* a *Objective-C*. [46, 48]

Tento nástroj lze nainstalovat např. pomocí *package manageru Homebrew* příkazem `brew install pods`.

Poté je potřeba v hlavní složce daného *Xcode* projektu pomocí příkazu `pods init` vytvořit soubor `Podfile`, konfigurační soubor *CocoaPods*. Do tohoto souboru se poté musí přidat názvy knihoven, které budou použity při vývoji. Pro účely této bakalářské práce se jedná o:

`pod 'Firebase/Auth'` pro *Firebase Authentization*

`pod 'Firebase/Core'` obsahuje esenciální knihovny *Firebase*

`pod 'Firebase/Firestore'` pro *Firestore* a *Firebase Cloud*

Nakonec se příkazem `pods install` spustí instalace samotných závislostí. Po provedení tohoto procesu je možné používat `include Firebase` ve zdrojových kódech konfigurovaného projektu a využívat tak služby *Firebase*.

Inifializace Firebase

Při každém spuštění aplikace je nutné inicializovat *Firebase*. Tato inicializace je provedena v souboru `AppDelegate.swift`, který je součástí každého *Xcode* iOS projektu. Tento soubor obsahuje metody, které se spouští v různých fázích běhu aplikace. Následující kód ukazuje přesné použití inicializační metody `FirebaseApp.configure()` po startu aplikace. [46]

```
import Firebase

...

func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey: Any]?
                ) -> Bool {

        /* Override point for customization after
          application launch. */

        FirebaseApp.configure()

        return true
}
```

Po provedení výše zmíněných kroků lze naplno využívat backendové služby *Firebase*. Tento krok je důležitou částí implementace vyvíjené aplikace. Zmíněné služby jsou dále využívány téměř ve všech modulech aplikace, což může být vidět i v dalších částí této kapitoly.

5.2 Ukázky implementace

V této sekci jsou prezentovány významné části implementace ze všech důležitých částí vyvíjené aplikace, tedy z části autentizace uživatele, zobrazování lekcí a správy aplikace. Popis průběhu implementace je doplněn ukázkami zdrojových kódů.

Implementace proběhla tak, aby splnila stanovené funkční a nefunkční požadavky adekvátních priorit a to podle vytvořeného návrhu a technologií vybraných v předchozích kapitolách. Došlo tedy k použití návrhového vzoru MVC, vývoji v IDE *Xcode* za použití *storyborads* pro tvorbu UI. Celá aplikace pak používá backendovou službu *Firebase*.

5.2.1 Autentizace

V autentizační části byly implementované funkční požadavky registrace (s potvrzovacím emailem), přihlášení a resetování zapomenutého hesla. Zároveň bylo implementováno uživatelské rozhraní na základě předchozího návrhu.

Způsob implementace u všech zmíněných požadavků se velmi podobá, proto je v této práci uveden jen jeden konkrétní příklad (registrace) implementace. Následuje popis celého procesu vývoje této funkcionality.

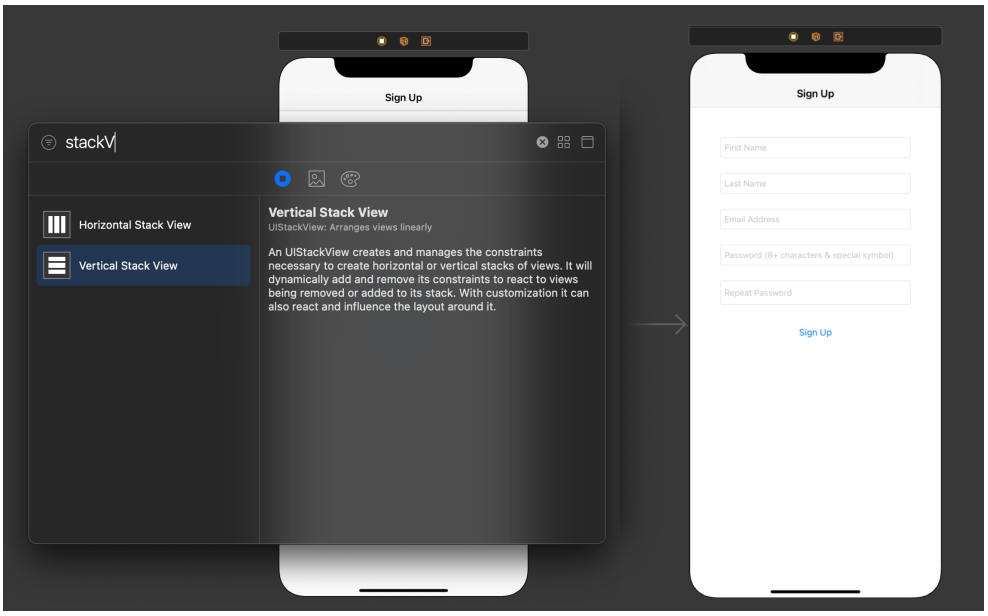
Prvním krokem vývoje je vytvoření příslušného *controlleru* a *storyboard* (*view*). V tomto případě bude vytvořen soubor `Auth.storyboard`, který bude obsahovat všechny obrazovky procesu autentizace a rovněž `SignUpController.swift`, který bude řídit proces registrace. *Controller* má každá obrazovka svůj. Nutně také musí být vytvořen *model* pro uživatele, viz následující kód.

```
struct UserModel {

    let firstName : String

    let lastName  : String

    let UID       : String
}

extension UserModel     : JSONSerializable {
    var encode: AnyObject {
        let dictionary. : [String: AnyObject] =
            ["firstName": firstName.encode,
             "lastName" : lastName.encode,
             "UID"      : UID.encode]

        return dictionary as AnyObject
    }

    static func decode(_ json: Any?) -> UserModel? {
        guard let json  = json as? [String: AnyObject],
              let fName = String.decode(json["firstName"]),
              let lName = String.decode(json["lastName"]),
              let uid   = String.decode(json["UID"])
        else {
            return .none
        }

        return UserModel(firstName : fName,
                         lastName  : lName,
                         UID       : uid)
    }
}
```

Jelikož je známé, jaká data budou o uživateli uložena do *Firestore* databáze (která jak již bylo zmíněno v předchozích kapitolách ukládá data ve formátu JSON), je na místě rovnou implementovat protokol `JSONSerializable`. To

umožní jednoduchou serializaci, resp. deserializaci, do, resp. z tohoto datového formátu.

Nyní již byly vytvořeny všechny tři komponenty MVC, proto se může začít se samotnou implementací. Jako první je vhodné implementovat uživatelské rozhraní. Jeho implementace proběhne ve dvou fázích. Nejprve bude utvořena scéna pro registraci v `Auth.storyboard` a umístěny do ní objekty frameworku `UIKit`. Tento proces ukazuje následující obrázek 5.1.



Obrázek 5.1: Ukázka implementace UI `Auth.storyboard`

Tato scéna se nyní skládá z jednoho `UIView` zakrývajícího celou scénu. V tomto *view* se ve vrchní části nachází `UIStackView`, což je *container*, ve kterém se jednotlivé položky skládají pod sebe, konkrétně `UITextField` – textová pole, `UILabel`, který bude sloužit k upozornění uživatele na nesprávně zadané údaje a potvrzovací `UIButton`.

Dalším krokem je propojení jednotlivých objektů scény s příslušným *controllerem* pomocí `IBOutlet`. Dále pak ještě musí být propojeny i akce (pomocí `IBAction`), které mohou v daném *view* nastat. V tomto případě pouze stisk jediného tlačítka.

Po provedení všech zmíněných propojení by měl výsledný zdrojový kód v příslušném *controlleru* vypadat následovně.

```
import UIKit

class SignUpController: UIViewController {

    // outlets to the UI objects
    @IBOutlet weak var firstNameTextField: UITextField!
    @IBOutlet weak var lastNameTextField: UITextField!
    @IBOutlet weak var emailTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var repeatPasswordTextField: UITextField!
    @IBOutlet weak var signUpButton: UIButton!
    @IBOutlet weak var errorLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        // runs after loading the main view

         setUpUI()
    }

    @IBAction func signUpButtonTapped(_ sender: Any) {
        // runs when signUp button is tapped
    }
}
```

Druhou fází tvorby UI je implementace metody `setUpUI()`, která se spustí bezprostředně po načtení hlavního *view*, ve funkci `viewDidLoad()`. Tato metoda nastaví design všech UI elementů. Část implementace této metody může vypadat například následovně:

```
private func setUpUI() {

    // set up the SignUp button design
    signUpButton.backgroundColor = UIColor.init(named: "Blue")
    signUpButton.tintColor = UIColor.init(named: "Cyan")
    signUpButton.layer.cornerRadius = 20.0

    // set up other UI elements
    ...
}
```

Uživatelské rozhraní je v tuto chvíli zcela hotové, tudíž zbývá implementace samotné registrace. K tomu se využívají metody knihovny `Firebase/Auth` umožňující komunikaci s backendovou službou autentizace. Na tuto službu

jsou pomocí příslušného API posílány požadavky, na které se poté náležitě reaguje.

K ukládání dodatečných údajů o uživateli (viz zdrojový kód pro UserModel na začátku sekce 5.2.1) bude použito *Firestore Database* a její knihovna `Firebase/Firestore`.

Proces registrace začíná po kliknutí na tlačítko „Sign Up“, které je, jak již bylo zmíněno, spojeno s funkcí `SignUpButtonTapped()`.

V prvním kroku je nutné zkontrolovat všechna potřebná pole, jestli jsou ve správném formátu, jestli se shodují hesla a splňují stanovené požadavky na bezpečnost, apod. k tomu poslouží metoda `validateFields()`.

```
import UIKit

class SignUpController: UIViewController {
    ...

    override func viewDidLoad() {
        super.viewDidLoad()
        ...
    }

    @IBAction func signUpButtonTapped(_ sender: Any) {
        // runs when signUp button is tapped
        validateFields()
        createUser()
    }

    private func createUser() {
        // create user and store data into the database
    }

    private func validateFields() {
        // validate all fields
    }

    ...
}
```

Samotnou registraci pak implementuje metoda `createUser()`. V této metodě dojde nejprve o pokus registrace uživatele (včetně potvrzovacího emailu) pomocí *Firebase Authentication* (s náležitou reakcí na případný neúspěch) a poté k uložení doplňujících údajů do *Firestore Database*, opět s podchycením případné chyby. Následující zdrojový kód ukazuje tento proces (pozn. tělo metody `createUser()` není odsazené z důvodu nedostatku místa).

```
private func createUser() {
// create an authentication session
let session = Auth.auth()

// create a database connection
let db = Firestore.firestore()

// attempt to create a new user with email and password
session.createUser(
    withEmail  : emailTextField.text!,
    password   : passwordTextField.text!,
    completion : { result, error in

    if error != nil {
        // show error using UILabel, if something went wrong
    } else {
        // else send an activation link to the user
        result!.user.sendEmailVerification(completion: nil)

        // create a model object with collected data
        let userModel = UserModel(
            firstName : firstNameTextField.text!,
            lastName  : lastNameTextField.text!,
            UID       : result!.user.uid)

        // store additional user data into the database
        db.collection(COLLECTION_NAME).addDocument(
            data: userModel.encode as! [String: AnyObject])
            { error in

            if error != nil {
                // let the user know if there was an error
            } else {
                /* else notify the user about success with
                   UILabel or overscreen alert */
                // make a segue to another screen of the app
            }
        }
    }
})
}
```

Tím byl ukázán celý proces registrace – vytvoření uživatele, zaslání potvrzovacího emailu a uložení dodatečných dat o uživateli do databáze. Ostatní

procesy autentizace (a z velké části i u lekcí a ladičky) jsou principiálně velmi podobné a nemá je tedy proto smysl dále rozebírat.

V následujících sekcích budou vyzdvihnuty odlišené části implementace, nicméně samotný postup se nijak zásadně nemění – vždy se nejprve uspořádá UI pomocí *storyboards*, dále spojí jeho objekty s kódem příslušného *controlleru*, vytvoří design těchto objektů a nakonec je napsána samotná logika uvnitř *controlleru*.

Komunikace s backendovou službou pak probíhá pomocí příslušné API. Důležitá je možnost serializovat objekty modelů do formátu JSON. K tomu slouží již použitý protokol `JSONSerializable` a jeho metody `encode()` a `decode()`.

5.2.2 Lekce a jejich správa

Nejzajímavější částí lekcí je přehrávání videa. Stahování a nahrávání se principiálně velmi podobá ukázce nahrání dodatečných dat o uživateli v 5.2.1 jen s tím rozdílem, že média se nahrávají do *Firebase Cloud*. Každé médium je poté určeno svým URL. Náznak implementace nahrávání videí ukazuje následující zdrojový kód (pozn. nutnost `import FirebaseFirestore`).

```
private func uploadVideo() {
    // initialize data service
    let data = Data()
    // initialize Firebase Storage
    let storage = Storage.storage()
    // create a storage reference
    let storageRef = storage.reference()
    // create a media reference
    let mediaRef = storageRef.child("video_name.mp4")
    // safely create a URL to video that's gonna be uploaded
    guard let localURL = URL(string: LOCAL_FILE_PATH) else {
        // react if error occured
    }

    // do the upload itself
    let upload = mediaRef.putFile(from.    : localURL,
                                  metadata : nil))
    { metadata, error in
        // react to upload errors
        /* metadata contains additional
           information about the uploaded file */
    }
}
```

V následujícím zdrojovém kódu, který přibližuje implementaci přehrávání videa, se používá framework `AVKit`, který nabízí mnoho užitečných funkcí, usnadňujících práci s videem. Tento framework je součástí *Swift* SDK.

Metoda `playVideo()` nastiňuje implementaci stažení videa z dané URL a jeho spuštění za použití `AVKit`.

```
private func playVideo(FIREBASE_VIDEO_URL : String) {

    // safely prepare videoURL
    guard let videoURL = URL(string : FIREBASE_VIDEO_URL)
    else {
        // react if video url was not correct
    }

    // initialize a video player with the URL
    let avPlayer = AVPlayer(url : videoURL)

    // set up a player viewController and it's view
    let playerVC: AVPlayerViewController = {

        /* init AVPlayerVC and assign
        coresponding AVPlayer */
        let playerVC    = AVPlayerViewController()
        playerVC.player = avPlayer

        // set up the video view frame size
        playerVC.view.frame.size.height = set_height
        playerVC.view.frame.size.width  = set_width

        return playerVC
    }()

    /* now add subview to the current view to present
       the video accordingly (correct position, etc.) */

    // "self.view.addSubview(playerVC.view)"
}
```

5.2.3 Ladička

Jako poslední ukázka implementace přichází na řadu ladička. Ta je implementována za pomoci frameworku `AudioKit`, který je rovněž součástí *Swift* SDK. Pro celý proces vývoje viz sekci 5.2.1, zde se nachází opět pouze vybrané části zdrojových kódů.

Logika ladění není vůbec tak složitá, jak by se mohlo na první pohled zdát. Díky již zmíněnému frameworku `AudioKit`, lze nad snímaným zvukem provádět mnoho operací, které jsou pro správnou funkci ladičky potřebné.

Hlavní myšlenka spočívá v identifikaci frekvence snímaného tónu, jeho transformace do nějaké zvolené tóniny (oktávy) a následně porovnání s referenčním tónem. Podle rozdílu frekvencí lze poté určit jak by měla být struna kytary natahována nebo povolována k dosažení naladění.

Funkcionalita ladičky je rozdělena do tří modulů – Tone (model reprezentující daný tón a operace, které nad ním lze provádět), Reference (modul, který obsahuje logiku hledání referenčního tónu podle frekvence zvuku snímaného mikrofonem) a modul Tuner, který zapouzdřuje zmíněné dva moduly a obsahuje celkovou logiku ladičky. Spojení celého procesu do jedné zjednodušené metody může vypadat jako v nadcházející ukázce. Tato metoda musí pak být obalena jinou metodou, která ji bude periodicky spouštět po určitém časovém intervalu, např. 0,25 sekund.

```
import AudioKit

public func doTuning -> Double {

    let microphone. : AKMicrophone
    let analyzer    : AKAudioAnalyzer
    let reference   : Reference

    // start listening
    analyzer.play()
    microphone.play()

    // retrieve frequency from the analyzer as a Double
    let frequency = analyzer.trackedFrequency

    // find closest reference tone frequency
    let closestRef = reference.closestTone(frequency)

    // return the difference to the controller
    return (freqeuncy - closestRef)

    /* there, in the controller, it should be acted
    accordingly - let the user know what to do
    by changing the UI */
}
```

Tímto byl vysvětlen postup při implementaci ladičky. Na závěr je důležité říci, že tato ladička byla inspirována projektem citovaném jako zdroj [49].

5.3 Shrnutí implementace

V této kapitole došlo k popsání způsobu, jakým byla vyvíjena zadaná aplikace. Tento vývoj proběhl na základě stanovených funkčních a nefunkčních požadavků a také podle návrhu a technologií z předchozích kapitol. Ke každé důležité části byl vybrán příklad implementace a náležitě okomentován. Více informací ohledně výsledků implementace je rozvinuto v kapitole Výsledky a další možný rozvoj (7).

Splněním implementace, a to jak již bylo zmíněno podle klasického vývojového cyklu v softwarovém inženýrství, byla splněna implementační část zadání této práce.

Kapitola 6

Testování

Tato kapitola se zabývá testováním a to jak v průběhu samotného vývoje, tak po dokončení implementace všech potřebných požadavků. Testování je důležitou součástí softwarového vývojového cyklu. Pro účely této práce proběhlo unit testování při psaní zdrojových kódů a poté uživatelské testování.

Oba tyto druhy testování jsou popsány v následujících sekcích, které vždy začínají obecnými informacemi o dané metodě a poté ukazují konkrétní příklady (např. pomocí zdrojového kódu).

6.1 Unit testování

Unit testování je základním testovacím nástrojem při vývoji software. Toto testování spočívá v kontrolování krátkých celků kódu (*units*), obvykle jedné metody. Testy jsou psány softwarovými vývojáři a plní funkci validace správnosti chování napsané metody. Standardně bývají automatizované např. pomocí GitLab[4] CI. [50]

6.1.1 Vlastnosti

Výhoda unit testování spočívá v častém podchycení chyb brzy při vývoji, což poskytuje dostatek času na nápravu. Díky izolaci jednotlivých částí kódu při testování je tento způsob velmi spolehlivý. Navíc unit testy pro různé metody jsou často téměř stejné, tudíž lze kopírovat podobný kód a ušetřit si tak práci. [50]

Unit testy zachycují pouze chyby malých částí zdrojového kód, avšak nedokáží podchytit problémy systému jako takového, k tomu je pak nutné použít robustnější typy testování, například integrační nebo systémové testy. [50]

[4] populární systém pro správu verzí

Každý unit test se skládá ze tří fází. Těmito fázemi jsou:

- **Prepare** – Fáze, kdy se připraví objekt a data pro testovanou metodu.
- **Act** – Spuštění metody s danými parametry.
- **Assert** – Porovnání očekávaných (*expected*) a skutečných (*actual*) výsledků. [50]

Zároveň existuje několik metodik, které říkají, co by měl správný unit test splňovat. Jednou takovou metodikou je A TRIP, podle které by unit testy měly být:

- **Automatic** – Testy by měly být samy spuštěny při každém sestavení aplikace a zároveň by mělo dojít k automatickém zkontrolování výsledku.
- **Through** – Pokrytí kódu testy by mělo být dostatečné – je důležité myslet na všechny případy.
- **Repeatable** – Unit testy by měly při každém spuštění (pokud nikdo nezmění kód testované metody) dávat stejný výsledek. Jejich výstup nesmí záviset na vstupních parametrech testovaného celku.
- **Independent** – Jak již bylo řečeno, unit test by měl testovat pouze jednu malou samostatnou část implementace. Testy by se navíc neměly překrývat nebo na sobě záviset.
- **Professional** – I v kódu samotných testů by měl být dodržen pořádek. Testovacího kódu ve výsledku může být stejné množství jako toho produkčního. [51, str. 77–83]

Tím byly shrnuty všechny vlastnosti unit testů – jejich výhody a nevýhody, fáze průběhu testu a poté byla vybrána metoda A TRIP, která popisuje vlastnosti a obecně prospěšné zvyky, které by měly být dodrženy při vytváření unit testů.

6.1.2 Ukázka

Následuje ukázka jednoho unit testu vyvíjeného software. Jedná se o test modelu `UserModel` a jeho metody `encode()`. Ukázka zahrnuje celou testovací třídu. K unit testování *Swift* kódu slouží *framework* `XCTest`.

Tato ukázka uzavírá sekci unit testování. Vyvíjená aplikace byla po celou dobu implementace dostatečně pokryta unit testy a to v souladu s již zmíněným A TRIP. Tím došlo k částečnému splnění testování, které je součástí zadání této práce.

```
import XCTest
@testable import Guitario

class GuitarioTests: XCTestCase {
    var sut : UserModel!     // sut is 'System Under Test'

    override func setUpWithError() throws {
        // This method is called before the invocation
        try super.setUpWithError()

        sut = UserModel(
            firstName : "John",
            lastName  : "Wick",
            UID       : "fds432sfx3"
        )
    }

    override func tearDownWithError() throws {
        // This method is called after the invocation
        sut = nil

        try super.tearDownWithError()
    }

    func testUserModelEncode() {
        // given
        let dictionaryMock : [String: AnyObject] =
            ["firstName"   : "John"       as AnyObject,
             "lastName"    : "Wick"       as AnyObject,
             "UID"         : "fds432sfx3" as AnyObject]

        // when
        let dictionary = sut.encode

        // then
        XCTAssertEqual(
            dictionary     as! [String: String],
            dictionaryMock as! [String: String],
            "The encode method failed for model UserModel."
        )
    }
}
```

6.2 Uživatelské testování

Dalším typem testování, který byl použit při vývoji této aplikace je uživatelské testování. Jedná se o typ testování, který se provádí až po implementaci a je při něm testována celková funkčnost software. [52]

Reální lidé zkoušejí používat vyvinutý produkt a dělat s ním úkony popsané v testovacích scénářích. Poté, na základě zpětné vazby, může být produkt upraven a tím odstraněny nalezené nedostatky. [52]

Výhodou uživatelského testování je odzkoušení produktu před jeho oficiálním nasazením do produkce. Lze tak odchytat chyby, které by jiný typ testování neodhalil – to i z důvodu, že testeři jsou často lidé bez zkušeností v oblasti informačních technologií a mohou si tak všimnout chyb, které vývojáři přehlídli. Navíc úkony testovacího scénáře mohou být dostatečně komplexní k odhalení netypických chyb. [52]

Nevýhodou pak mohou být zvýšené náklady na provedení testování – poskytnutí vybavení, zaplacení testerů, apod. Dále skupina testerů bude pravděpodobně celkem malá, tudíž zrovna mohou všichni bez problému zvládnout to, co bude obtížné pro běžného budoucího uživatele. Navíc následná analýza zpětné vazby po dokončení testování může být náročná. [52]

6.2.1 Testovací scénáře

Jak již bylo zmíněno, testovací scénář obsahuje instrukce, které by měl tester učinit, aby dosáhl vykonání zadaného úkonu (např. registrace) a tím tuto funkcionalitu otestovat. Tyto scénáře často vychází z případů užití.

Před začátkem daného testu by měly být testerovi poskytnuty tzv. prerekvizity, to znamená počáteční podmínky ke splnění testovacího scénáře – například, pokud půjde pouze o testování přihlášení bez registrace, tester musí mít k dispozici přihlašovací jméno a heslo. Dále proběhne samotný test podle přesných instrukcí. Nakonec je důležité, aby tester poskytl vypovídající zpětnou vazbu, kterou lze později analyzovat.

Testování vyvíjené aplikace se zúčastnili čtyři testeři, známí autora této práce. Jednalo se o dva vysokoškolské studenty informačních technologií, jednu ženu ve věku 34 let a muže ve věku 61 let. Přesný popis některých testovacích scénářů se nachází v příloze A.

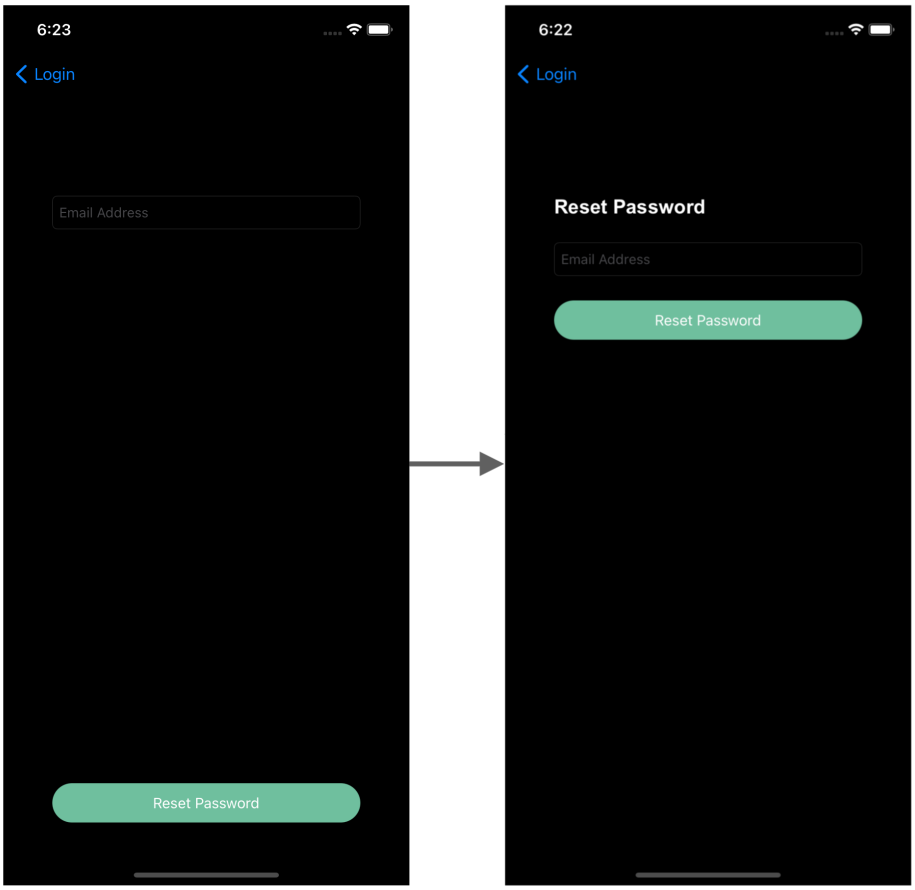
6.2.2 Úpravy po dokončení testování

Po dokončení uživatelského testování byla provedena analýza zpětné vazby a na jejím základě pak úpravy aplikace. Jednalo se pouze o úpravy detailů uživatelského rozhraní, které nebylo vždy ideální.

Jako příklad je uvedeno změna polohy tlačítek v celé autentizační sekci aplikace. Tato tlačítka byla příliš nízko, tedy daleko od vyplňovaných polí např. při registraci, což značně ztěžovalo ovládání jedním prstem. Navíc se

tato tlačítka neposouvala při vysunutí klávesnice, takže pro jejich stisknutí musel uživatel nejprve klávesnici skrýt.

Tento problém byl vyřešen přesunutím tlačítka bezprostředně pod textová pole, čímž se vyřešily oba dva zmíněné problémy. Ukázka této úpravy následuje v obrázku 6.1.



Obrázek 6.1: Ukázka změny UI po uživatelském testování

6.3 Shrnutí testování

V této kapitole byly popsány způsoby, kterými byla testována vyvíjená aplikace. Testování bylo provedeno tak, aby odpovídalo běžným postupům softwarového inženýrství a zadání této bakalářské práce.

Kapitola 7

Výsledky a další možný rozvoj

Tato krátká a zároveň poslední kapitola prezentuje výsledky této bakalářské práce a nastiňuje další možný rozvoj vyvíjené aplikace. Jedná se o poslední zatím nesplněný bod zadání.

7.1 Výsledky práce

Hlavní náplní této sekce je zhodnocení úspěchu při analýze, návrhu a výběru technologií, což je možné až nyní – po implementační části a testování. Následuje stručný popis splnění funkčních a nefunkčních požadavků.

7.1.1 Analýza, návrh a výběr technologií

Analytická část této práce poskytla dobrou představu o tom, jakým způsobem je vhodné vyučovat hru na kytaru a také jak vypadají podobné aplikace, což umožnilo určitou inspiraci. Konkurenční aplikace zároveň poukázaly na některé své vlastnosti a nedostatky, které jejich uživatelé nemají příliš v oblibě, a proto bylo lepší se jim vyvarovat. Ze všech těchto poznatků v kombinaci se zadáním práce byly sestaveny funkční a nefunkční požadavky.

Následně vybraný návrhový vzor se ukázal jako velmi vhodný pro účely tohoto vývoje. MVC poskytlo dostatečné rozdělení do komponent, proto je výsledný kód dostatečně přehledný a poskytuje prostor ke změnám a rozšíření. Zároveň díky Apple návodům, kde se také téměř vždy používá MVC, jednoduchosti a oblíbenosti tohoto návrhového vzoru nenastaly problémy při hledání dokumentace a názorných příkladů.

U výběru technologií byla situace o něco jednodušší – jelikož byla aplikace vyvíjena na operačním systému macOS a pro platformu iOS, bylo po analýze vybráno IDE *Xcode* a to se ukázalo jako správná volba. *Storyboards*, které jsou součástí *Xcode* a umožňují rychlejší tvorbu uživatelského rozhraní, přidali na rychlosti vývoje. Jediným nedostatkem *storyboards* pak byly občasné delší doby načítání.

Celkově se tedy dá říci, že mezi vybranými technologiemi nebyl žádný nástroj, který by ztěžoval proces vývje, a proto lze považovat výběr za úspěšný.

7.1.2 Splnění požadavků

V této práce se podařilo dodržet všechny nefunkční požadavky. Aplikace funguje na platformě iOS, díky návrhovému vzoru MVC je z hlediska implementace relativně jednoduchá a zároveň i jednoduše rozšiřitelná (za předpokladu použití stejných technologií při případném budoucím vývoji).

Stanovené funkční požadavky, které byly ohodnoceny prioritou *must have* byly splněny všechny, stejně tak i požadavky priority *should have*. Co se týče funkčních požadavků typu *could have*, tak u těch došlo k částečnému splnění. Priorita *won't have* nebyla implementována v souladu s časovým rozvržením projektu.

7.1.3 Shrnutí výsledků

V této práci se úspěšně podařilo splnit všechny body zadání. Od analýzy k sestavení požadavků, poté vytvoření návrhu a implementaci, při které se nakonec podařilo splnit všechny požadavky náležitých priorit (*must have* a *should have*) a i některé s nižší prioritou. Závěrečné testování odhalilo několik nedostatků, na které bylo možné reagovat opravou.

Výsledkem je tedy otestovaná funkční mobilní aplikace pro platformu iOS, která byla vyvinuta podle standardního postupu softwarového inženýrství a rovněž za účelem splnění všech bodů zadání, což se podařilo.

7.2 Další možný rozvoj

Tato sekce nabízí krátký pohled na možný budoucí vývoj vytvořené mobilní aplikace. Konkrétní nápady jsou popsány v několika následujících sekcích.

7.2.1 Z výsledků testování

Díky uživatelskému testování, při kterém se s aplikací dostalo do styku několik lidí různých zaměření a rozdílných věkových skupin, bylo shromážděno značné množství zpětné vazby, jejíž součástí byly i návrhy na některé nové funkce.

Jednalo se především o změny v uživatelském rozhraní, které by mohly vytvořit ještě o něco příjemnější pocit z užívání této aplikace. Konkrétně se jednalo například o přidání spojitého zobrazení aktuální frekvence u ladičky, namísto jednoho zobrazovaného tónu. Tím by se proces ladění zpřesnil a zároveň by se tak daly ladit tóny, které ladička momentálně nezobrazuje.

Jiným návrhem pak bylo umožnění nahrávání obsahu uživateli – tedy aby všichni uživatelé mohli vytvářet lekce. Tuto funkci by bylo možné implementovat jen za předpokladu validace správnosti všeho nahraného obsahu, což by

mohlo být velmi náročné. V opačném případě, jak bylo potvrzeno v analýze Ultimate Guitar, sekce 2.2.2, není vhodné tuto funkcionalitu implementovat.

Návrhy vyplývající z uživatelského testování jsou ve výsledku velmi užitečné – dávají přibližný obraz o tom, co si běžný uživatel zhotovené aplikace přeje vidět jako další implementovanou funkcionalitu. Některé z těchto nápadů však nemusí být z různých důvodů nejlepší.

7.2.2 Zbylé funkční požadavky

I přes implementování většiny funkčních požadavků zbývají ty, které jsou priority *won't have* a také jeden požadavek typu *could have*. Jelikož se jedná o původně stanovené funkční požadavky, bude pravděpodobně vhodné je implementovat jako jedny z prvních nadcházejících funkcionalit aplikace.

7.2.3 Shrnutí budoucího rozvoje

Díky kvalitnímu návrhu, na kterém je aplikace postavena a dodržení požadavku na rozšiřitelnost, je aplikace dobře připravena k dalšímu rozvoji. Zároveň existuje dostatečné množství nápadů, ať už ze strany uživatelů, kteří aplikaci testovali, tak od samotného autora práce. Je tedy na místě analyzovat, kterou další funkcionalitu implementovat jako první a poté je možné se ihned opět pustit do vývoje.

Závěr

Cílem této bakalářské práce bylo vytvořit iOS mobilní aplikaci určenou k výuce hry na kytaru podle standardního postupu tvorby software v softwarovém inženýrství. To znamená provést analýzu, vytvořit návrh, implementovat a testovat samotnou aplikaci a nakonec zhodnotit výsledky a navrhnout další možný rozvoj.

Jako první krok bylo nutné analyzovat klasický přístup k hraní na kytaru a zároveň analyzovat konkurenční mobilní aplikace. Na základě poznatků z analýzy bylo možné sestavit funkční a nefunkční požadavky na aplikaci, které byly prezentovány pomocí případů užití.

V další kapitole byla vybrána vhodná architektura, která vyhověla stanoveným požadavkům. Zároveň došlo k sestrojení databázového schématu a navržení uživatelského rozhraní.

Ve čtvrté kapitole došlo k výběru technologií a to včetně důkladného zvážení nejpopulárnějších alternativních řešení. Bylo rozhodnuto vyvíjet aplikaci v jazyce *Swift* a IDE *Xcode* a vytvářet uživatelské rozhraní pomocí *storyboards*. Zároveň bylo rozhodnuto využívat backendovou službu *Firebase*.

V následujících dvou kapitolách přišla na řadu samotná implementace a testování. Při implementaci byly splněny všechny požadavky odpovídající priority. Zdrojové kódy zůstaly v průběhu vývoje pokryty unit testy. Následné uživatelské testování odhalilo menší nedostatky, které byly brzy opraveny.

V závěrečné kapitole byly zhodnoceny výsledky celé práce a poté byl navrhnut další možný rozvoj vyvíjené aplikace. Došlo ke splnění všech hlavních a dílčích cílů, tudíž i zadání této bakalářské práce.

Literatura

[1] TrueFire Guitar: TrueFire Guitar Lessons [online]. 2021, [cit. 2021-05-05]. Dostupné z: https://apps.apple.com/us/app/truefire-guitar-lessons/id690143001

[2] Wang, M.: Ultimate Guitar: iOS Mobile App [online]. 2021, [cit. 2021-05-05]. Dostupné z: https://dribbble.com/shots/12045306-Ultimate-Guitar-iOS-Mobile-App

[3] Yousician Ltd: Yousician - Your Music Teacher [online]. 2021, [cit. 2021-05-05]. Dostupné z: https://apps.apple.com/us/app/yousician-your-music-teacher/id959883039

[4] Mlejnek, J.: Analýza a sběr požadavků [online]. 2021, [cit. 2021-05-06]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/388197/mod_resource/content/4/03.prednaska.pdf

[5] Creately: Use Case Diagram Relationships Explained with Examples [online]. 2020, [cit. 2021-05-10]. Dostupné z: https://creately.com/blog/diagrams/use-case-diagram-relationships/

[6] Capka, D.: Lesson 4 - UML - Domain Model [online]. 2021, [cit. 2021-05-07]. Dostupné z: https://www.ictdemy.com/software-design/uml/uml-domain-model

[7] Mozilla Developer: MVC [online]. 2021, [cit. 2021-05-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/MVC

[8] Gossman, J.: Advantages and disadvantages of M-V-VM [online]. 2006, [cit. 2021-05-07]. Dostupné z: https://docs.microsoft.com/en-us/archive/blogs/johngossman/advantages-and-disadvantages-of-m-v-vm

[9] Martin, R. C.: The Clean Architecture [online]. 2012, [cit. 2021-05-07]. Dostupné z: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[10] Lucidchart: Entity-Relationship Diagram Symbols and Notation [online]. 2021, [cit. 2021-05-08]. Dostupné z: https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning

[11] Karczewski, D.: Swift vs Objective-C: Which Should You Pick For Your Next iOS Mobile App? [online]. 2020, [cit. 2021-05-09]. Dostupné z: https://steelkiwi.medium.com/swiftui-vs-uikit-benefits-and-drawbacks-6a540cced684

[12] Fajmon, D.: Postup výuky hry na kytaru [online]. 2021, [cit. 2021-05-10]. Dostupné z: http://www.lekcekytary.cz/postup-vyuky.php

[13] Thorpe, D.: 14 Scientifically Proven Ways to Learn and master the guitar faster and quicker than ever [online]. 2021, [cit. 2021-05-05]. Dostupné z: https://guitardomination.net/14-scientifically-proven-ways-to-learn-and-master-the-guitar-faster-and-quicker-than-ever/

[14] Duncan, L.: What Is The Best Way To Teach Yourself Guitar? [online]. 2021, [cit. 2021-05-05]. Dostupné z: https://www.musicindustryhowto.com/what-is-the-best-way-to-teach-yourself-guitar/

[15] Trent, D.: TrueFire Review (Sep 2021): Is TrueFire Worth the Money? [online]. 2021, [cit. 2021-05-05]. Dostupné z: https://guitarspace.org/tips/is-truefire-the-best-guitar-instruction-website/

[16] Ultimate Guitar: Ultimate Guitar: Chords & Tabs [online]. 2021, [cit. 2021-05-05]. Dostupné z: https://apps.apple.com/us/app/ultimate-guitar-chords-tabs/id357828853

[17] Cooper, A.: The 13 Best Guitar Apps That You Will Actually USE In 2020 [online]. 2021, [cit. 2021-05-05]. Dostupné z: https://guitarsongsmasters.com/best-guitar-apps/

[18] Macháčková, E.; Macháček, Z.: Metoda MoSCoW a model KANO [online]. 2016, [cit. 2021-05-10]. Dostupné z: https://www.systemonline.cz/rizeni-projektu/metoda-moscow-a-model-kano.htm

[19] Tutorial Point: Design Patterns - MVC Pattern [online]. 2021, [cit. 2021-05-06]. Dostupné z: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

[20] Voorhees, D. P.: *Guide to Efficient Software Design*. Springer Nature Switzerland AG, ISBN 978-3-030-28501-2.

[21] Jacobs, B.: What Is Wrong With Model-View-Controller [online]. 2020, [cit. 2021-05-06]. Dostupné z: https://cocoacasts.com/what-is-wrong-with-model-view-controller

[22] Martin, R. C.: *Clean Architecture*. Pearson Education, Inc., ISBN 978-0-13-449416-6.

[23] Apple Inc.: Model-View-Controller [online]. 2018, [cit. 2021-05-08]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[24] Blaha, M.: *UML Database Modeling Workbook*. Technics Publications, LLC, ISBN 978-1-9355045-1-1.

[25] Apple Inc.: UI Design Dos and Don'ts [online]. 2021, [cit. 2021-05-08]. Dostupné z: https://developer.apple.com/design/tips/

[26] Fueled Design Team: Fueled's Ultimate Guide to Creating Mobile App Wireframes [online]. 2018, [cit. 2021-05-08]. Dostupné z: https://fueled.com/blog/creating-mobile-app-wireframes/

[27] Apple Inc.: Tab Bar [online]. 2020, [cit. 2021-05-08]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/ios/bars/tab-bars/

[28] Apple Inc.: Settings [online]. 2020, [cit. 2021-05-08]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/settings/

[29] Apple Inc.: Swift [online]. 2021, [cit. 2021-05-09]. Dostupné z: https://developer.apple.com/swift/

[30] Apple Inc.: Xcode IDE [online]. 2021, [cit. 2021-05-09]. Dostupné z: https://developer.apple.com/xcode/features/

[31] CompareCamp: Xcode IDE Review [online]. 2020, [cit. 2021-05-09]. Dostupné z: https://comparecamp.com/xcode-ide-review-pricing-pros-cons-features/

[32] Ather, S.: Xcode or AppCode: Which is more suitable for iOS app development [online]. 2021, [cit. 2021-05-09]. Dostupné z: https://www.iphoneglance.com/2020/01/28/xcode-or-appcode-for-ios-app-development/

[33] JetBrains: AppCode [online]. 2021, [cit. 2021-05-09]. Dostupné z: https://www.jetbrains.com/objc/

[34] Flutter: Flutter for iOS developers [online]. 2021, [cit. 2021-05-09]. Dostupné z: https://flutter.dev/docs/get-started/flutter-for/ios-devs

[35] Apple Inc.: UIKit [online]. 2021, [cit. 2021-05-09]. Dostupné z: https://developer.apple.com/documentation/uikit

[36] SwiftDev365: Storyboards vs Programmatic iOS Development? [online]. 2019, [cit. 2021-05-09]. Dostupné z: https://medium.com/@matlyles/storyboards-vs-programmatic-ios-development-40c5b5907f85

[37] Apple Inc.: Designing with Storyboards [online]. 2021, [cit. 2021-05-09]. Dostupné z: https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/DesigningwithStoryboards.html

[38] Apple Inc.: Storyboard [online]. 2018, [cit. 2021-05-09]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

[39] Ramnath, R.; Loffing, C.: *Beggining iOS Programming For Dummies*. John Wiley & Sons, Inc., ISBN 978-1-118-79931-4.

[40] Apple Inc.: SwiftUI [online]. 2021, [cit. 2021-05-09]. Dostupné z: https://developer.apple.com/xcode/swiftui/

[41] SteelKiwi Inc.: SwiftUI vs UIKit: Benefits and Drawbacks [online]. 2020, [cit. 2021-05-09]. Dostupné z: https://steelkiwi.medium.com/swiftui-vs-uikit-benefits-and-drawbacks-6a540cced684

[42] Google Inc.: Learn about Build products [online]. 2021, [cit. 2021-05-10]. Dostupné z: https://firebase.google.com/docs/build

[43] Google Inc.: Firebase Authentication [online]. 2021, [cit. 2021-05-10]. Dostupné z: https://firebase.google.com/docs/auth

[44] Google Inc.: Cloud Firestore [online]. 2021, [cit. 2021-05-10]. Dostupné z: https://firebase.google.com/docs/firestore

[45] Google Inc.: Firebase Realtime Database [online]. 2021, [cit. 2021-05-10]. Dostupné z: https://firebase.google.com/docs/database

[46] Google Inc.: Add Firebase to your iOS project [online]. 2021, [cit. 2021-05-11]. Dostupné z: https://firebase.google.com/docs/ios/setup

[47] Apple Inc.: Preparing Your App for Distribution [online]. 2021, [cit. 2021-05-11]. Dostupné z: https://developer.apple.com/documentation/xcode/preparing-your-app-for-distribution

[48] Durán, E.; Pelosin, F.: What is CocoaPods [online]. 2021, [cit. 2021-05-11]. Dostupné z: https://guides.cocoapods.org/using/getting-started.html

[49] Tim van Elsloo: Guitar Tuner for iOS [online]. 2017, [cit. 2021-05-12]. Dostupné z: https://github.com/elslooo/guitar-tuner

[50] Tim van Elsloo: Unit test basics [online]. 2019, [cit. 2021-05-12]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics

[51] Hunt, A.; Thomas, D.: *Pragmatic Unit Testing*. Pragmatic Programmers, LLC, ISBN 0-9745140-1-2.

[52] Hotjar: A beginner's guide to user usability testing [online]. 2021, [cit. 2021-05-13]. Dostupné z: https://www.hotjar.com/usability-testing/

Příloha **A**

Ukázky scénářů uživatelského testování

A.1 Registrace

Prerekvizity – Uživatel musí mít možnost přístupu k nějakému emailu, na který se bude registrovat.

Testovací scénář

1. **Zapnutí aplikace** – Na dlaždicové ploše najděte ikonku aplikace a spusťte ji. Poté počkejte na zobrazení úvodní animace a zobrazení výchozí obrazovky.
2. **Přechod na registraci** – Nyní na ve spodní části obrazovky klikněte na tlačítko „Sign Up“. Budete přesměrování na obrazovku registrace.
3. **Vyplnění osobních údajů** – Vyplňte své osobní údaje do odpovídajících textových políček, přesněji jméno a příjmení, email, poté si zvolte heslo a v dalším poli jej potvrďte. V případě zadání chybného údaje budete notifikováni. V takovém případě se pokuste o vyplnění znovu.
4. **Potvrzení emailu** – V případě úspěšného vyplnění požadovaných údajů otevřete svoji emailovou schránku, ve které bude potvrzovací email. Otevřete odkaz, který je obsahem tohoto emailu.

Tímto je proces registrace hotový. Pokud jste splnil/a všechny kroky scénáře, vytvořili jste si svůj uživatelský účet a test úspěšně skončil.

A.2 Nahrání lekce

Prerekvizity – Uživatel musí znát přístupové údaje k uživatelskému účtu, který je nastaven jako administrátorský.

Testovací scénář

1. **Zapnutí aplikace** – Zapněte aplikaci a vyčkejte na zobrazení výchozí obrazovky. Ve spodní části této obrazovky klikněte na tlačítko „Login“.
2. **Přihlášení** – Na přihlašovací obrazovce vyplňte požadované údaje, tedy email a heslo. Poté se přihlaste pomocí tlačítka „Login“ pod textovými poli. Po úspěšném přihlášení byste se měli nacházet na hlavní stránce aplikace.
3. **Vytvoření lekce** – Ve spodní liště aplikace klikněte na ikonku nastavení (nejvíce napravo). V seznamu nastavení pod sekcí „Administrator“ vyberte položku „Upload Tutorial“. Budete přesměrování na obrazovku pro vytváření lekcí.
4. **Nahrání médií** – V horní části této obrazovky klikněte na tlačítko „Upload Video“, případně „Upload Picture“. Vždy vyberte odpovídající média (video nebo obrázek) a potvrďte svůj výběr. Poté klikněte na tlačítko „Upload Thumbnail“ a vyberte obrázek náhledu lekce.
5. **Vložení textu a zveřejnění videa** – Do (směrem dolů) následujícího textového pole vložte název lekce. Poté do textového pole o jedno níže vložte text lekce. Po dokončení všech předešlých kroků klikněte na tlačítko „Upload“ ve spodní části obrazovky. Budete přesměrování na domovskou obrazovku aplikace.

V případě, že jste se dostali až na konec pátého kroku scénáře, jste úspěšně zvládli nahrát novou lekce a tím i splnit tento test.

A.3 Naladění kytary

Prerekvizity – Uživatel potřebuje pro zvládnutí tohoto testu uživatelský účet zaregistrovaný pomocí emailu a hesla. Dále je nutné mít k dispozici kytaru, která bude v průběhu testu naladěna.

Testovací scénář

1. **Zapnutí aplikace** – Zapněte aplikaci a vyčkejte na zobrazení výchozí obrazovky. Ve spodní části této obrazovky klikněte na tlačítko „Login“.
2. **Přihlášení** – Na přihlašovací obrazovce vyplňte požadované údaje, tedy email a heslo. Poté se přihlaste pomocí tlačítka „Login“ pod textovými poli. Po úspěšném přihlášení byste se měli nacházet na hlavní stránce aplikace.
3. **Otevření ladičky** – Na spodní navigační liště najděte záložku pro ladičku. Tato záložka je označena jako „Tuner“ a její ikonka vypadá jako rychloměr. Klikněte na tuto ikonu.
4. **Ladění kytary** – Nyní přichází část ladění kytary. Zahrajte prázdnou libovolnou strunu. Na ladičce uvidíte referenční tón, který je nejblíže tónu, který hrajete. Pokud se vedle názvu tónu zobrazí šipka nahoru, musíte utahovat strunu. Pokud dolu, je potřeba strunu o něco povolit. Pokud se namísto šipky zobrazí zelený obrázek symbolizující úspěch, daná struna je naladěna. Opakujte tento proces pro všechny struny kytary.

Pokud se vám podařilo dostat až ke kroku čtyři a opakovat proces ladění pro všechny struny, vaše kytara je naladěna a test byl úspěšně ukončen.

A.4 Odhlášení

Prerekvizity – Uživatel musí být na začátku procesu přihlášen v nějakém uživatelském účtě.

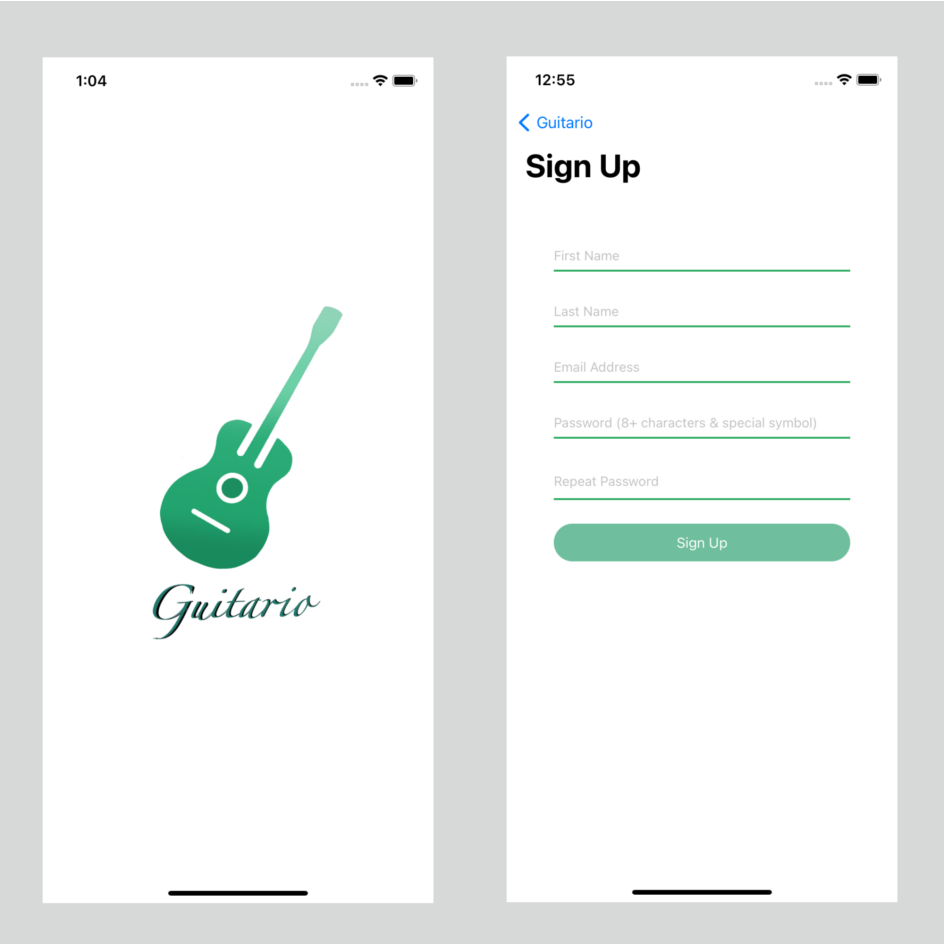
Testovací scénář

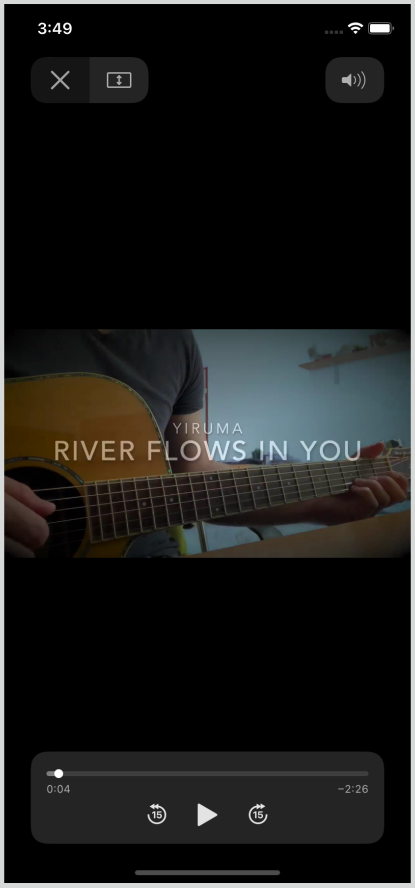
1. **Otevření nastavení** – Na spodní navigační liště najděte záložku pro nastavení. Jedná se o ikonku, která vypadá jako ozubené kolo a je nejvíce vpravo.
2. **Odhlášení** – Po zobrazení nastavení uvidíte seznam všech možností rozdělený do skupin. Najděte položku „Log Out“ ve skupině „Account“ a klikněte na ni. Tato skupina se nachází ve spodní části seznamu.
3. **Potvrzení odhlášení** – Po kliknutí na již zmíněnou položku seznamu se vám zobrazí upozornění o odhlášení, které nyní musíte potvrdit. Pod zobrazenou zprávou klikněte na tlačítko „Yes“, čímž dokončíte proces odhlášení. V opačném případě akci zrušíte.

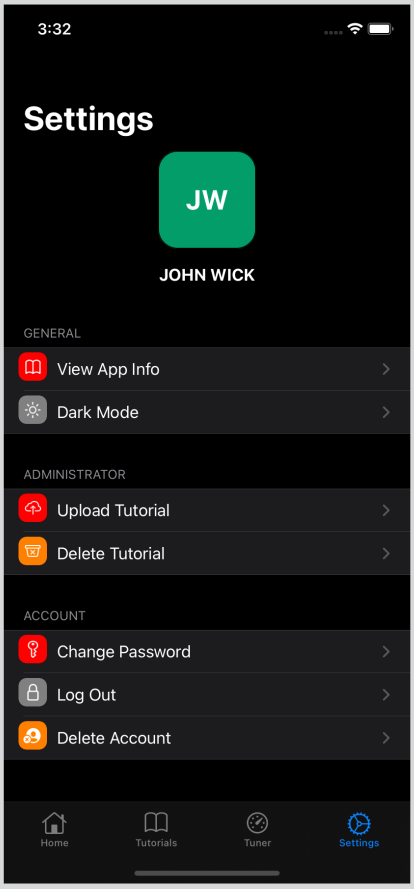
Pokud se vám podařilo projít celým procesem odhlášení, nacházíte se na výchozí obrazovce aplikace a jste odhlášení. Zároveň jste tím splnili tento test.

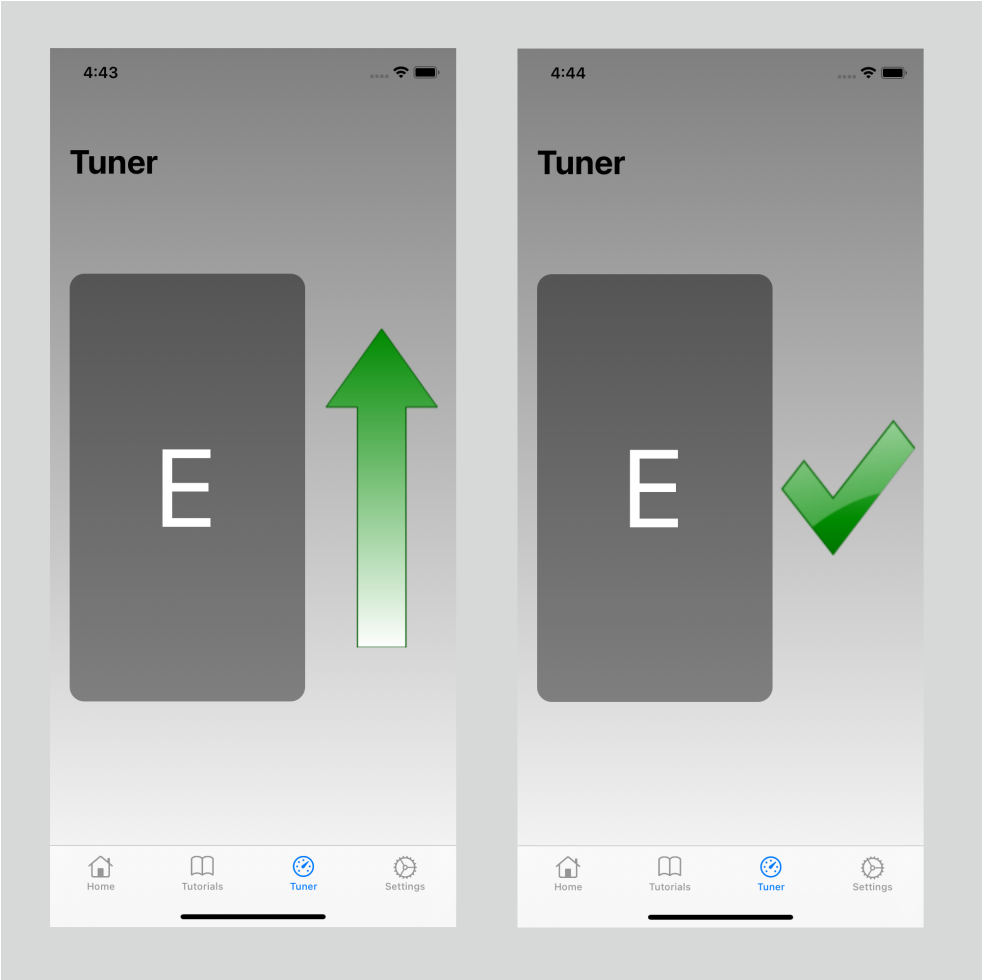
Příloha B

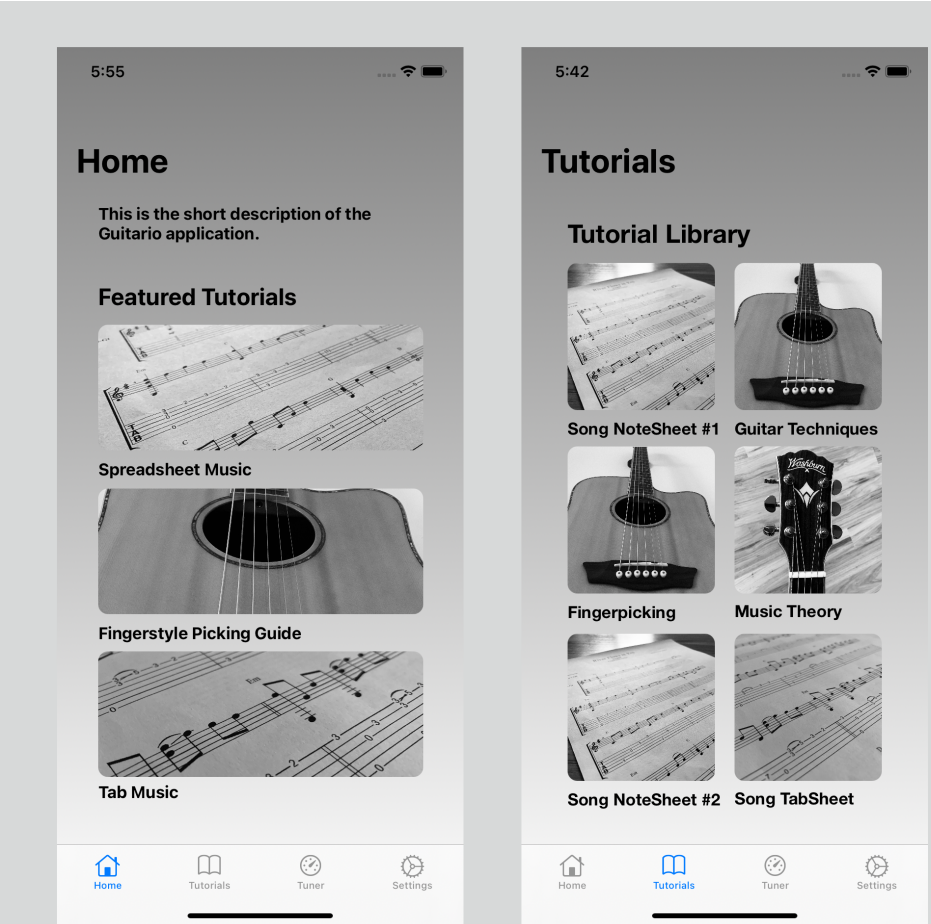
Vzhled vybraných obrazovek výsledné aplikace




3:49
YIRUMA
RIVER FLOWS IN YOU
0:04
-2:26


3:32
Settings
JW
JOHN WICK
GENERAL
View App Info
Dark Mode
ADMINISTRATOR
Upload Tutorial
Delete Tutorial
ACCOUNT
Change Password
Log Out
Delete Account
Home
Tutorials
Tuner
Settings


4:43
Tuner
E
Home
Tutorials
Tuner
Settings
4:44
Tuner
E
Home
Tutorials
Tuner
Settings


5:55
Home
This is the short description of the Guitario application.
Featured Tutorials
Spreadsheet Music
Fingerstyle Picking Guide
Tab Music
Home
Tutorials
Tuner
Settings
5:42
Tutorials
Tutorial Library
Song NoteSheet #1
Guitar Techniques
Fingerpicking
Music Theory
Song NoteSheet #2
Song TabSheet
Home
Tutorials
Tuner
Settings

Příloha C

Seznam použitých zkratek

API Application Programming Interface

A TRIP Automatic, Through, Repeatable, Independent, Professional

CI Continuous Integration

CRUD Create, Read, Update, Delete

ERM Enetity Relation Model

GUI Graphical User Interface

IB Interface Builder

IDE Integrated Development Environment

JSON JavaScript Object Notation

MoSCoW Must have, Should have, Could have, Won't have

MVC Model-View-Controller

MVVM Model-View-ViewModel

OS Operační systém

SDK Software Development Kit

UI User Interface

URL Uniform Resource Locator

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `examples` ..... ukázky výsledné aplikace
  - `screenshots` ..... obrazovky výsledného vzhledu aplikace
- `src` ..... zdrojové soubory
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF