Prototyp vzdělávací mobilní aplikace pro platformu iOS

Alexandr Čížek

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Alexandr Čížek**
Osobní číslo:	**A20301**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Prototyp vzdělávací mobilní aplikace pro platformu iOS**
Téma práce anglicky:	**Prototype Educational Mobile Application for the iOS Platform**

Zásady pro vypracování

1. Proveďte průzkum trhu v oblasti vzdělávacích mobilních aplikací pro iOS.
2. Proveďte literární rešerši a popište nástroje pro programování nativních aplikací na platformě iOS.
3. Definujte funkční a nefunkční požadavky pro prototyp aplikace.
4. Popište návrh uživatelského rozhraní prototypu.
5. Vytvořte prototyp aplikace za použití frameworku SwiftUI a popište důležité části implementace, datové modely a strukturu databáze.

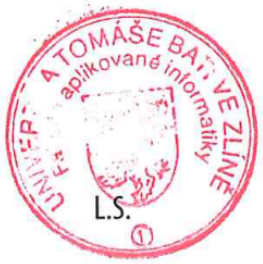
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. SMYTH, Neil. SwiftUI Essentials: Learn to Develop iOS Apps Using SwiftUI, Swift 5.5 and Xcode 13. IOS 15 Edition. Payload Media, 2022. ISBN 978-1951442439.
2. BERLIN, Josh a René CACHEAUX, FERGUSON, Darren, ed. Advanced iOS App Architecture: Real-World App Architecture in Swift. Fourth Edition. Razeware LLC, 2022. ISBN 978-1950325610.
3. The Swift Programming Language (Swift 5.7) [online]. Cupertino: Apple Inc., 2022 [cit. 2022-10-09]. Dostupné z: https://docs.swift.org/swift-book/
4. Firebase Documentation: Build Documentation [online]. San Francisco: Firebase, 2021 [cit. 2022-10-09]. Dostupné z: https://firebase.google.com/docs/build
5. ADAMS, Sean, Peter DAWSON, John FOSTER a Tony SEDDON. Graphic Design Rules: 365 Essential DOS and Don'ts. Revised edition with a new introduction by Sean Adams. New York: Princeton Architectural Press, 2020. ISBN 978--1616898762.
6. Xcode: Apple Developer Documentation [online]. Cupertino: Apple Inc., 2022 [cit. 2022-10-09]. Dostupné z: https://developer.apple.com/documentation/xcode
7. MISHRA, Abhishek. IOS Code Testing: Test-Driven Development and Behavior-Driven Development with Swift. New York: Apress, 2017. ISBN 978-1484226902.
8. WOOD, Brian. Adobe XD Classroom in a Book (2020 release). San Jose: Adobe Press, 2020. ISBN 978-0136583806.

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**



doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne
10. 5. 2023 Alexandr Čížek, v.r.

ABSTRAKT

Bakalářská práce se zabývá tvorbou prototypu mobilní aplikace pro vzdělávání. Cílem aplikace je uživatelům jednoduše a rychle zpřístupnit kvalitní informace o požadovaném tématu. Základní datovou strukturou v aplikaci je “List”, který obsahuje více typů výukových materiálů. Uživatelé mohou vytvořit vlastní List, nebo vyhledat a získat List od ověřených autorů. První část je zaměřena na průzkum trhu a studium konkurence v oblasti vzdělávacích mobilních aplikací. Průzkum obsahuje porovnání podobných platforem a jejich zhodnocení. Druhá část představuje koncept a filozofii projektu spolu s nejdůležitějšími funkcionalitami. Třetí část definuje funkcionální a nefunkcionální požadavky aplikace na základě předešlého průzkumu. Dále uvádí základní srovnání prototypu s konkurencí. Čtvrtá část obsahuje návrh vzhledu uživatelského rozhraní s použitím základních pravidel pro tvorbu grafického designu. Poslední část je zaměřena na implementaci aplikace v nativním frameworku SwiftUI. Dále předvede datové modely, strukturalizaci databáze a příklady testovaní iOS aplikace.

Klíčová slova: vzdělávací mobilní aplikace, výukové materiály, prototyp, uživatelské rozhraní, implementace aplikace, SwiftUI, databáze, iOS

ABSTRACT

The bachelor thesis deals with the creation of a prototype of a mobile application for education. The aim of the application is to provide users with easy and quick access to quality information on the desired topic. The basic data structure in the application is "List" which contains multiple types of learning materials. Users can create their own List or search and retrieve List from verified authors. The first part focuses on market research and studying competitors in the field of educational mobile applications. The survey includes a comparison of similar platforms and their evaluation. The second part presents the concept and philosophy of the project along with the most important functionalities. The third part defines the functional and non-functional requirements of the application based on the previous research. It also provides a basic comparison of the prototype with the competitors. The fourth part contains the design of the user interface using basic graphic design rules.

The last section focuses on the implementation of the application in the native SwiftUI framework. It also demonstrates data models, database structuring, and examples of iOS app testing.

Keywords: educational mobile app, educational materials, prototype, user interface, app implementation, SwiftUI, database, iOS

V této části bych rád poděkoval svým rodičům, kteří mě při celém studiu podporovali a pomohli mi se stylistickými a gramatickými úpravami při tvorbě bakalářské práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Práce obsahuje popis procesu návrhu a implementace prototypu mobilní aplikace pro platformu iOS s použitím nativního frameworku SwiftUI. Na začátku se zabývá průzkumem konkurenčních platforem z kategorie vzdělávacích aplikací, který obsahuje srovnání jednotlivých řešení. Na základě průzkumu jsou poté definovány požadavky pro prototyp. V teoretické části jsou popsány všechny technologické nástroje, zejména vybraný programovací jazyk Swift, framework SwiftUI, návrhový vzor MVVM a vývojové prostředí Xcode. Dále je zde zmíněna platforma Firebase a její služby, které jsou použity v aplikaci a aplikace Adobe Xd pro tvorbu návrhu uživatelského rozhraní. V závěru teoretická část obsahuje popis a způsob použití návrhových vzorů a technik při implementaci aplikace. Praktická část zachycuje proces tvorby návrhu uživatelského rozhraní a vytvoření prototypu. Popisuje hlavní strukturu projektu z hlediska rozřazení jednotlivých zdrojových souborů v souladu s návrhovým vzorem MVVM. Dále popisuje implementaci základních funkcionalit, například registraci a přihlašování uživatelů, vyhledávání obsahu, nebo generování a skenovaní QR kódů. Součástí praktické části je popis databázového systému Firebase Firestore a datových modelů, které aplikace používá.

I. TEORETICKÁ ČÁST

1 POPIS TECHNOLOGICKÝCH NÁSTROJŮ

Úvodní část práce obsahuje popis technologických nástrojů, které byly použity pro vývoj prototypu mobilní aplikace. Jde zejména o jazyk Swift, framework SwiftUI a službu Firebase.

1.1 Swift

Swift je moderní, kompilovaný, objektově orientovaný programovací jazyk pro obecné účely, který vznikl v roce 2014, aby nahradil Objective-C. Jeho výhodami jsou vyšší rychlost, efektivnější způsob správy paměti, lepší čitelnost a jednodušší syntax. Jazyk je otevřený zdroj (open source) a na jeho vývoji se podílí rozsáhlá komunita vývojářů. V dnešní době se stal standardem při vývoji aplikací pro všechna zařízení vyrobena společností Apple. Tento fakt je pro Apple vývojáře velkou výhodou, protože se používá stejný jazyk pro vývoj aplikací pro počítače (Mac) a mobilní zařízení (iPhone, iPad apod.). Swift disponuje řadou funkcionalit, které jsou v dnešní době pro moderní programovací jazyk samozřejmostí. Jednou s nich je již zmíněný efektivní a lehce čitelný syntax, dále typová bezpečnost, odvozování typů nebo automatické počítání referencí. Jako objektově orientovaný jazyk podporuje dědičnost, tvorbu protokolů, zapouzdření a generiku. Základní knihovna obsahuje optimalizované funkce pro efektivní a rychlou manipulaci s běžnými datovými strukturami. Swift obsahuje systém pro zpracování chyb a výjimek a umožňuje programátorovi na tyto situace reagovat. Pro vývojáře, kteří vytvářejí software pro Apple zařízení, je tento jazyk, kromě Objective-C, který ještě nevymizel, jedinou volbou. [1][2]

1.1.1 Typová bezpečnost

Swift je typově bezpečný jazyk, který při překladu kódu zachycuje chyby datových typů a umožňuje tak minimalizovat chyby v běhu programu. To znamená, že proměnné a konstanty mají přesně definovaný datový typ, který nemůže být měněn. Díky tomu se eliminuje mnoho chyb častých v jiných jazycích, jako například přiřazení hodnoty jiného datového typu do proměnné nebo předání argumentu funkce s nesprávným datovým typem. Typová bezpečnost je klíčová pro vývoj robustních a spolehlivých aplikací. [3]

1.1.2 Automatické počítání referencí

Automatic Reference Counting (ARC) je technologie používaná v jazyce Swift pro správu paměti. Jedná se o mechanismus, který sleduje množství existujících referencí na daný

objekt v paměti a automaticky uvolňuje jeho paměť, když na něj neexistuje žádná reference. Díky tomu není nutné manuálně uvolňovat paměť pomocí volání funkcí jako v jazyce C nebo Objective-C, což minimalizuje riziko paměťových úniků a tím usnadňuje vývoj aplikací. [4]

1.1.3 Dědičnost

Dědičnost je jedním z klíčových konceptů objektově orientovaného programování a Swift umožňuje definovat třídy, které mohou být odvozeny od jiných tříd. Dědičnost dává možnost třídám sdílet vlastnosti a chování rodičovské třídy a přidávat k nim další vlastnosti a chování. Tím se zjednodušuje a zrychluje proces vývoje, protože není nutné vytvářet nové třídy od základu, ale lze využít již existujících. [5]

1.1.4 Generika

Generika umožňuje vytvářet funkce, struktury a třídy, které při jejich definování pracují s parametrickými typy. Ve stádiu použití jsou parametrické typy nahrazeny konkrétními. Při definici generické funkce, struktury nebo třídy používáme zástupné jméno parametrického typu – v praxi často označovaného písmenem „T“. Pomocí generiky lze vytvářet flexibilní kód, který je funkční pro různé datové typy. Je také velmi užitečná pro kolekce. Standardní knihovna jazyka Swift obsahuje mnoho generických kolekcí, jako jsou pole, seznamy a slovníky. Lze je použít s libovolnými datovými typy, což výrazně zvyšuje jejich flexibilitu. [6]

1.2 SwiftUI

Framework SwiftUI je vývojový nástroj pro tvorbu uživatelských rozhraní v operačních systémech společnosti Apple, jako jsou iOS, macOS, watchOS a tvOS. Tento framework (softwarová struktura pro podporu programování) byl představen na konferenci WWDC 2019 a nabízí moderní přístup k tvorbě uživatelských rozhraní pomocí deklarativního programování. [7] SwiftUI umožňuje programátorům rychle a snadno vytvářet uživatelská rozhraní s minimálním množstvím kódu. Jednou z hlavních vlastností frameworku je jeho deklarativní přístup. Programátor definuje vzhled a chování uživatelského rozhraní, framework se postará o to, jak bude vypadat implementace. Další výhodou SwiftUI je jeho podpora pro řadu funkcionalit, jako jsou animace, gesta a vstup z klávesnice. Framework také nabízí řadu vestavěných komponent, jako jsou tlačítka, textová pole, seznamy apod. Tyto komponenty jsou optimalizovány pro použití v různých zařízeních a operačních

systémech. Pro vývoj ve SwiftUI je nutné používat jazyk Swift. Framework je dostupný pro vývojáře od verze Xcode 11 a podporuje všechna zařízení s operačními systémy firmy Apple. Vzhledem k tomu, že se jedná o poměrně nový framework, je třeba brát v úvahu, že všechny funkcionality nemusí být aktuálně dostupné, proto je někdy nutné použít alternativní řešení ze staršího frameworku UIKit. [8][9]

1.2.1 Struktura komponenty uživatelského rozhraní

Každá komponenta ve SwiftUI je strukturou. Tato struktura musí implementovat protokol s názvem s názvem „View“. Uvnitř struktury musí být proměnná, jejíž návratová hodnota je „some View“, což představuje jakoukoliv jinou strukturu, která implementuje protokol View. V místě před definicí proměnné „body“ je prostor pro vytvoření dalších proměnných, které bude komponenta používat. Jelikož se jedná o strukturu, je zde také možnost definovat metody, které lze v rámci bloku struktury volat. V proměnné „body“ je definován vzhled komponenty. V ní lze použít již předem definované struktury jako „Text“, „Button“, nebo „Image“. [10] Na tyto struktury lze volat funkce, které upravují jejich vzhled a funkcionalitu. Příkladem je metoda „foregroundColor“, která mění barvu textu. Apple označuje ve SwiftUI tyto metody jako „View modifier“, volně přeloženo jako „modifikátor pohledu“. [11] Pro vytvoření komponenty zavolá struktura svůj „konstruktor“ (metoda třídy nebo struktury, která se volá ve chvíli vytváření její instance). V případě, že vývojář neuvede vlastní konstruktor pomocí klíčového slova „init“, struktura použije konstruktor implicitní. [12]

1.3 Architektura MVVM

Architektura MVVM (Model-View-View Model) je v dnešní době velmi populární způsob, jak přistupovat k vývoji aplikace ve SwiftUI. Tato metodika se objevila již na začátku 21. století ve společnosti Microsoft. [13] Architektura rozděluje aplikaci do tří logických vrstev – Model, View a View model. Vrstva modelu obsahuje definici datových entit, které figurují v aplikaci. Může obsahovat také všechny operace pro jejich tvorbu, čtení, aktualizaci a mazání. Objekty, které tyto funkcionality implementují, často využívají návrhový vzor Singleton, který zajistí, aby v rámci životnosti aplikace existovala pouze jedna instance této třídy. [14] View model je vrstva, která je závislá na Modelu i na View. Ve SwiftUI aplikaci se používá technika Dependency injection (vkládání závislostí) – ve třídě View modelu je vytvořena proměnná, v niž je uložena reference na instanci jiné třídy. [15] View model se stará o uživatelské interakce, jako například stisk tlačítka. Obsahuje metody, které tyto akce obsluhují. Tyto metody volají další metody z Modelu, např. dotaz na API (Application

Programming Interface) a mění data, která vlastní View model. Tato data jsou pak předávána vrstvě View, která obsahuje definici vzhledu uživatelského rozhraní v aplikaci. Změna dat ve View modelu vyústí v opětovné vykreslení View s novými daty. [16] [17]

1.4 Xcode

Xcode je integrované vývojové prostředí pro vývoj aplikací pro Apple zařízení, jako jsou iPhone, iPad, Mac a Apple Watch. Xcode byl vytvořen společností Apple a je k dispozici zdarma na App Store. [18] Xcode nabízí kompletní sadu nástrojů pro tvorbu aplikací, od editoru kódu až po nástroje pro testování, ladění a distribuci aplikací. Xcode podporuje vývoj aplikací v programovacím jazyce Swift a Objective-C, stejně tak v C a C++. [19] Xcode také podporuje vývoj v rámci několika platforem, včetně iOS, iPadOS, macOS, watchOS a tvOS. Integrovaný editor kódu nabízí funkcionality, jako jsou automatické doplňování kódu, syntaxe a kontrola chyb, což usnadňuje vývoj a umožňuje vývojářům soustředit se na logiku aplikace. Xcode také nabízí nástroje pro vizualizaci a úpravu uživatelského rozhraní. Xcode obsahuje funkce pro testování a ladění aplikací, včetně nástrojů pro profilování a analýzu výkonu. Tyto nástroje umožňují identifikovat a odstranit potenciální problémy v aplikaci, což vede k tomu, že je pak aplikace stabilnější. Kromě toho Xcode nabízí funkce pro správu verzí a kolaboraci s týmem, včetně podpory pro distribuovaný systém správy verzí Git. Tyto funkce vývojáři využívají k jednodušší a efektivnější spolupráci a řízení vývoje aplikací. [20]

1.4.1 Náhled a simulátor

Xcode nabízí několik nástrojů, které pomáhají vývojářům vytvořit a ladit aplikace snadno a efektivně. Těmito nástroji jsou „náhled“ (preview) a „simulátor“ (simulator). Oba nástroje mají své výhody a nevýhody, takže je důležité vědět, kdy použít jednu z nich. Náhled se používá pro rychlé a snadné testování designu a interakce uživatele v rámci Xcode. Umožňuje rychle vyzkoušet a upravit design, aniž by bylo třeba spouštět celou aplikaci v simulátoru. Náhled se hodí pro rychlou iteraci a ladění designu, kde nejde o komplexní funkčnost. Simulátor na druhé straně simuluje skutečné zařízení. Vývojář může testovat aplikaci v reálných podmínkách. Simulátor umožňuje vývojářům testovat aplikaci v různých rozlišeních obrazovek a zařízeních, což je důležité pro optimalizaci aplikace pro různé modely zařízení. Simulátor testuje funkčnost aplikace v reálných situacích, jako je například odesílání a přijímání dat přes internet nebo práce se senzory v zařízení. [21][22]

1.4.2 Testování aplikace na fyzickém zařízení

Nejspolehlivějším způsobem, jak otestovat aplikaci, je spustit ji na fyzickém zařízení. V případě, že potřebujeme využít funkcionality, které simulátor nenabízí (např. kameru), musíme připojit reálné zařízení. Dalším důvodem, proč použít fyzické zařízení je, že aplikace simulátoru je spouštěna na Macu, tím pádem výkon konkrétního simulovaného zařízení neodpovídá realitě. Prvním krokem pro testovaní aplikace na fyzickém zařízení je jeho připojení pomocí kabelu k Macu a dále jeho odemknutí. Před spuštěním aplikace je třeba splnit řadu předpokladů. Nejprve je nutné uvést Apple ID uživatele v nastavení účtu v Xcode. Dále specifikovat validní „tým“ v sekci „podepisování a schopnosti“. V případě, že chceme aplikaci publikovat na App Store, je třeba být součástí Apple vývojářského programu (Apple Developer Program). Toto členství je zpoplatněno částkou $99 ročně. [23] Poté lze vytvořit vývojářský tým, pod kterým se bude aplikace publikovat. Bez poplatku je také možné aplikaci testovat na reálném zařízení vytvořením provizorního týmu s omezenými funkcionalitami. Posledním krokem je povolení vývojářského režimu v hlavním nastavení testovaného zařízení. Poté je třeba pouze vybrat fyzické zařízení z nabídky v Xcode a stisknout tlačítko pro spuštění, nebo vybrat možnost „Product-Run“. [24]

1.5 Firebase

Firebase je cloudová platforma pro vývoj aplikací. Byla založena v roce 2011 jako start-up se jménem Envolve a v roce 2014 byla koupena společností Google. [25] Od té doby se stala jedním z nejpopulárnějších nástrojů pro vývoj aplikací a poskytuje širokou škálu funkcí a služeb pro vývojáře. Platforma je zaměřena na zlepšení vývoje aplikací a snižování časové náročnosti na jejich vývoj. Nabízí funkce jako správu uživatelů, autentizaci, databázi, hosting, analytiku a mnoho dalších. Tyto funkce jsou k dispozici jako služby, které lze snadno integrovat do aplikací. Vývojáři mohou využít Firebase při vývoji aplikací pro různé platformy, včetně iOS, Android a webových aplikací. Platforma je do omezené míry použití zdarma a nabízí rozšiřitelnost pomocí knihoven a pluginů od třetích stran. Výběr Firebase při vývoji iOS aplikací má řadu výhod. Snadná integrace – poskytuje jednoduchou integraci s aplikacemi pomocí knihoven pro různé platformy, včetně iOS. Poskytuje databázi v reálném čase, což znamená, že data jsou aktualizována okamžitě, jakmile se změní. To umožňuje vytvářet aplikace s živými daty, jako je například aplikace pro sdílení polohy. Dále poskytuje služby pro správu uživatelů, včetně autentizace, lze tak snadno implementovat a spravovat uživatelské účty v aplikaci. Tyto služby nabízejí širokou škálu

možností autentizace, včetně e-mailu a hesla, telefonního čísla, sociálních sítí a dalších. Firebase nabízí mnoho analytických nástrojů, pomocí kterých lze sledovat a analyzovat chování uživatelů v aplikaci. Součástí je také jednoduchý a spolehlivý hosting pro aplikace, proto je lze snadno publikovat. [26][27][28]

1.5.1 Vytvoření a integrace projektu do iOS aplikace

Integrace platformy Firebase do iOS projektu zahrnuje několik kroků. Prvním z nich je vytvoření účtu na Firebase, lze použít také již existující Google účet. Následuje vytvoření projektu – je nutné vytvořit nový projekt a nakonfigurovat ho tak, aby vyhovoval potřebám aplikace. Poté je třeba aplikaci registrovat. V případě integrace Firebase pro iOS je nutné vyplnit tzv. „bundle ID“. Jde o unikátní identifikátor aplikace, který lze nalézt v Xcode v nastavení projektu. [29] Dalším krokem je přidání vygenerovaného souboru s názvem „GoogleService-Info.plist“ do Xcode projektu. Soubor obsahuje konfigurační data, která jsou potřebná k připojení k danému Firebase projektu přes Firebase SDK (software development kit). Dále je nutno stáhnout Firebase SDK přes Swift Package Manager. Nástroj je určen pro automatizaci procesu stahování, kompilování a linkování závislostí do projektu. [30] Posledními dvěma kroky jsou importování knihovny Firebase do iOS aplikace a konfigurace SDK. Pro aplikaci používající SwiftUI se v souboru s označením „@main“ použije příkaz „import Firebase“. Konfiguraci SDK lze provést voláním metody „FirebaseApp.configure()“ v konstruktoru úvodní struktury aplikace. [31] Po provedení těchto kroků by měla být integrace Firebase do iOS projektu úspěšně dokončena a je možno začít využívat všechny služby, které nabízí. [32]

1.5.2 Auth

Firebase Auth (authentication) je služba, která se stará o ověřování a identifikaci uživatelů v aplikaci. S Auth lze snadno a rychle ověřovat uživatele pomocí e-mailu a hesla, sociálních médií, telefonního čísla a dalších metod. Tento nástroj je výborným řešením pro vývojáře, kteří hledají jednoduchý a efektivní způsob, jak integrovat ověřování uživatelů do svých aplikací pro iOS. Metody ověřování lze snadno implementovat pomocí funkcí z Firebase Auth API (application programming interface). Ověřování pomocí e-mailu a hesla je nejběžnější formou ověřování uživatelů. Služba nabízí jednoduchý a intuitivní API pro registraci a ověření uživatele pomocí e-mailu a hesla. Funkcionality jako je registrace, přihlášení nebo reset zapomenutého hesla lze se službou jednoduše realizovat. Ověřování pomocí telefonního čísla je další oblíbenou formou ověřování uživatelů. Auth nabízí

jednoduchou integraci s aplikacemi pro ověřování pomocí telefonního čísla a SMS zprávy. Tuto funkci lze snadno implementovat a umožňuje vývojářům ověřit uživatele pomocí ověřovacího kódu zaslaného na telefonní číslo uživatele. [33][34]

1.5.3 Firestore

Firestore je NoSQL databáze, kterou poskytuje Firebase. Je to jedna z nejjednodušších možností pro ukládání a získávání dat v aplikaci, zejména v aplikacích mobilních a webových. Firestore se odlišuje od tradičních SQL databází tím, že nepracuje s tabulkami, ale s dokumenty. Tyto dokumenty se skládají z jednotlivých položek, které mohou mít různé atributy. [35] Jednou z hlavních výhod použití této služby je její schopnost automaticky synchronizovat data mezi více zařízeními. Data ve Firestore jsou uložena v cloudu a mohou být snadno přístupná všem aplikacím, které jsou registrovány k jednomu projektu v rámci Firebase. [36] To umožňuje vývojářům vyhnout se složitému procesu vývoje a správy serveru, který by udržoval data a synchronizaci mezi různými zařízeními. Firestore má také vynikající funkce pro správu dat, jako jsou například transakce, které zajišťují, že data budou změněna pouze v případě, že všechny dílčí operace v rámci transakce proběhnou úspěšně. [37][38] K dispozici jsou také dotazovací funkce, které umožňují vývojářům snadno vyhledávat a získávat data z databáze. [39] Firestore Rules (pravidla) jsou jedním z klíčových prvků pro zabezpečení a správu dat. Tato pravidla fungují jako firewally, které určují, kdo má přístup k datům v databázi a jaké operace s nimi mohou provádět. Vývojáři mohou definovat pravidla, která například povolují pouze přihlášeným uživatelům přistupovat k datům, nebo omezit, kteří uživatelé mohou provádět určité operace, jako je čtení, zápis nebo mazání. [40]

1.5.4 Storage

Storage je služba v rámci Firebase, ve které lze ukládat soubory v cloudu, a která umožňuje vývojářům k nim snadno přistupovat. Tuto službu lze použít pro ukládání souborů v rámci aplikace, například obrázky profilů uživatelů, videa, audio, dokumenty apod. Tyto soubory jsou uloženy na serveru a vývojáři mohou získat odkaz na stažení pro zpětné získání dat. Služba umožňuje vytvářet aplikace, ve kterých mohou uživatelé ukládat a sdílet své soubory snadno a bezpečně. Storage také poskytuje funkce pro správu souborů s možností stahování, mazání, úpravy a další. Služba také podporuje ukládání souborů na více platformách, včetně iOS, Android a webových aplikací. [41][42]

1.6 Adobe Xd

Adobe Xd (Experience Design) je softwarový nástroj pro návrh uživatelských rozhraní, který vyvinula společnost Adobe. Umožňuje designérům a vývojářům tvorbu prototypů uživatelských rozhraní pro webové i mobilní aplikace. Cílem Adobe Xd je poskytovat zjednodušený a účinný nástroj pro návrh a testování uživatelských rozhraní. [43] V Adobe Xd mohou designéři kreativně pracovat se vzhledem a interakcemi aplikace, včetně animací a vizuálních efektů. Vývojáři pak mohou exportovat HTML (Hypertext Markup Language) kód pro implementaci webové aplikace. [44] Software nabízí řadu funkcí pro spolupráci mezi designéry a vývojáři, včetně sdílení a komentování návrhů, a také pro spolupráci s klienty a zákazníky, kteří mohou návrh testovat a poskytovat zpětnou vazbu. [45] V případě vytváření aplikace pro iOS lze využít Adobe Xd ke kreativnímu návrhu uživatelského rozhraní a interakcí aplikace. Designéři mohou vytvořit a testovat prototypy v reálných podmínkách pomocí funkce pro emulaci zařízení iOS. Adobe Xd nabízí roční paušál nebo měsíční předplatné a také plán pro studenty a učitele za sníženou cenu. [46]

1.6.1 Základní nástroje

V této části jsou uvedeny základní nástroje, se kterými lze v softwaru Adobe Xd pracovat a vytvářet prototypy uživatelského rozhraní: [47]

- Selection Tool – nástroj pro vybrání aktivně upravovaného objektu
- Rectangle Tool – nástroj pro tvorbu obdélníku, obsahuje výplň, konturu, stíny, hraniční poměr atd.
- Ellipse Tool – nástroj pro tvorbu elipsy, při držení klávesy „Shift“ lze vykreslit kruh
- Polygon Tool – nástroj pro vytvoření jakéhokoliv polygonu
- Line Tool – nástroj pro vykreslení rovné čáry
- Pen Tool – nástroj pro tvorbu komplexních nepravidelných tvarů
- Text Tool – nástroj pro přidání textu, lze upravit velikost, font, barvu, vzdálenost písmen atd.
- Artboard Tool – nástroj pro vytvoření nové plochy, kde se umísťují jednotlivé grafické elementy, je zde možnost vytvořit různé plochy, definované pro mobil, web atd.
- Zoom (Z) – nástroj pro přibližování návrhu

1.6.2 Mobilní aplikace pro náhled na fyzickém zařízení

Při tvorbě prototypu uživatelského rozhraní v Adobe Xd je velmi užitečné použít funkcionalitu pro náhled designu na fyzickém zařízení. Vývojář tak může otestovat návrh na různých typech zařízení a validovat ho. Prvním krokem je stažení mobilní aplikace Adobe Xd na iOS, nebo Android. Dále je třeba připojit fyzické zařízení patřičným kabelem k počítači. Po otevření mobilní aplikace je aktivně vybraná plocha (Artboard) zobrazena na fyzickém zařízení. Velkou výhodou je, že změny, které jsou v návrhu prováděny, jsou v reálném čase reflektovány v náhledu na fyzickém zařízení. Není tedy potřeba aplikaci při úpravě designu znovu aktivovat. [48]

2 POPIS POUŽITÝCH PARADIGMAT A NÁVRHOVÝCH VZORŮ

Následující část obsahuje popis použitých paradigmat a návrhových vzorů, které byli použity při implementaci prototypu v jazyce Swift.

2.1 Singleton

Singleton je návrhový vzor, který zajišťuje, že třída má pouze jednu instanci a poskytuje jednotný bod přístupu k této instanci. Hlavním účelem tohoto vzoru je kontrolovat počet instancí třídy v aplikaci a zajistit, aby třída měla pouze jednu instanci, která je globálně přístupná.

Výhody vzoru Singleton:

- zajišťuje, že třída má pouze jednu instanci
- poskytuje jednotný bod přístupu k instanci
- důsledkem je snižování množství použité paměti

Nevýhody vzoru Singleton:

- slabá testovatelnost – vzor Singleton ztěžuje testování aplikace, protože třída má pouze jednu instanci, což znamená, že nelze otestovat vliv jednotlivých změn na instanci bez ovlivnění jiných částí aplikace
- závislost na stavu – Singleton má stav, který může být ovlivněn jinými částmi aplikace, což může vést ke komplikacím a chybám [49] [50]

V jazyce Swift lze k implementaci vzoru Singleton použít statickou vlastnost členské proměnné a privátní konstruktor třídy k zajištění, že bude vytvořena pouze jedna instance třídy. [51] Zde je příklad implementace třídy se vzorem Singleton:

```
// definition
class Singleton{
    // property to get instance
    static let shared = Singleton()
    // private initializer
    private init() { }
}
// usage
let singleton = Singleton.shared
```

Obrázek 1 Implementace návrhového vzoru Singleton

2.2 Dependency injection

Vzor Dependency injection je výkonný nástroj pro oddělení závislostí v aplikaci. Toto oddělení umožňuje snadnější testování, údržbu a rozšiřování aplikace. V tomto návrhovém vzoru se realizuje předávání závislostí objektu ve formě jiných objektů. Objekt, který přijímá závislost, bývá označován jako „klient“ a objekt, který je předáván jako „služba“. [52][53]

Výhody použití vzoru Dependency injection:

- snadné testování – lze snadno testovat jednotlivé části aplikace odděleně od ostatních částí, což umožňuje lépe porozumět funkčnosti aplikace a opravit případné chyby
- slučitelnost s jinými knihovnami – lze snadno integrovat knihovny a moduly do aplikace, aniž by bylo nutné upravovat stávající kód
- oddělení závislostí – umožňuje oddělení závislostí mezi jednotlivými částmi aplikace, což zlepšuje čitelnost a přehlednost kódu

Nevýhody použití vzoru Dependency injection:

- složitost kódu – může být pro některé vývojáře složitý na pochopení a implementaci, což může vést k chybám a komplikacím
- složitější ladění kódu – může být těžké najít chybu v předávané závislosti, zejména pokud se jedná o framework třetí strany
- složité nastavení – někdy může být složité nastavit vzor závislosti tak, aby odpovídal potřebám aplikace, což může vést k dodatečným úpravám kódu a komplikacím [54] [55] [56]

Aplikování vzoru Dependency injection lze provést několika způsoby:

- přes konstruktor – závislosti se předávají jako argumenty do konstruktoru třídy. Tato metoda je nejjednodušší a často se používá pro menší projekty

```swift
// object to represent dependency
class Dependency{

}

// class to inject dependency
class MyClass{

    private var dependency: Dependency

    // dependency injection in initializer
    init(dependency: Dependency){
        self.dependency = dependency
    }
}
```

Obrázek 2 Implementace Dependency injection přes konstruktor

- přes členskou proměnnou třídy – závislosti se předávají jako proměnné třídy. Tuto metodu lze použít v případě, že je potřeba přistupovat k závislostem z různých částí třídy

```swift
// object to represent dependency
class Dependency{

}

// class to inject dependency
class MyClass{
    // dependency injection with class property
    private var dependency: Dependency?
}
```

Obrázek 3 Implementace Dependency injection přes členskou proměnnou

- přes argument metody – závislosti se předávají jako argumenty do metody třídy. Tuto metodu lze použít v případě, že je potřeba přistupovat k závislostem pouze z určité metody [57][58]

```swift
// object to represent dependency
class Dependency{

}

// class to inject dependency
class MyClass{
    // dependency injection with class method
    func foo(dependency: Dependency){

    }
}
```

Obrázek 4 Implementace Dependency injection přes argument metody

2.3 Dědičnost

Součástí jazyka Swift, jakožto objektově orientovaného jazyka, je princip dědičnosti mezi třídami. Třída může dědit metody, proměnné a další charakteristiky od jiné třídy. V jazyce Swift je třída, která dědí označována jako „podtřída“ (subclass) a třída od které se dědí jako „supertřída“ (superclass). Důvodem používání dědičnosti je, že podtřída může použít, rozšířit, nebo modifikovat již vytvořenou funkcionalitu ze supertřídy. Prvním krokem pro ustanovení hierarchie mezi třídami je vytvoření tzv. „základní třídy“ (base class). Jde o třídu, která nedědí vlastnosti od jiné třídy a měla by reprezentovat nejobecnější vyjádření entity v hierarchii. Poté je vytvořena jedna, nebo více podtříd, které dědí vlastnosti od základní třídy. Příkladem jednoduché hierarchie je třída „Vozidlo“, jakožto základní třída a „Bicykl“, jakožto podtřída dědící od základní třídy. Třída „Bicykl“ může používat zděděné vlastnosti od základní třídy pomocí klíčového slova „super“ a také přidat vlastní funkcionality. V jazyce Swift lze z třídy „Bicykl“ opět dědit v nové podtřídě, která zdědí charakteristiky třídy „Vozidlo“ i „Bicykl“. V případě, že chceme modifikovat již existující funkcionalitu supertřídy v podtřídě, můžeme ji tzv. „přepsat“. Lze například přepisovat proměnné a metody supertřídy pomocí klíčového slova „override“ v definici podtřídy. [59][60]

```swift
// superclass
class Vehicle{
    let model: String
    let serialNumber: String

    init(model: String, serialNumber: String){
        self.model = model
        self.serialNumber = serialNumber
    }

    func printModel(){
        print("Vehicle: \(model)")
    }
}

// subclass
class Bicycle: Vehicle{
    func decodeSerialNumber(){
        // super keyword to reference superclass
        print("Decoded: \(super.serialNumber)")
    }
    // overriding method from superclass
    override func printModel() {
        print("Bicycle: \(model)")
    }
}
```

Obrázek 5 Příklad použití dědičnosti tříd

2.4 Protokoly

Protokoly v jazyce Swift se používají k definici požadavků, které musí datové typy, které vyhovují danému protokolu, implementovat. Tyto požadavky by měly vyjadřovat nástroje pro řešení konkrétní funkcionality nebo problému. Definice protokolu může obsahovat metody, proměnné a další požadavky. Datové entity, které mohou implementovat protokol jsou třída, struktura a enumerace (výčet). Definice protokolu v jazyce Swift obsahuje klíčové slovo „protocol“ a volitelný název. V těle definice protokolu je prostor pro výčet proměnných bez přiřazených hodnot a metod bez konkrétních implementací. V definici tříd, struktur, nebo enumerací, které budou protokol používat, se již bude nacházet konkrétní implementace požadovaných metod. V těle protokolu lze mimo jiné definovat i požadavek na tvar konstruktoru, který musí být v třídě, či struktuře, která protokol implementuje.

```swift
// protocol definition
protocol ShapeProtocol{
    // requirements
    var name: String { get }
    init(name: String)
    func getName() -> String
}

// protocol implementation on class
class Rectangle: ShapeProtocol{
    var name: String

    required init(name: String) {
        self.name = "Rectangle"
    }

    func getName() -> String {
        return self.name
    }
}
```

Obrázek 6 Příklad použití protokolu s třídou

Protokoly lze použít i jako datové typy. Například jako parametr metody, konstruktoru, návratovou hodnotu funkce, typ konstanty, typ proměnné, typ položek v poli, slovníku atd. Výhodou je, že místo jednoho konkrétního datového typu můžeme použít jakoukoliv instanci jiného datového typu, který implementuje daný protokol. [61]

```swift
// protocol as variable type
var shape: ShapeProtocol?
// protocol as array type
let shapesArray: [ShapeProtocol] = []
// protocol as dictionary type
let shapesDictionary: [String:ShapeProtocol] = [:]
// protocol as function argument
func printShape(shape: ShapeProtocol){ }
```

Obrázek 7 Příklad použití protokolu jako datového typu

II. PRAKTICKÁ ČÁST

3 PRŮZKUM A SROVNÁNÍ KONKURENČNÍCH APLIKACÍ

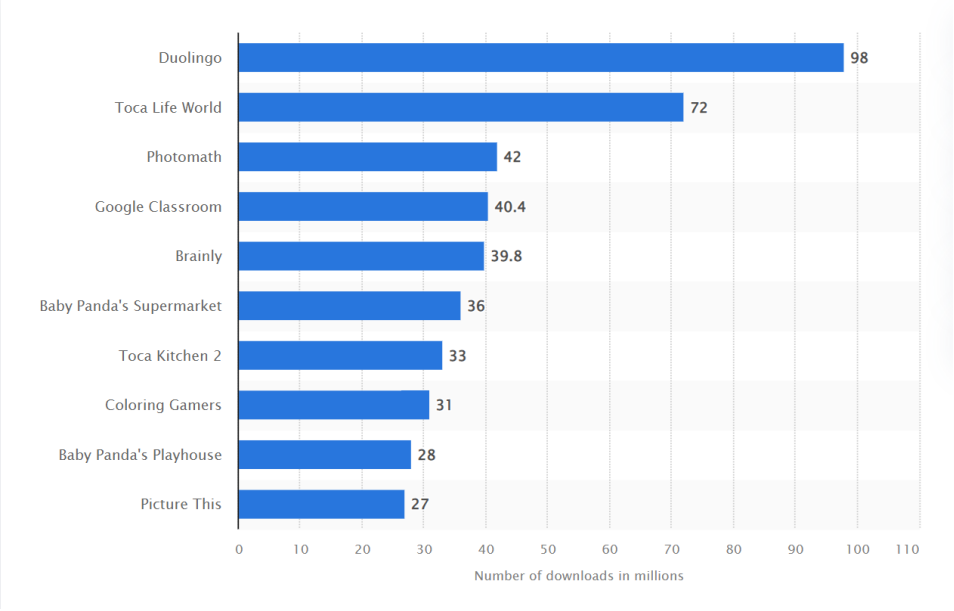
Základem této práce je vytvoření analýzy mobilních aplikací pro vzdělávání. Tato analýza bude sloužit k identifikaci nejefektivnějších mobilních aplikací a k pochopení toho, jaké parametry přispívají k jejich úspěchu. Cílem je porovnat výkon aplikací a navrhnout prototyp nové vzdělávací aplikace, která by uživatelům nabídla ještě lepší zážitek. Aplikace se budou hodnotit podle následujících parametrů:

- Uživatelské rozhraní: Jak snadno se aplikace ovládá a používá?
- Kvalita obsahu: Jak dobře aplikace prezentuje informace a jak jsou relevantní?
- Funkčnost: Nabízí aplikace širokou škálu funkcí a nástrojů pro zlepšení vzdělávacího zážitku?
- Dostupnost: Jak snadno se aplikace dostává k různým zařízením a platformám?
- Zpětná vazba a podpora: Jak dobře aplikace poskytuje uživatelům zpětnou vazbu a podporu?

Pro každý parametr bude uděleno bodové hodnocení od 0 do 10, přičemž 10 znamená nejvyšší možné hodnocení. Každý parametr má váhu založenou na jeho důležitosti a celkové hodnocení aplikace bude vypočteno podle následujícího vzorce:

Celkové hodnocení = (Uživatelské rozhraní x 0,2) + (Kvalita obsahu x 0,3) + (Funkčnost x 0,3) + (Dostupnost x 0,1) + (Zpětná vazba a podpora x 0,1)

Výsledkem analýzy bude identifikace nejlepších mobilních aplikací pro vzdělávání a vyplynutí důležitých parametrů pro vytvoření efektivního prototypu. V analýze se budou hodnotit tři aplikace pro vzdělávání: Duolingo, Google Classroom a Brainly. Tyto platformy byly vybrány na základě statistických údajů z portálu Statista o celosvětovém počtu stažení aplikací z kategorie pro vzdělávání v roce 2022. [62]



Obrázek 8 Graf nejstahovanějších vzdělávacích aplikací za rok 2022

3.1 Duolingo

Duolingo je světově nejpopulárnější aplikace pro učení jazyků. Cílem společnosti je vytvořit nejlepší platformu pro učení jazyků, která bude široce dostupná pro všechny. V průběhu učení prochází uživatel krátkými lekcemi, za které získává body a odemyká nové lekce. Kromě samotné aplikace společnost poskytuje vlastní certifikát s názvem Duolingo English Test, který je uznáván mnoha institucemi po celém světě. Základní verze aplikace je zdarma s možností předplatit si program Super Duolingo, který obsahuje extra funkcionality a eliminuje reklamy v aplikaci. [63] [64]

3.1.1 Shrnutí hodnocení

Uživatelské rozhraní aplikace Duolingo je velmi kvalitně zpracováno. Pro dospělého člověka může design působit dětsky, ale mladší uživatele jistě zaujmou všudypřítomné animace a ilustrace. Šest ikon v hlavním menu působí při běžném používání zbytečně, jejich počet by se určitě dal redukovat pro zvýšení přehlednosti. Duolingo podporuje světlý a tmavý mód, což je velké plus. Platforma obsahuje velkou možnost personifikace, jako nastavení učeného jazyka, nastavení cílů apod., což je její velká výhoda. Aplikace byla testována jeden týden pro učení anglického a německého jazyka. Za toto období byla zpozorována pouze jedna drobná chyba ze všech lekcí. Kvalita obsahu je tedy vysoká a lekce jsou velmi různorodé z různými typy úkolu, pro cvičení psaní, výslovnosti apod. Aplikace

je zaměřená pouze na učení jazyků, na vzdělávání jiného druhu stavěná není. Dostupnost aplikace je velká. Je k dispozici na App Store, Google Play i jako webová aplikace. Možnost zpětné vazby a podpory je přítomna v mobilní i webové aplikaci. Uživatel může navštívit tzv. „centrum nápovědy“ s odpověďmi na často kladené otázky, nebo kontaktovat vývojáře pomocí e-mailu s předem vygenerovanými logy z aplikace.

3.2 Google Classroom

Google Classroom je nástroj pro spolupráci a komunikaci mezi uživateli a je k dispozici na mobilních zařízeních i jako webová aplikace. V rámci platformy lze použít další služby od společnosti Google jako je Gmail, Google Docs, nebo Google Calendar. Učitelé mají možnost vytvořit třídu, do které se mohou připojit jednotliví studenti. Učitel může zahájit videohovor, zadávat a hodnotit úkoly, udělovat známky apod. Studenti se mohou zapojovat pomocí chatu a dalších nástrojů. [65]

3.2.1 Shrnutí hodnocení

Nejkvalitnějším aspektem aplikace jsou funkcionality pro spolupráci mezi učiteli a studenty. Systém nabízí velice intuitivní a jednoduché uživatelské rozhraní. Mobilní aplikace na iOS nepodporuje tmavý mód, což může být pro uživatele, kteří jsou na něj zvyklý, nepříjemné. Hlavní nabídka menu v jednotlivých třídách je velmi přehledná a uživatel nemusí hledat ve velkém množství tlačítek. Aplikace byla testována při založení reálné online třídy, kde studenti nahrávali videa, ve kterých se představovali a učitelé na ně mohli napsat zpětnou vazbu. Všechny funckionality sdílení souborů fungovaly bezchybně s integrací aplikace Google Docs. Kvalita obsahu není udávána platformou, protože zdroje nahrávají jednotliví učitelé. Dostupnost je opět velká, aplikace je k dispozici na všech platformách. Mobilní aplikace na iOS poskytuje podobně jako Duolingo sekci s návodem pro použití platformy a poté možnost zaslání zpětné vazby s možností přidání screenshotů a logů.

3.3 Brainly

Brainly je platforma, ve které mohou uživatelé formulovat otázky, na které odpovídají ostatní uživatelé ve stylu fóra. Hlavní ideou je pomáhat studentům s úkoly a zodpovídat jejich otázky. Je důležité poznamenat, že odpovědi na otázky zde neposkytují kvalifikovaní profesionálové, nýbrž komunita uživatelů aplikace. Společnost však označuje příspěvky, které byly ověřeny experty, aby se mohli uživatelé na pravdivost uvedené odpovědi spolehnout. [66]

3.3.1 Shrnutí hodnocení

Uživatelské rozhraní aplikace je velmi jednoduché a neobsahuje žádné rušivé elementy, nebo příliš mnoho barev. Mobilní aplikace na iOS nepodporuje tmavý mód. Celá navigace mobilní aplikace se skládá z pěti přehledných sekcí. Nejlepší funkcionalitou je vyhledávání již existujících otázek. Uživatel může svou otázku buď napsat na klávesnici, namluvit ji hlasově, nebo naskenovat článek, či učebnici. Poté systém zobrazí již existující otázky k hledanému tématu s odpověďmi. Pokud jsou tyto informace nedostatečné, může uživatel vytvořit zcela novou otázku. Zajímavá je taky sekce, která nabízí učebnice jako zdroje informací. Součástí fóra jsou pouze textové otázky a odpovědi. Možnosti pro komunikaci mezi uživateli jsou zde poměrně omezené. Kvalita obsahu je velmi různorodá. Vzhledem k tomu, že odpovědi může poskytovat kdokoliv, nejsou informace vždy stoprocentně akurátní. To je do určité míry eliminováno pomocí systému ověřování odpovědí. Aplikace je opět k dispozici na všech platformách. Mobilní aplikace obsahuje možnost kontaktování společnosti pomocí chatu. Neobjevuje se zde sekce s příručkou, jak platforma funguje a jak ji používat.

3.4 Výsledek průzkumu

Tabulka 1 Výsledek průzkumu konkurenčních aplikací

aplikace	rozhraní	obsah	funkčnost	dostupnost	zpětná vazba	skóre
Duolingo	6	8	7	10	9	**7,6**
Google Classroom	8	5	8	10	8	**7,3**
Brainly	7	7	9	10	7	**7,9**

Z testování aplikací po dobu jednoho týdne vyplynuly nejdůležitější aspekty pro tvorbu vzdělávací mobilní aplikace a návrh kvalitního uživatelského rozhraní, které budou aplikovaný při tvorbě prototypu.

4 PŘEDSTAVENÍ A POŽADAVKY PROTOTYPU

LearnKit je prototyp vzdělávací mobilní aplikace, která byla vytvořena ve SwiftUI pro iOS. Jejím hlavním účelem je rychle a efektivně poskytovat uživatelům kvalitní a přesné informace ve formě materiálů od ověřených expertů na dané téma. Hlavní entitou v aplikaci je "List", který obsahuje různé typy materiálů, jako jsou kvízy (psaní odpovědí, pravda/lež, volba z více možností, řazení), hlasové záznamy, videa, odkazy a další. Jedním z příkladů seznamu může být "Sluneční soustava". Materiály, které mohou být součástí tohoto seznamu, jsou mapa sluneční soustavy, video o planetách, hlasové záznamy o historii výzkumu vesmíru, odkazy na relevantní webové stránky a další. Tyto informace mohou poskytnutovat ověření experti v astronomii nebo kosmologii, kteří uvedou nejnovější a nejvíce relevantní informace. Tato aplikace je užitečná pro širokou škálu lidí, kteří se chtějí učit a zlepšovat své znalosti v různých oblastech. Jedním z hlavních přínosů aplikace je to, že poskytuje uživatelům snadný a efektivní způsob, jak se učit nové informace. Uživatelé mohou také používat různé typy materiálů, které jsou k dispozici v seznamu, aby si přizpůsobili své učení jejich vlastnímu stylu učení. Dalším přínosem této aplikace je, že je snadno dostupná a použitelná kdykoliv a kdekoliv. Uživatelé přistupují k Listům a materiálům v aplikaci na svých mobilních telefonech, což znamená, že mohou studovat a učit se i během cestování nebo čekání. Kromě toho může tato aplikace být užitečná i pro vzdělávací instituce a školy, kde mohou být Listy využívány jako doplňkový zdroj informací. Učitelé a instruktoři mohou použít seznamy jako podporu pro výuku a testování znalostí studentů. V neposlední řadě, aplikace může být užitečná i pro profesionály a odborníky, kteří potřebují udržovat své znalosti a dovednosti v aktuálním stavu. Listy mohou být použity jako rychlý a snadný zdroj informací, které lze konzultovat při řešení konkrétních problémů. Celkově je LearnKit užitečným nástrojem pro vzdělávání a rozvíjení znalostí v různých oblastech. Aplikace může být užitečná pro školy, studenty, odborníky a všechny, kteří se chtějí učit nové informace a rozvíjet své znalosti.

4.1 Funkcionální požadavky

Následující tabulka obsahuje výčet funkcionálních požadavků pro prototyp LearnKit, které budou splněny pro první iteraci vývoje. Každý požadavek obsahuje unikátní identifikátor, popis a prioritu pro zhotovení.

Tabulka 2 Definice funkcionálních požadavků prototypu

ID	popis	priorita
FR001	Uživatel má možnost se registrovat do aplikace pomocí e-mailu a hesla.	vysoká
FR002	Uživatel má možnost se přihlásit do aplikace pomocí e-mailu a hesla.	vysoká
FR003	Uživatel má možnost se v nastavení aplikace odhlásit ze svého účtu.	vysoká
FR004	Uživatel má možnost si zažádat o resetování hesla ke svému účtu pomocí e-mailu.	střední
FR005	Po registraci uživatele systém odešle ověřovací e-mail s odkazem pro aktivaci účtu.	střední
FR006	Uživatel má možnost vytvořit a editovat nový List s názvem, kategorií a barvou.	vysoká
FR007	Uživatel má možnost přidat herní kvízy v rámci Listu. Kvíz obsahuje čtyři druhy otázek – napsaní odpovědi, výběr z více možností, pravda/nepravda a seřazení více možností.	vysoká
FR008	Uživatel má možnost spustit herní kvíz a upřesnit jeho nastavení jako časový limit, počet nápověd, pořadí otázek apod.	vysoká
FR009	Uživatel má možnost si zobrazit výsledky po vyplnění herního kvízu.	vysoká
FR010	Uživatel má možnost přidat do Listu soubor s videem, jeho názvem a popisem.	vysoká
FR011	Uživatel má možnost přehrávat, stopovat a posunout video.	vysoká
FR012	Uživatel má možnost vytvořit novou hlasovou nahrávku a přidat ji do Listu.	vysoká
FR013	Uživatel má možnost přehrávat, stopovat a posunout hlasovou nahrávku.	vysoká
FR014	Uživatel má možnost přidat do Listu hypertextový odkaz na webovou stránku.	vysoká
FR015	Uživatel má možnost navštívit hypertextový odkaz na webovou stránku pomocí prohlížeče Safari.	vysoká
FR016	Uživatel má možnost přepínat mezi jednotlivými Listy a měnit tak aktivní List.	vysoká
FR017	Uživatel má možnost přepínat mezi jednotlivými typy výukových materiálu v aktivním Listu (kvízy, video, zvukové nahrávky, odkazy).	vysoká
FR018	Uživatel má možnost vyhledat ověřeného autora nebo List pomocí naskenování QR kódu kamerou zařízení.	střední
FR019	Uživatel má možnost si zobrazit informace a vyhledaném autorovi, včetně Listů, které publikoval.	vysoká
FR020	Uživatel má možnost přidat/odebrat ověřeného autora nebo List do seznamu oblíbených.	střední
FR021	Uživatel má možnost vygenerovat QR kód reprezentující ověřeného autora nebo List.	střední
FR022	Uživatel má možnost si rozsvítit baterku mobilního zařízení při procesu skenování QR kódu.	nízká
FR023	Uživatel má možnost uložit vygenerovaný soubor jako obrázek do svého alba.	střední
FR024	Uživatel má možnost si vybrat, zda chce aplikovat předdefinovanou šablonu s logem aplikace a popisem QR kódu.	nízká
FR025	Uživatel má možnost navštívit účty sociálních sítí od ověřeného autora.	nízká
FR026	Uživatel má možnost třídit položky v seznamu oblíbených. První možnost je zobrazit pouze oblíbené autory, nebo pouze Listy, nebo vše dohromady.	střední
FR027	Uživatel má možnost v aplikaci povolit haptickou odezvu, zvuky a muziku.	nízká
FR028	Uživatel má možnost vyjádřit zpětnou vazbu tvůrcům aplikace přes e-mail.	střední
FR029	Uživatel má možnost pokračovat v interakci s aplikaci při přehrávání médií.	vysoká
FR030	Aplikace bude podporovat tmavý mód pro uživatelské rozhraní.	nízká

4.2 Nefunkcionální požadavky

Tabulka 3 Definice nefunkcionálních požadavků prototypu

ID	popis
NR001	Aplikace by měla používat šifrovanou komunikaci se serverem pro ochranu soukromých dat poskytnutých uživatelem.
NR002	Aplikace by měla podporovat zařízení iPhone s operačním systémem iOS od verze 16.0.
NR003	Aplikace by měla nabízet jednoduchý a intuitivní uživatelské prostředí a navigaci.
NR004	Aplikace by měla poskytovat možnost vytvoření personifikovaných uživatelských účtů.
NR005	Aplikace by měla obsahovat mechanismus pro vyjádření zpětné vazby od uživatelů.
NR006	Aplikace by měla dodržovat oficiální pokyny pro přístupnost aplikace od společnosti Apple (Apple accessibility guidelines).

4.3 Srovnání prototypu s konkurencí

V následující části je uvedeno základní srovnání vlastností prototypu LearnKit s dříve uvedenými konkurenčními aplikacemi. Srovnání je rozděleno do pěti částí, které kopírují parametry pro hodnocení z průzkumu.

4.3.1 Uživatelské rozhraní

Na základě poznatků z průzkumu konkurenčních aplikací bylo vytvořeno uživatelské rozhraní pro prototyp LearnKit. Cílem bylo implementovat jednoduché a intuitivní uživatelské rozhraní s lehce čitelnými texty a s efektivním rozložením elementů. Pro přehlednost byl redukován počet položek v nabídce hlavního menu na čtyři položky. Konkurenční aplikace Duolingo nebo Brainly obsahovaly položek více, což vedlo k horší přehlednosti v navigaci. Prototyp plně podporuje tmavý mód stejně jako Duolingo nebo Google Classroom. Součástí uživatelského rozhraní jsou i drobné animace při navigaci v aplikaci, které v aplikaci Brainly chyběly.

4.3.2 Kvalita obsahu

Prototyp dává uživatelům možnost vytvářet vlastní obsah jako u aplikace Google Classroom nebo Brainly. Tyto informace však nemusí být vždy aktuální a pravdivé. Výhodou aplikace LearnKit je systém pro odebírání obsahu od ověřených autorů, kteří by eventuálně byli podrobeni procesu schválení. Tím je uživateli dána možnost si vybrat, jaké informace chce konzumovat. Prototyp se nezaměřuje na jednu konkrétní tématiku jako Duolingo, které je určeno pouze k učení jazyků. Funguje jako obecný nástroj pro získávání nových znalostí.

4.3.3 Funkčnost

Aplikace prezentuje znalosti v mnoha formách jako kvízy, videa, nahrávky apod. s možností přidání více variant v budoucnu. Na rozdíl od aplikace Brainly, která používá jako zdroj informací pouze text, obsahuje LearnKit různé nástroje pro prezentaci obsahu.

4.3.4 Dostupnost

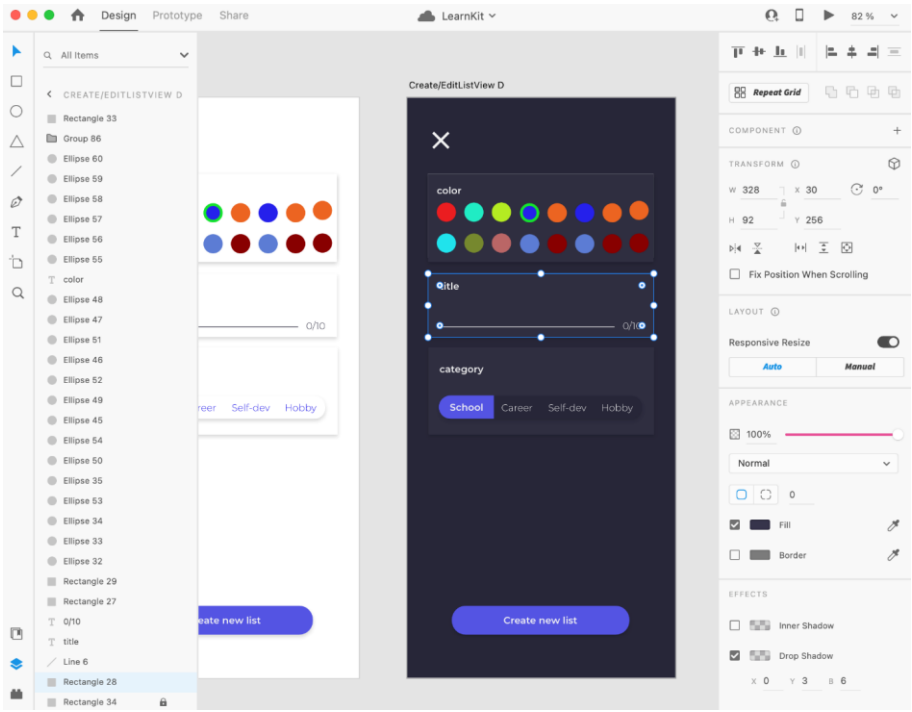
Dostupnost aplikace LearnKit je v této chvíli nízká. Je zatím implementována pouze pro platformu iOS. Kvůli časové náročnosti nebylo možné v jednom člověku vytvořit aplikaci i na platformu Android a na web. Budoucí iterace s touto podporou počítají.

4.3.5 Zpětná vazba a podpora

Podobně jako u testovaných konkurenčních aplikací obsahuje i prototyp možnost vyjádření zpětné vazby od uživatele. V sekci nastavení existuje možnost, která přesměruje uživatele do nativní aplikace Mail s předvyplněnou adresou pro kontaktování tvůrců aplikace.

5 NÁVRH UŽIVATELSKÉHO ROZHRANÍ

Pro tvorbu návrhu uživatelského rozhraní pro prototyp byl použit software Adobe Xd. Tvorba nového projektu je velice jednoduchá. V nabídce hlavního menu stačí vybrat „New file“, nebo vybrat z již existujících projektů. Aplikace dává možnost zvolit jednu z předdefinovaných šablon s konkrétní velikostí v pixelech, například pro iPhone 14 Pro Max. Součástí předplatného Adobe pro studenty je také cloudové uložiště o velikosti 100 GB. [67] Nový projekt s názvem „LearnKit“ byl uložen jako cloudový dokument, aby byl dostupný i na jiných zařízeních, které používají stejný Adobe účet. Samotné uživatelské rozhraní aplikace je velmi jednoduché a dělí se do čtyř hlavních částí.



Obrázek 9 Tvorba uživatelského rozhraní prototypu v aplikaci Adobe Xd

V levé části se nachází hlavní nástroje, které lze při tvorbě designu používat. Vpravo od této sekce je přehled všech vytvořených grafických elementů, například elipsy, obdélníky, čáry, texty nebo obrázky. V prostřední části je prostor pro samotnou editaci rozhraní pomocí nástrojů a pro vybrání aktuálně upravovaného elementu. Ve sloupci v pravé části aplikace lze nastavovat jednotlivé atributy vybraného elementu a měnit tak jeho vzhled.

5.1 Využití základních pravidel pro tvorbu grafického designu

Tvorba designu uživatelského rozhraní pro prototyp byla inspirována základními pravidly pro tvorbu grafického designu z knihy Graphic Design Rules: 365 Essential Dos and Don'ts. Příručka obsahuje mnoho pouček o vhodném výběru typografie, efektivním rozložení a designu, správném použití barev, obrázků apod. [68]

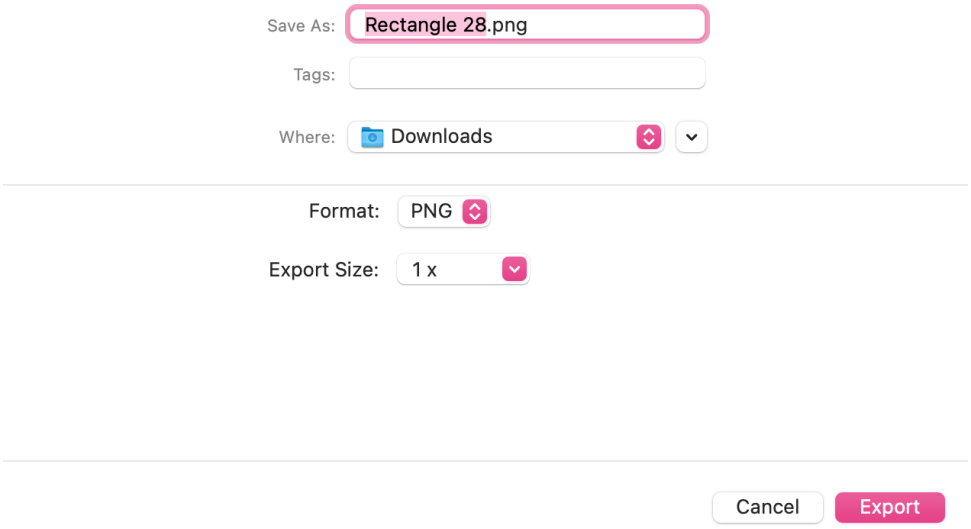
Obrázek 10 Ukázka stránky z knihy pravidel pro tvorbu grafického designu

Jedním z mnoha příkladů využití těchto pouček v návrhu je pravidlo o barvě fontu v malém odstavci textu. Pro tento typ není vhodné volit barvy typu azurová, purpurová nebo žlutá. Jsou příliš světlé a pro lidské oko je obtížnější oddělit jednotlivé znaky a spojit je ve slova a věty. Důsledkem je, že uživateli může trvat přečíst tento odstavec textu déle.

5.2 Export grafických komponent

V rámci aplikace existuje možnost pro exportovaní jednotlivých grafických komponent, které jsou součástí celkového designu. V prototypu byla tato funckionalita využita pro exportování ikon, které byly původně ve formátu SVG (Scalable Vector Graphics), do formátu PNG (Portable Network Graphics), se kterým je aktuálně práce v rámci frameworku

SwiftUI jednodušší. Vybranou grafickou komponentu lze exportovat na MacOS v nabídce „File-Export“, nebo pomocí klávesové zkratky „cmd + E“. Poté stačí vybrat, kam bude nový soubor uložen a zvolit jeden z podporovaných formátů.



Obrázek 11 Nastavení exportu v aplikaci Adobe Xd

6 IMPLEMENTACE PROTOTYPU

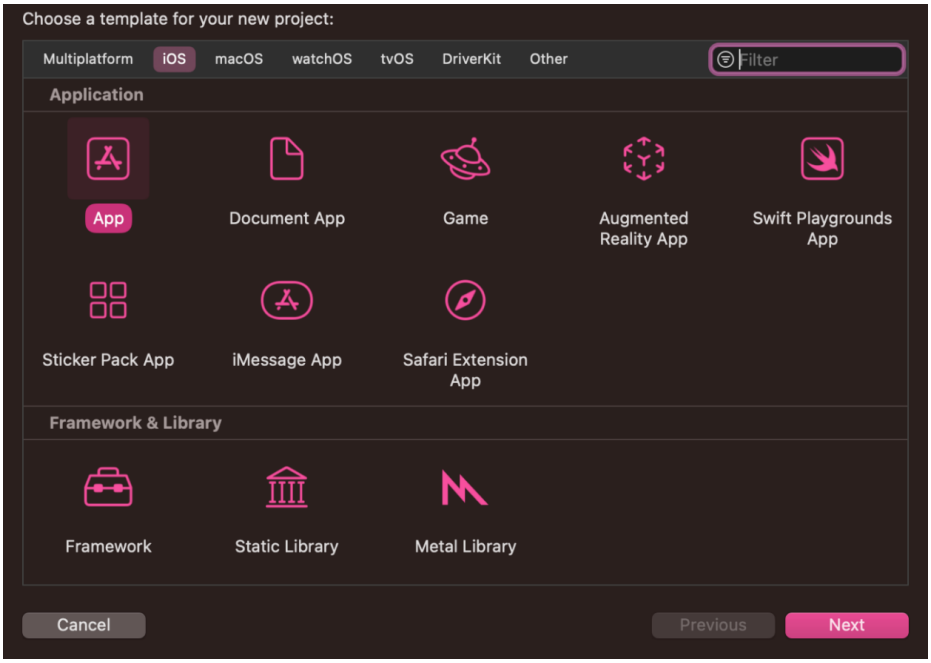
V této části jsou popsány nejdůležitější části při implementaci prototypu LearnKit na platformě iOS ve frameworku SwiftUI.

6.1 Tvorba a nastavení projektu

Pro tvorbu nového projektu v Xcode lze zvolit jednu ze tří variant – vytvořit nový projekt, klonovat existující projekt z Git repozitáře, nebo otevřít již existující projekt nebo soubor. Pro inicializaci projektu byla zvolena první možnost.

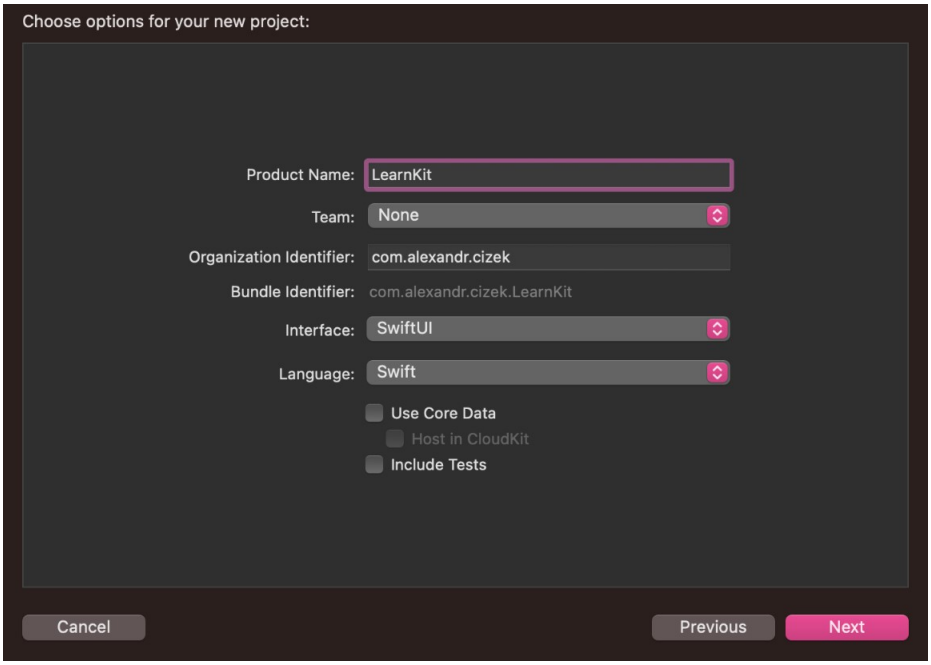
Obrázek 12 Úvodní obrazovka aplikace Xcode

Nyní je třeba zvolit platformu, pro kterou chceme aplikaci vyvíjet. Xcode nabízí také řadu šablon pro specifické využití, například pro vývoj her, rozšíření prohlížeče Safari, nebo aplikace s rozšířenou realitou. Při použití šablony jsou automaticky vygenerovány zdrojové soubory a naimportovány potřebné knihovny.



Obrázek 13 Volba platformy a šablony v aplikaci Xcode

Pro tvorbu prototypu je zvolena platforma iOS a nebyla použita žádná ze šablon – byla zvolena možnost „App“. V další fázi je třeba vyplnit název aplikace, jako rozhraní zvolit SwiftUI a pro jazyk vybrat Swift. V poslední řadě jsou zde dva checkboxy. Prvním z nich je „Use Core Data“. Core Data je standardizovaný framework od Apple vývojářů pro vytvoření lokální databáze a manipulaci s datovými entitami. Po vybrání této možnosti se framework automaticky naimportuje a vygeneruje se soubor pro definici entit a jejich vztahů mezi nimi. Poté druhý checkbox – „Include Tests“, naimportuje framework pro testování iOS aplikací „XCTest“ a vygeneruje ukázky definicí tříd pro unit a UI testy. Obě z těchto variant jdou aktivovat zpětně až po vytvoření projektu, takže na začátku lze tyto dvě možnosti ignorovat.



Obrázek 14 Nastavení projektu v aplikaci Xcode

6.1.1 Ikona aplikace

Velmi důležitou součástí aplikace je grafická podoba ikony, která mobilní aplikaci reprezentuje. Společně s názvem aplikace je ikona první věc, které si uživatel všimne při stahovaní z obchodu App Store. Je žádoucí, aby byla ikona na první pohled jasně rozeznatelná od ostatních a upoutala pozornost.

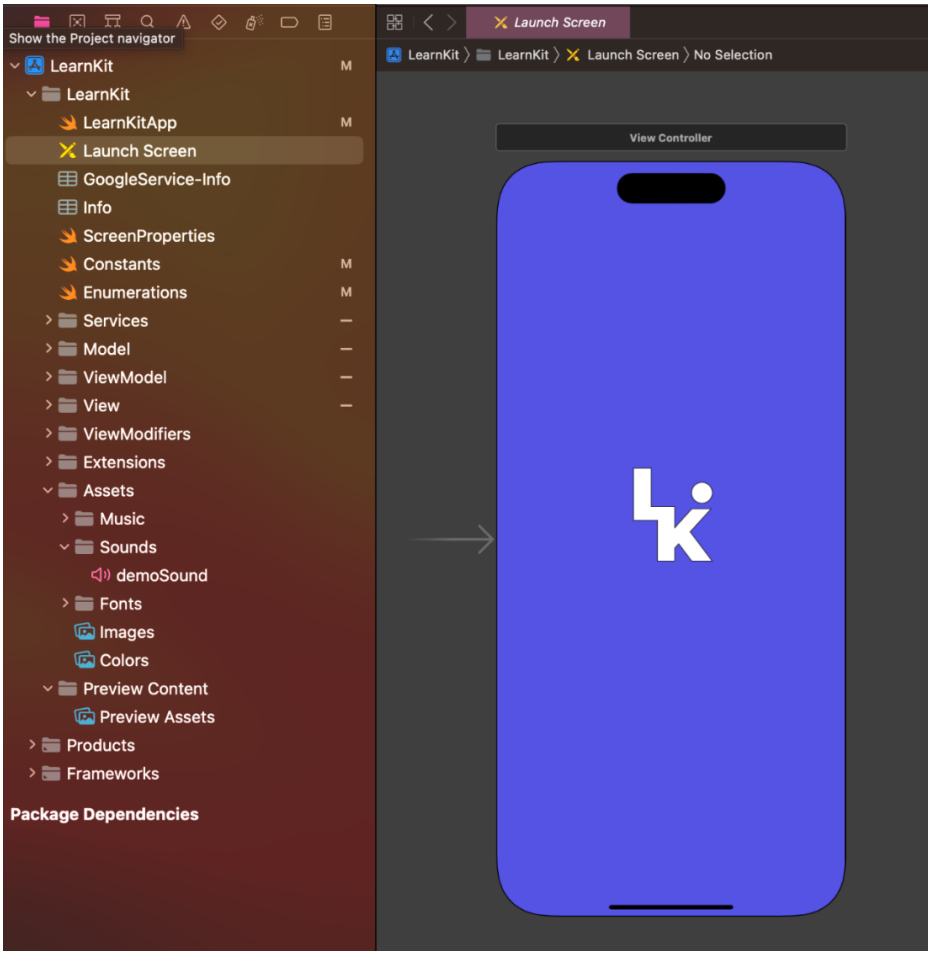
Obrázek 15 Ikona aplikace LearnKit

Při vytvoření nového projektu je vytvořena entita s názvem „AppIcon“, která je součástí souboru „Assets.xcassets“. Zde může vývojář nahrát obrázek pro ikonu aplikace. Grafická podoba ikony pro prototyp byla vytvořena v programu Adobe Illustrator. Defaultní položka

„AppIcon“ pro nastavení ikony aplikace se dá změnit v nastavení, konkrétně v „General“ v sekci „App Icons and Launch Screen“.

6.1.2 Spouštěcí obrazovka

Spouštěcí obrazovka (splash screen) je úvodní okno, které se zobrazí při startu aplikace. V již zmíněné sekci „App Icons and Launch Screen“ lze vybrat soubor, který bude použit pro spouštěcí obrazovku. Soubor musí být ve formátu „Storyboard“. Tento typ souboru se objevil v iOS od verze 5.0 pro vytváření uživatelského rozhraní ve frameworku UIKit s přístupem „drag and drop“. [69] Tímto způsobem lze definovat vzhled spouštěcí obrazovky i přes to, že zbytek aplikace používá SwiftUI. V prototypu je na spouštěcí obrazovce umístěno logo aplikace s modrým pozadím.



Obrázek 16 Spouštěcí obrazovka aplikace LearnKit

6.2 Struktura projektu

V následující části je předvedena základní struktura projektu LearnKit v Xcode. Popisuje roztřídění nejdůležitějších souborů a jejich význam pro prototyp.

6.2.1 Soubor main

Soubor s označením „@main“ obsahuje počáteční strukturu, která se inicializuje po spuštění SwiftUI aplikace. Vzhledem k tomu, že se jedná o strukturu, i tato má svůj konstruktor. V konstruktoru je volána metoda „FirebaseApp.Configure()“, která je zmíněna v dokumentaci Firebase jako nezbytná pro konfiguraci SDK. Dále jsou ve struktuře inicializovány všechny třídy, které je třeba vytvořit již při startu aplikace.

```
@main
struct LearnKitApp: App {

    // MARK: FIREBASE INIT
    init() {
        FirebaseApp.configure()
    }

    // MARK: GLOBAL VIEW MODELS
    @StateObject var authViewModel = AuthViewModel()
    @StateObject var navViewModel = MainNavigationViewModel()
    @StateObject var settingsViewModel = SettingsViewModel()
    @StateObject var networkService = NetworkService()
    @StateObject var mediaPlayerViewModel = MediaPlayerViewModel()

    // MARK: MAIN BODY
    var body: some Scene {
        WindowGroup {
            Group{
                if networkService.isConnected{
                    if authViewModel.userSession == nil{
                        SignInView()
                    } else{
                        MainView()
                    }
                } else {
                    NoConnectionView()
                }
            }
            .environmentObject(authViewModel)
            .environmentObject(navViewModel)
            .environmentObject(settingsViewModel)
            .environmentObject(mediaPlayerViewModel)
        }
    }
}
```

Obrázek 17 Soubor main v prototypu

Příkladem takové třídy je v prototypu „AuthViewModel“, který obsahuje přehled o stavu aktuálně přihlášeného uživatele. Tato informace je při startu aplikace klíčová. Protože na základě toho, zda je uživatel přihlášen, nebo ne, se mu zobrazí jiný obsah. Pokud přihlášen není, zobrazí se mu formulář pro přihlášení (případně registraci) a pokud přihlášen je, zobrazí se mu nabídka hlavního menu. Tato logika je uvedena v proměnné „body“ pomocí větve „if-else“. V poslední řadě je zde použita funkce „environmentObject“, která zajistí, že jakákoliv struktura v aplikaci má přístup k referenci na objekt, který je předán jako argument této funkce. Funkcionalita je použita pro hlavní třídy, které jsou v aplikaci frekventovaně používány na mnoha místech.

6.2.2 Enumerace

V případě, že v aplikaci existuje proměnná, jejíž možné hodnoty lze vyjádřit konečnou množinou stavů, je vhodné v jazyce Swift použít enumeraci. Příkladem využití je v prototypu enumerace s názvem „ListContentType“, která definuje výčet všech typů materiálů, které mohou být obsahem Listu, jako jsou kvízy, hlasové nahrávky apod. Enumeraci lze velmi efektivně využít v rozhodovací větvi algoritmu, kdy můžeme otestovat aktuální stav proměnné pomocí konstrukce „switch“. Užitečnou funkcionalitou je také iterování přes pole obsahující všechny stavy definované enumerace.

```
enum ListContentType: String, CaseIterable, Codable{
    case roadmap = "Roadmap"
    case quizzes = "Quizzes"
    case video = "Video"
    case voiceMemos = "Voice memos"
    case links = "Links"

    func getIconString() -> String{
        switch self{
        case .roadmap:
            return "flag.2.crossed"
        case .video:
            return "film"
        case .quizzes:
            return "questionmark"
        case .voiceMemos:
            return "mic"
        case .links:
            return "safari"
        }
    }
}
```

Obrázek 18 Enumerace pro výčet typů materiálu v seznamu

Na obrázku lze vidět, že pro definici enumerace v jazyce Swift je použito klíčové slovo „enum“ společně s volitelným názvem. Dále je zde prostor pro definovaní protokolů pro danou enumeraci. V ukázce je použit protokol „CaseIterable, který umožňuje zavolat na enumeraci metodu „allCases“, která vrací pole všech stavů dané enumerace. Tato funkcionalita je v prototypu použita pro tvorbu postranního menu uvnitř Listu. Uživatel si může vybrat, který druh materiálu chce vyhledat. Pomocí již zmíněné metody „allCases“ a dále struktury „ForEach“, která je součástí frameworku SwiftUI, lze generovat jednotlivé položky menu pomocí smyčky. Velkou výhodou zde je, že pokud upravíme počet, nebo popis stavů v definici enumerace „ListContentType“, pak bude změna automaticky reflektována i v menu. V našem případě obsahuje definice enumerace pět stavů (roadmap, quizzes, video, voiceMemos, links) – menu bude obsahovat také pět položek. V definici enumerace lze také vytvořit metody stejně jako ve struktuře nebo třídě. V ukázce je metoda s názvem „getIconString“, která vrací název ikony pro grafickou reprezentaci položky postranního menu. V metodě je demonstrován mechanismus rozhodovací větve s použitím enumerace a konstrukce switch. V enumeraci je dále použit protokol „Codable“, který je úzce spjat s mapováním objektů pro databázový systém.

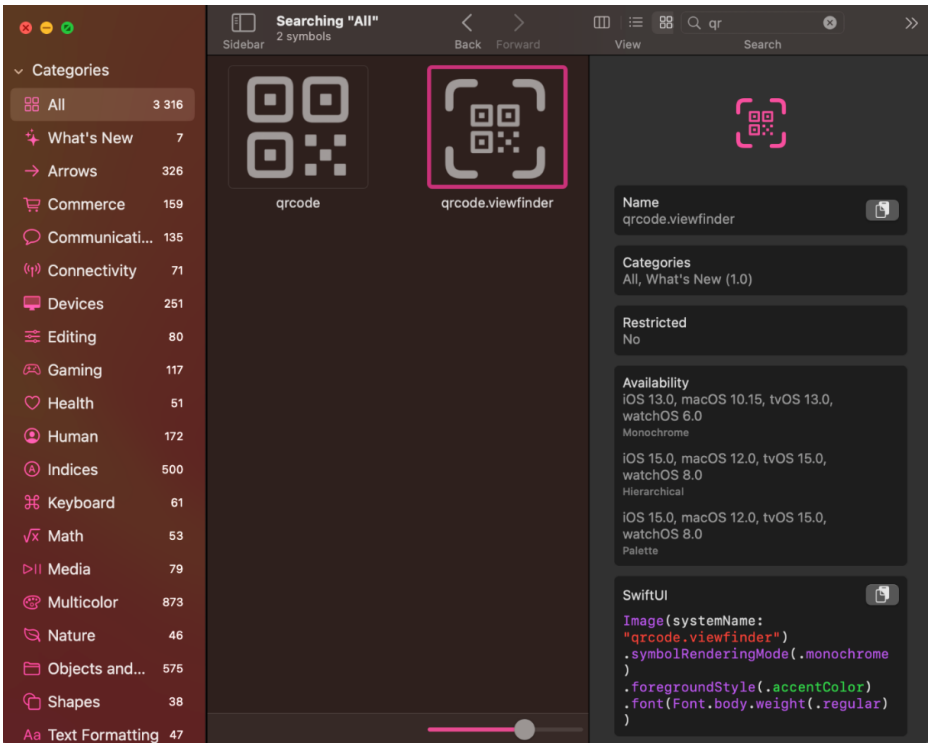
6.2.3 SF symboly

V rámci vývoje aplikace ve SwiftUI lze použít zabudovanou grafickou knihovnu s tzv. „SF symboly“. Zkratka SF označuje San Francisco, což je název pro aktuální systémový font na Apple platformách. SF symboly jsou grafické ilustrace ikon v různých barvách a velikostech. Knihovna obsahuje k dnešnímu datu přes 4400 různých ikon, které jsou roztříděny do několika kategorií. [70] Pro použití těchto zdrojů nemusí vývojář nic instalovat ani nastavovat. V prototypu jsou SF symboly využity pro tvorbu uživatelského rozhraní na několika místech. Příkladem je ikona, která pro uživatele indikuje možnost naskenování QR kódu v sekci vyhledávání obsahu.

```
Image(systemName: "qrcode.viewfinder")
    .resizable()
    .scaledToFit()
    .frame(width: 18)
    .foregroundColor(.offwhite)
```

Obrázek 19 Použití SF symbolu ve SwiftUI

Použití SF symbolů je velmi jednoduché. Je volán konstruktor struktury s názvem „Image“, která je součástí SwiftUI, s parametrem „systemName“. Unikátním identifikátorem každého SF symbolu je jeho název, který je třeba v konstruktoru použít ve formě datového typu „String“. Název SF symbolu uvedeného v ukázce je „qrcode.viewfinder“. Pro přehled dostupných SF symbolů a jejich názvů byla při vývoji použita aplikace San Fransymbols, kterou lze stáhnout z obchodu App Store zdarma. Aplikace nabízí přehledný seznam SF symbolů roztříděných do různých kategorií. Je zde dokonce i možnost si nakonfigurovat konkrétní vzhled vybraného symbolu, jako například barvu, pozadí, nebo tloušťku a poté zkopírovat vygenerovaný kód buď pro UIKit, nebo SwiftUI. Další užitečnou informací o SF symbolech je dostupnost pro konkrétní platformu a její verzi.



Obrázek 20 Aplikace San Fransymbols

6.2.4 Definice konstant

Existují-li data, která aplikace opakovaně využívá na více místech za stejným účelem, může být dedikovaný soubor pro definici globálních konstant velice efektivní. Důvodem je, že pokud je v budoucnu třeba jednu z uvedených hodnot změnit, stačí tak provést pouze na jednom místě, kde je definována. V jazyce Swift lze konstanty definovat mnoha způsoby. Například použitím struktury, třídy, enumerace nebo lokálního souboru. Pro definici konstant v prototypu je použita enumerace. Důvodem je, že z enumerace nelze vytvořit

instanci jako u struktury, nebo třídy. To znemožní vývojáři potenciálně vytvořit několik redundantních instancí, které by zbytečně spotřebovávali paměť. Enumerace lze i vnořovat a vytvořit tím přehlednější hierarchii dat. K jednotlivým hodnotám lze přistoupit přes název enumerace a statické konstanty definované v těle enumerace.

```
// definition
enum Constants{
    static let LIST_TITLE_MAX_CHARS: Int = 30
}

// usage
let constant = Constants.LIST_TITLE_MAX_CHARS
```

Obrázek 21 Příklad definice konstant

6.2.5 Model

Vrstva Modelu v prototypu obsahuje soubory s definicí datových entit vystupující v aplikaci v podobě struktur. Jde o datové modely, na které se budou mapovat data z databáze Firebase Firestore. Pro každou datovou entitu je v projektu vytvořen vlastní soubor pro zvýšení přehlednosti a všechny tyto soubory se nachází ve složce s názvem „Model“.

```
import FirebaseFirestoreSwift
import SwiftUI

struct UserModel: Codable{
    @DocumentID var id: String?
    var username: String
    var email: String
    var birthdate: Date
    var isPremium: Bool
    var likedAuthors: [String]
    var likedLists: [String]
}
```

Obrázek 22 Datová entita reprezentující uživatele

V ukázce je datový model „UserModel“ popisující jednotlivé atributy konkrétního uživatele v aplikaci. Každý datový model v prototypu obsahuje na prvním místě proměnnou „id“, která plní funkci unikátního identifikátoru objektu ve formě textového řetězce. Tato proměnná je jako jediná označena pomocí výrazu „@DocumentID“. Znakem „@“ se v jazyce Swift od verze 5 označuje tzv. „property wrapper“, volně přeloženo jako „obal

proměnné“. [71] Za tímto obalem se skrývá vrstva, která definuje, jak se proměnná ukládá, nebo jak je její hodnota vypočítávána při čtení. Používá se jako náhrada opakujícího se kódu pro „gettery“ a „settery“. Modul „FirebaseFirestoreSwift“ obsahuje právě definici pro obal s názvem „DocumentID“, který zajistí, že unikátní identifikátor dokumentu uloženého v databázi Firestore se namapuje do vybrané proměnné. Dále struktura obsahuje atributy jako přezdívku, e-mail, datum narození nebo pole identifikátorů oblíbených autorů uživatele. Poslední částí definice struktury je implementace protokolu s názvem „Codable“, která je nutná pro nastavení korektního mapování dat z databáze na entitu z vrstvy modelu. [72]

6.2.6 View model

Účelem této vrstvy prototypu je oddělení vzhledu a funkcionality ve vrstvě View. View model je třída, která implementuje protokol „ObservableObject“ [73] s členskými proměnnými označenými pomocí „@Published“. Změna hodnot těchto proměnných vyústí v automatickou aktualizaci vrstvy View. Třída obsahuje veřejné proměnné, ke kterým bude mít přístup View a privátní proměnné, které lze použít pouze v rámci bloku definice třídy. Dále je ve třídě prostor pro definice konstruktoru a destruktoru objektu. Poslední částí jsou její metody. Privátní metody by měly plnit účel pomocných operací v rámci interní logiky třídy. Veřejné metody ve vrstvě View modelu obsluhují operace vyvolané na základě interakce s uživatelským rozhraním aplikace. Typický příkladem takové interakce je stisk tlačítka. Obsluha této události je uskutečněna ve vrstvě View pomocí volání veřejné metody z třídy View modelu.

6.2.7 View

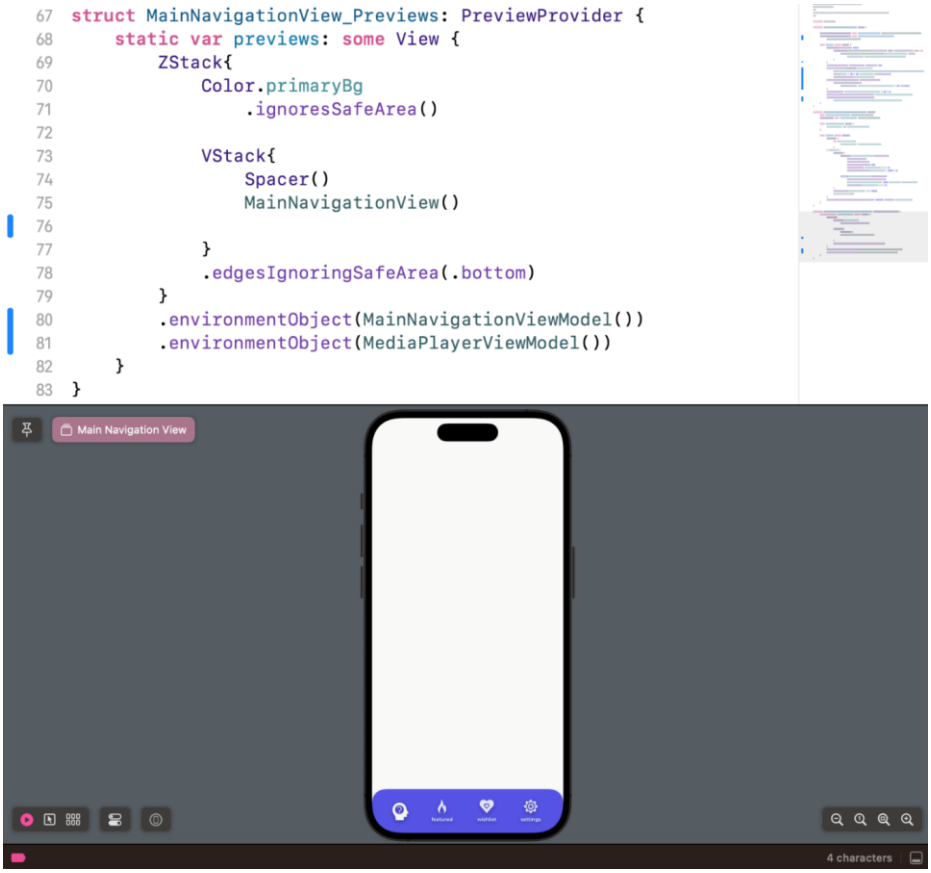
Každý soubor s definicí struktur, které implementují protokol „View“, jsou v projektu umístěny do stejnojmenné složky. Tyto struktury definují vzhled uživatelského rozhraní a jeho chování při interakci s uživatelem. Aby struktura implementovala protokol „View“, musí obsahovat proměnnou „body“. V rámci struktury lze definovat další proměnné a metody, které může „View“ používat. Tomu se ale v rámci MVVM architektury chceme vyhnout. Proměnné a metody, které bude komponenta používat obsahuje již zmiňovaný View model. Ve struktuře komponenty pouze inicializujeme objekt View modelu. Framework SwiftUI obsahuje knihovnu standardizovaných komponent, které lze při tvorbě vlastního uživatelského rozhraní použít. Tyto komponenty jsou taktéž struktury. Vývojář je inicializuje pomocí konstruktoru s parametry. Příkladem je struktura „Text“, které se do konstruktoru předá proměnná datového typu „String“ a tento textový řetězec zobrazí na

displeji. Pro vytvoření nového souboru z vrstvy View lze použít zkratku „cmd + N“ a ze sekce „User Interface“ zvolit „SwiftUI View“.

```
struct MyView: View {
    var body: some View {
        Text("Hello, World!")
    }
}

struct MyView_Previews: PreviewProvider {
    static var previews: some View {
        MyView()
    }
}
```

Obrázek 23 Příklad View ve SwiftUI

Vybraná možnost vygeneruje nový soubor se dvěma strukturami. První z nich je již zmíněná struktura implementující protokol „View“. Druhá struktura implementuje protokol „PreviewProvider“. Pomocí této struktury Xcode generuje náhled pro rychlé zobrazení změn v rámci vytvářeného View. V těle statické proměnné „previews“ lze definovat náhled pro otestovaní vzhledu a chování komponenty v různých podmínkách, například pro různé velikosti a orientace obrazovky, tmavý mód atd. Tyto struktury nejsou součástí produkčního vydání aplikace. Jsou určeny pouze pro testování komponent ve fázi vývoje.



Obrázek 24 Náhled View pro hlavní menu aplikace

Ukázka zobrazuje strukturu pro náhled komponenty hlavní navigace v prototypu. V rámci náhledu je použita barva pro hlavní pozadí a poté je menu posunuto do spodní části obrazovky, kde se bude při použití nacházet. Tímto způsobem je testován vzhled komponenty ještě předtím, než ji použijeme. Způsob použití náhledu je velmi individuální a záleží na vzhledu a funkcionalitě dané komponenty a při jakých podmínkách ji chceme testovat.

6.2.8 Property List

Jedním z automaticky vygenerovaných souborů při vytvoření nového projektu je soubor s názvem „Info.plist“. Soubor ve formátu XML (Extensible Markup Language) obsahuje prostor pro konfigurační data aplikace. [74] Data jsou ukládána pomocí párů „klíč-hodnota“. Příkladem informace v konfiguračním souboru je uvedení podporovaných orientací pro aplikaci pod klíčem s názvem „Supported interface orientations“.

6.3 Autentizace uživatelů

Následující sekce popisuje způsob autentizace uživatelů v prototypu pomocí funkcionalit ze služby Firebase Auth.

6.3.1 Přihlašování a registrace

V rámci prototypu je autentizace uživatelů řešena pomocí služby Firebase Auth, konkrétně s použitím e-mailu a hesla. Všechna logika, která se týká autentizace je v aplikaci součástí třídy „AuthViewModel“. Členské proměnné „userSession“ a „currentUser“ obsahují informace o právě přihlášeném uživateli. Při inicializaci objektu se nastaví aktivní uživatel a zavolá se metoda „getUserData“, která získá data o uživateli z databáze Firestore.

```
class AuthViewModel: ObservableObject{
    @Published var userSession: FirebaseAuth.User?
    @Published var currentUser: UserModel?

    init() {
        self.userSession = Auth.auth().currentUser
        getUserData()
    }
```

Obrázek 25 View model pro správu uživatelů

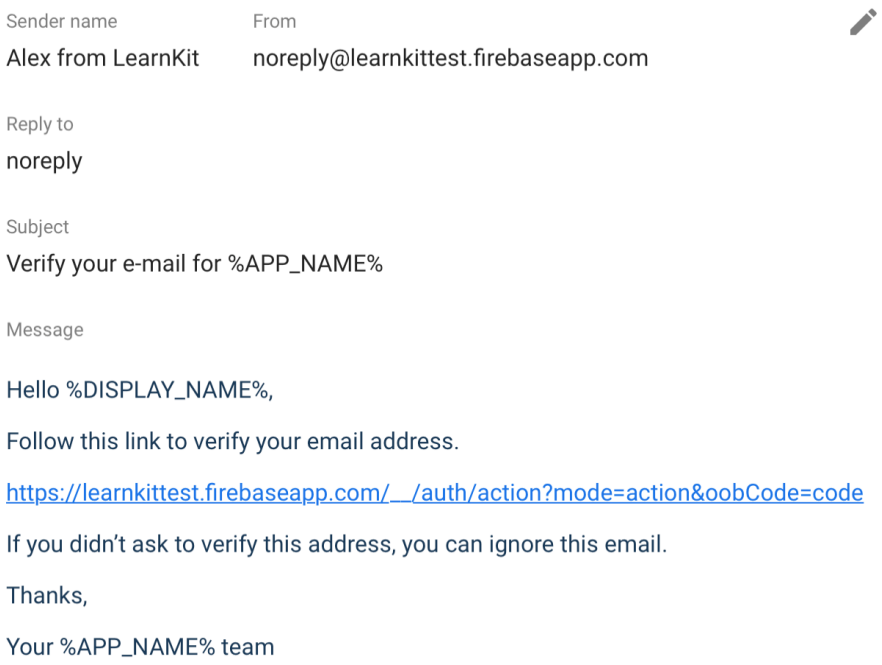
6.3.2 Ověřující e-mail

Služba Firebase Auth nabízí možnost odeslání e-mailu uživateli pro ověření jeho identity. V případě, že by aplikace tuto funckionalitu nepoužívala, pak by se mohl uživatel registrovat i s použitím e-mailové adresy, kterou nevlastní. Po tom, co se uživatel registruje do aplikace, tak mu systém zašle ověřující e-mail s vygenerovaným odkazem. Po kliknutí na odkaz provede uživatel aktivaci svého účtu. Firebase SDK obsahuje metodu s názvem „sendEmailVerification“, která automaticky odešle tento e-mail na adresu uživatele.

```
private func sendVerificationEmail(){
    guard let user = Auth.auth().currentUser else { return }
    user.sendEmailVerification { error in
        if let error = error{
            print(error.localizedDescription)
        }
    }
}
```

Obrázek 26 Metoda pro odeslání ověřujícího e-mailu

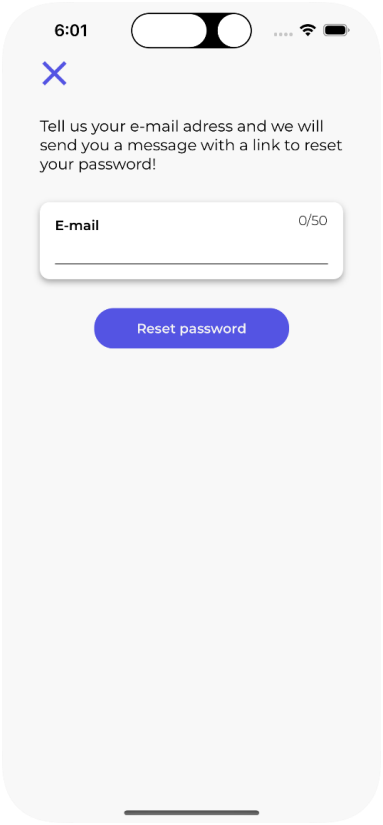
Nastavení vzhledu e-mailu lze nalézt v konzoli Firebase v sekci „Authentication“ pod kartou „Templates“, kde může vývojář nastavit jméno odesílatele, předmět a samotnou zprávu. V rámci textu zprávy lze použít proměnné definované službou Firebase, které jsou označené pomocí znaku „%“, například „%APP_NAME%“, ve které je uloženo jméno aplikace.



Obrázek 27 Šablona pro vzhled ověřujícího e-mailu v konzoli Firebase

6.3.3 Reset hesla

Aplikace dává možnost resetovat heslo k účtu. V případě, že uživatel zapomněl své aktuální heslo, může si v rámci aplikace zažádat o odeslání e-mailu s vygenerovaným odkazem pro reset hesla. Součástí formuláře pro přihlášení je v prototypu text „Forgot password?“. Po výběru této možnosti se zobrazí pohled pro reset hesla, ve kterém uživatel vyplní e-mail svého již existujícího účtu.



Obrázek 28 Obrazovka pro zaslání e-mailu pro reset hesla uživatele

V případě, že existuje účet s uvedenou adresou, zavolá se z Firebase SDK metoda „sendPasswordReset“, které se jako argument předá textový řetězec s e-mailovou adresu uživatele.

```
func sendPasswordResetEmail(withEmail email: String, completion:
    Auth.auth().sendPasswordReset(withEmail: email) { error in
        if let error = error{
            print(error.localizedDescription)
            completion(error)
            return
        }
        completion(nil)
    }
}
```

Obrázek 29 Metoda pro odeslání e-mailu pro reset hesla uživatele

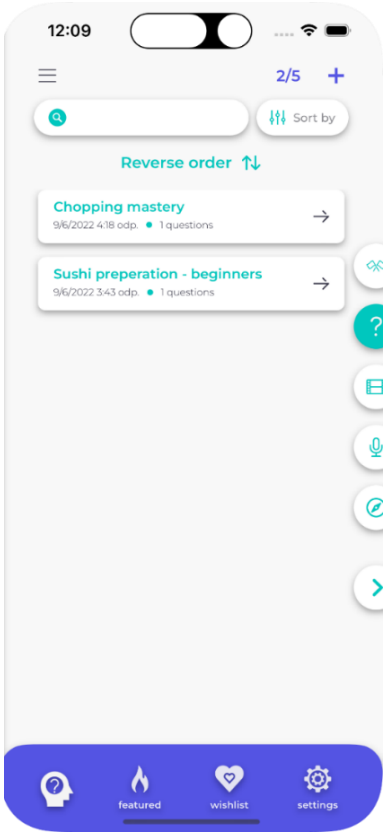
Nastavení šablony pro vzhled e-mailu lze opět nastavit v konzoli Firebase na stejném místě, jako u ověřujícího e-mailu. Po kliknutí na odkaz ve zprávě je uživatel přesměrován na webový formulář, ve kterém si může nastavit novou podobu svého hesla.

6.4 Hlavní menu

Základní částí uživatelského rozhraní prototypu je hlavní menu, které se nachází ve spodní části obrazovky. Hlavní menu obsahuje celkem čtyři položky. Jejich význam a vzhled je popsán v následující části.

6.4.1 Listy

Nejkomplexnější část aplikace se nachází v první položce hlavního menu s názvem „lists“. Zde uživatel manipuluje se svými Listy a všemi výukovými materiály. Základním pohledem je aktuálně vybraný List a postranní menu v pravé části obrazovky, ve kterém uživatel přepíná mezi jednotlivými druhy materiálů. Ikonou v levé horní části obrazovky lze zobrazit levé postranní menu, ve kterém může uživatel přepnout aktuální List, nebo vytvořit nový.



Obrázek 30 Obrazovka pro první položku hlavního menu

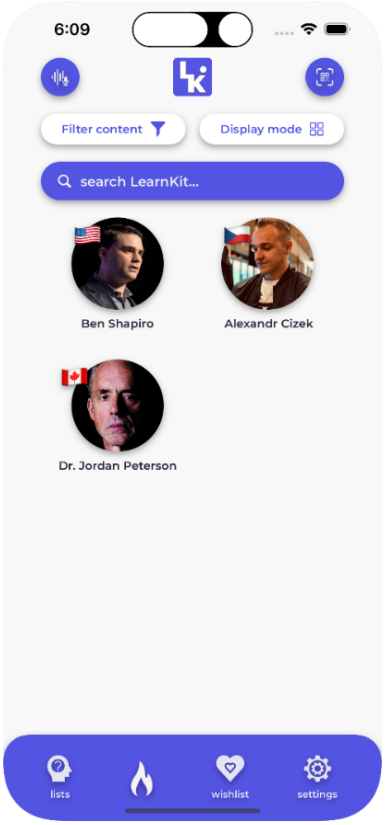
Naposledy aktivní List je ukládán ve formě jeho unikátního identifikátoru do persistentní paměti zařízení. Po opětovném spuštění aplikace zobrazí List, který byl naposledy aktivní.

```
@AppStorage("lastActiveList") var lastActiveListID = ""
```

Obrázek 31 Proměnná pro uložení identifikátoru posledního aktivního seznamu

6.4.2 Vyhledávání

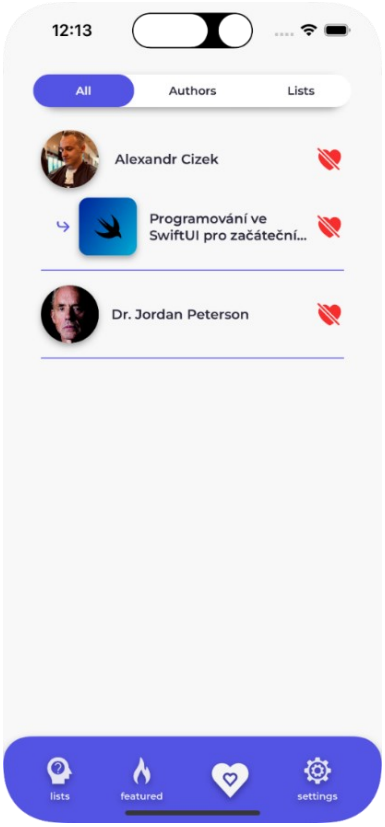
Druhá sekce hlavního menu je určena k vyhledávání obsahu od ověřených autorů. Uživatel může vyhledávat konkrétní autory nebo Listy pomocí naskenování QR kódu, nebo vyplnění informací pro filtraci obsahu. Uživatel odebírá konkrétní Listy, které může používat v první sekci hlavního menu.



Obrázek 32 Obrazovka pro druhou položku hlavního menu

6.4.3 Seznam oblíbených

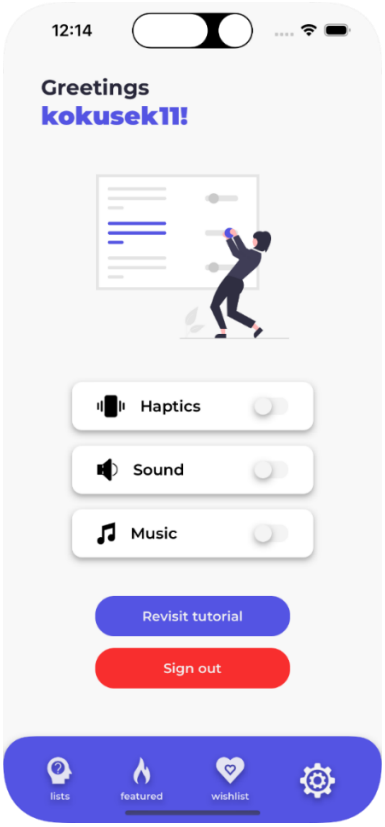
Funkcí seznamu je uložení konkrétních autorů a Listů, o které má uživatel zájem, ale zatím jejich obsah neodebírá. Důvodem je, že v budoucnu by mohly být Listy zpoplatněny. Uživatel si tak může dočasně uložit obsah, u kterého se rozhoduje, zda má zájem si ho pořídit.



Obrázek 33 Obrazovka pro třetí položku hlavního menu

6.4.4 Nastavení

V poslední sekci hlavního menu má uživatel možnost si upravit nastavení aplikace v rámci personifikace. Je zde možnost povolit přehrávání zvuku, muziky a použití haptické odezvy. Dále se zde uživatel může odhlásit ze svého účtu.



Obrázek 34 Obrazovka pro čtvrtou položku hlavního menu

6.5 Využití QR kódů

Funkcí QR (Quick Response) kódů v prototypu je jednoduché a intuitivní vyhledávání obsahu pro uživatele. V aplikaci figurují dvě základní datové entity, které lze vyhledat. Jde o ověřené autory a jejich Listy. Každý autor nebo List je spjat s unikátním identifikátorem v databázi. Informace o identifikátoru a typu datové entity je zakódována do QR kódu.

6.5.1 Generování

Každý uživatel v aplikaci má možnost vygenerovat QR kód pro konkrétního ověřeného autora, nebo List. Tento kód ve formě obrázku může uživatel sdílet buď v elektronické formě, nebo ho vytisknout a umístit na veřejné místo, například do školní, přednáškové nebo zasedací místnosti.

```
func generateQRCode(from string: String) -> UIImage?{
    let context = CIContext()
    let filter = CIFilter.qrCodeGenerator()
    filter.message = Data(string.utf8)

    if let outputImage = filter.outputImage?.transformed(by: CG
        if let cgimg = context.createCGImage(outputImage, from:
            print("DEBUG: generated QR for string: " + string)
            return UIImage(cgImage: cgimg)
        }
    }
    return nil
}
```

Obrázek 35 Metoda pro generování QR kódů v prototypu

Generování QR kódu je v prototypu realizováno metodou „generateQRCode“, která obsahuje parametr pro textový řetězec typu „String“, který má být do QR kódu zakódován. Metoda používá funkcionality nativní knihovny Core Image, která obsahuje různé filtry pro práci s obrázky. [75] Výstupem metody je v případě úspěšného vygenerování obrázek ve formě datového typu „UIImage“. Struktura vstupního textového řetězce je v následujícím tvaru: „datova_entita:unikatni_identifikator“. Datovou entitou může být buď klíčové slovo „list“, nebo „author“.

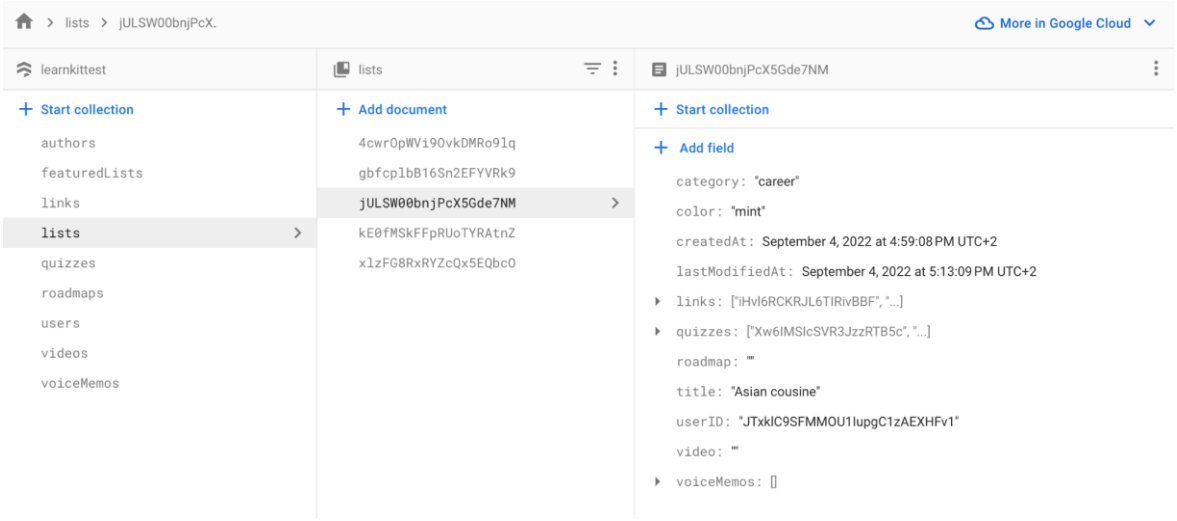
Obrázek 36 Příklad vygenerovaného QR kódu

6.5.2 Skenování

Prototyp používá pro skenování QR kódu knihovnu třetí strany s názvem „CodeScanner“ z platformy GitHub. [76] Knihovna obsahuje komponentu „CodeScannerView“, která je plně kompatibilní s frameworkem SwiftUI. Po naskenování QR kódu uživatelem přejde aplikace automaticky na novou obrazovku s hledaným autorem, nebo Listem.

6.6 Databáze Firebase Firestore

Základní jednotkou pro ukládání dat v databázi Firebase Firestore je dokument. Dokument je záznam, který obsahuje data. Tato data jsou v dokumentu uložena pomocí párů „klíč-hodnota“ (key-value pairs). Klíč pro daný atribut v dokumentu je vždy reprezentován textovým řetězcem. Pro samotné hodnoty atributů podporuje Firestore různé datové typy, například „boolean“, „number“, „string“, „timestamp“, „list“, „map“ a další. Maximální velikost pro jeden dokument je 1 MiB. [77]



Obrázek 37 Příklad dokumentu v databázi Firestore

Na obrázku je příklad struktury dokumentu, která v aplikaci reprezentuje data o jednom konkrétním Listu. Každý dokument obsahuje vlastní unikátní identifikátor. V prototypu je při tvorbě nového dokumentu generován řetězec jako unikátní identifikátor, který je použit pro název dokumentu. Jednotlivé řádky v dokumentu obsahují název klíče a spjatou hodnotu. Pro List se v prototypu ukládá jeho název – klíč „title“, nebo datum vytvoření – klíč „createdAt“. Každý dokument musí být součástí tzv. „kolekce“. Jde o kontejner pro ukládání více dokumentů. Kolekci není třeba vytvářet ani mazat. V momentě vytvoření prvního dokumentu vznikne požadovaná kolekce automaticky a případě, že je poslední dokument

v kolekci vymazán, zanikne i celá kolekce. Na obrázku jsou v levém sloupci uvedeny názvy jednotlivých kolekcí. V prostřední části poté unikátní identifikátory jednotlivých dokumentů v kolekci a na pravé straně konkrétní data jednoho vybraného dokumentu.

6.6.1 Mapovaní objektů

Všechny struktury v prototypu ve vrstvě modelu implementují protokol „Codable“, který je v jazyce Swift k dispozici od verze 4. Tento protokol usnadňuje proces mapovaní objektů z databáze Firestore na datové typy jazyka Swift a obráceně. Přečtený dokument z databáze je ve formátu slovníku. Při procesu mapování slovníku na objekt z vrstvy modelu může vývojář udělat několik chyb, například špatně uvést textový řetězec, který reprezentuje klíč pro danou položku ve slovníku. S použitím protokolu „Codable“ je toto riziko eliminováno. Při mapování musí všechny názvy a datové typy členských proměnných struktury z vrstvy Modelu odpovídat názvům a datovým typům proměnných v dokumentu. V případě, že toto neplatí, tak metoda pro mapování s názvem „data“ (viz. kapitola CRUD operace) vyhodí výjimku. Pokud chce vývojář použít názvy členských proměnných ve struktuře Modelu, které nekorespondují s názvy proměnných v dokumentu, musí v těle definice uvést enumeraci s názvem „CodingKeys“. V enumeraci lze pro členské proměnné Modelu uvést alternativní textový řetězec, který bude použit jako klíč pro získání hodnoty ze slovníku. [78]

```
struct ExampleModel: Codable{
    var normalizedTitle: String
    var description: String

    private enum CodingKeys: String, CodingKey {
        case normalizedTitle = "normalized_title"
        case description
    }
}
```

Obrázek 38 Příklad použití protokolu Codable

V ukázce je definice struktury z vrstvy modelu s použitím enumerace „CodingKeys“. Model má dvě členské proměnné – „normalizedTitle“ a „description“. V enumeraci je pro první z nich uveden alternativní řetězec pro mapování s hodnotou „normalized_title“. Dokument v databázi by v tomto případě obsahoval hodnotu s klíčem „normalized_title“, která by byla mapována do členské proměnné „normalizedTitle“.

6.6.2 Tvorba reference

Každý dokument v databázi Firestore je identifikován svou unikátní lokací v databázi. Pro přístup k této lokaci je nutno vytvořit tzv. „referenci“. Prvním krokem je tvorba základní reference pro přístup k celé databázi. To lze docílit příkazem „Firestore.firestore()“. Na tento objekt lze použít metody „collection“ a „document“ pro definování reference na konkrétní lokaci do databáze. Po tvorbě reference lze zavolat konkrétní metody pro manipulaci dat na požadované lokaci v databázi.

6.6.3 CRUD operace

Akronymem CRUD se anglicky označují čtyři základní operace pro manipulaci s persistentními daty v aplikaci. Těmito operacemi jsou vytvoření (create), čtení (read), aktualizace (update) a mazání (delete). [79] Modul „FirebaseFirestore“ z Firebase SDK obsahuje různé varianty metod pro CRUD operace. Všechny tyto metody jsou asynchronní. [80] V jazyce Swift je pro výsledek asynchronní operace použit tzv. „blok dokončení“ (completion block), který je volán po zpracování výsledku operace. V následující části jsou ukázány a vysvětleny příklady implementace zmiňovaných čtyř typů operací v prototypu. Pro vytvoření dokumentu v databázi lze použít buď metodu „addDocument“, nebo „setData“. První z nich vytvoří nový dokument s automaticky vygenerovaným identifikátorem. Naopak při použití druhé metody má vývojář možnost si identifikátor pro dokument definovat sám.

```
func createList(completion: @escaping (ListModel?) -> Void){
    guard let user = Auth.auth().currentUser else { return }
    let ref = FirestoreService.shared.db.collection("lists").document()
    let listID = ref.documentID
    let now = Date()
    let data: [String:Any] = [
        "category": self.category.rawValue,
        "color": self.color.rawValue,
        "createdAt": now,
        "lastModifiedAt": now,
        "quizzes": [],
        "links": [],
        "title": self.title,
        "userID": user.uid,
        "voiceMemos": [],
        "video": "",
        "roadmap": ""
    ]
    ref.setData(data) { error in
        if let error = error{
            print(error.localizedDescription)
            completion(nil)
            return
        }
        completion(ListModel(id: listID, title: self.title, category: self
            createdAt: now, lastModifiedAt: now, quizzes: [], links: [], v
    }
}
```

Obrázek 39 Metoda pro vytvoření nového Listu v databázi

Uvedený kód popisuje vytvoření nového dokumentu, který reprezentuje data o nově přidaném Listu od uživatele. Prvním krokem je vytvoření reference pro nový dokument v kolekci nazvané „lists“, která se uloží do konstanty „ref“. Definice metody „setData“ obsahuje argument pro data, ze kterých je nový dokument vytvořen. Tato data jsou datového typu „[String:Any]“. Jedná se o datovou strukturu slovníku s klíči, které jsou datového typu „String“ a hodnotami typu „Any“. Uvedené klíče budou odpovídat štítkům pro jednotlivé položky v dokumentu, proto musí být typu „String“. Naopak „Any“ vyjadřuje, že datový typ ukládané informace do dokumentu může být libovolný. Slovník je v ukázce uložen do konstanty „data“, která je použita ve volání metody „setData“. V bloku dokončení metody má vývojář přístup k proměnné typu „Error?“. Otazníkem jsou v jazyce Swift označeny takzvané „volitelné“ proměnné (optionals). Označení znamená, že hodnota této proměnné může být buď „nil“, nebo jakákoliv konkrétní hodnota uvedeného datového typu, v tomto případě „Error“. V kontextu metod z Firebase tato proměnná označuje, zda se vyskytla chyba při vykonávání dané operace. Hodnota proměnné bude „nil“, pokud byla operace

vytvoření nového dokumentu úspěšně dokončena. V opačném případě bude obsahovat konkrétní hodnotu datového typu „Error“. Proces, při kterém programátor zjišťuje, zda je hodnota volitelné proměnné „nil“, nebo ne, se nazývá „unwrapping optionals“, volně přeloženo jako „rozbalení volitelných položek“. V ukázce je toto provedeno pomocí konstrukce „if let“, při které zároveň vytváříme novou stejnojmennou konstantu „error“ a vyhodnocujeme, zda se vykoná podmíněný blok. Pokud není hodnota původní proměnné „error“ „nil“, pak bude výsledek podmínky „true“ a přejde se do podmíněného bloku. V rámci něj už je přístup k nové konstantě „error“, která je po rozbalení datového typu „Error“, nikoliv „Error?“. V bloku má programátor prostor pro reakci na chybu dané operace, jako například zobrazení chybové zprávy uživateli. Všechny metody pro CRUD operace ve Firestore mají tuto proměnnou v bloku dokončení k dispozici. „Error“ je enumerace, která obsahuje proměnnou „localizedDescription“. Jedná se o textový řetězec, který popisuje konkrétní chybu, která nastala.

```
func getAuthors(){
    FirestoreService.shared.db.collection("authors").getDocuments { snapshot
        if let error = error{
            print(error.localizedDescription)
            return
        }
        guard let snapshot = snapshot else { return }
        guard snapshot.documents.count > 0 else { return }
        for document in snapshot.documents{
            do{
                let author = try document.data(as: AuthorModel.self)
                DispatchQueue.main.async {
                    self.authors.append(author)
                }
            } catch{
                print(error.localizedDescription)
            }
        }
    }
}
```

Obrázek 40 Metoda pro získání ověřených autorů z databáze

Operace čtení dat je ve srovnání s ostatními CRUD operacemi nejkomplikovanější. Je zde k dispozici velké množství metod, například „getDocument“ pro čtení jednoho dokumentu nebo „getDocuments“ pro čtení více dokumentů v kolekci. V ukázce je vytvořena reference na kolekci s názvem „authors“. Metoda „getDocuments“ má v bloku dokončení nejen proměnnou typu „Error?“, ale také druhou typu „QuerySnaspshot?“, což je struktura, která obsahuje mimo jiné i přečtené dokumenty. Pomocí konstrukce „guard let“ se zkontroluje,

zda je hodnota proměnné „snapshot“ „nil“, či nikoliv. Objekt „snapshot“ obsahuje dále proměnnou „documents“, což je pole jednotlivých dokumentů typu „QueryDocumentSnapshot“. Dále se zkontroluje, zda toto pole není prázdné. Pokud dokumenty obsahuje, tak jsou procházeny pomocí cyklu „for“ a každý dokument je nejprve namapován pomocí metody „data“ na požadovanou strukturu z vrstvy Modelu, v tomto případě na „AuthorModel“. Mapování na objekt pomocí metody „data“ může selhat, proto se musí nacházet v konstrukci „do-catch“. V momentě, kdy by názvy proměnných a jejich datové typy v definici modelu „AuthorModel“ přesně nekorespondovaly se skutečnou podobou uložených dat v dokumentu, nastala by výjimka a v metodě by se přešlo do bloku „catch“. Pokud proběhne proces mapování bez chyby, pak má vývojář k dispozici konkrétní instanci datové struktury, kterou lze například uložit do pole, jako je učiněno v ukázce metodou „append“.

```
func editQuiz(completion: @escaping (QuizModel?) -> Void){
    guard Auth.auth().currentUser?.uid != nil else { return }
    guard let quizID = self.quizID else { return }

    // update quiz document data - title, questionCount, lastModifiedAt
    let now = Date()
    let quizData: [String:Any] = [
        "title": self.title,
        "questionCount": self.questionCount,
        "lastModifiedAt": now
    ]
    let ref = FirestoreService.shared.db.collection("quizzes").document(quizID)
    ref.updateData(quizData) { error in
        if let error = error{
            print(error.localizedDescription)
            completion(nil)
            return
        }
```

Obrázek 41 Metoda pro úpravu dat o kvízu v databázi

Pro aktualizaci dat v dokumentu existuje metoda „updateData“, která je svou definicí téměř totožná s metodou pro vytvoření dokumentu. Jediný rozdíl je, že reference na dokument, který chceme aktualizovat, musí v databázi již existovat. Pro vytvoření takové reference se využívá unikátní identifikátor, který je každému nově vytvořenému dokumentu přidělen. V ukázce je tento identifikátor uložen do proměnné „quizID“.

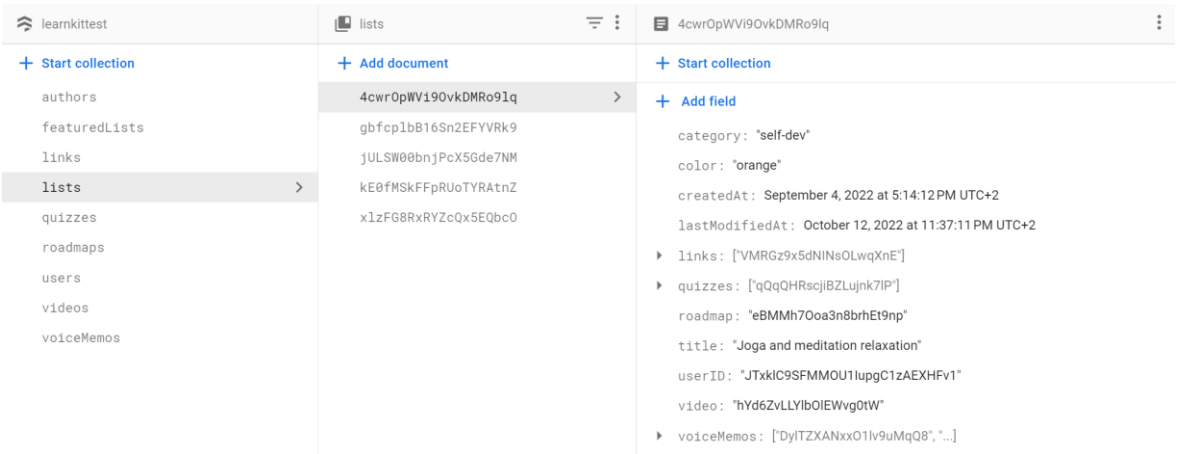
```
func deleteList(list: ListModel){
    guard let listID = list.id else { return }
    FirestoreService.shared.db.collection("lists").document(listID).delete {
        if let error = error{
            print(error.localizedDescription)
            return
        }
```

Obrázek 42 Metoda pro smazání seznamu z databáze

Smazání dokumentu je ze všech zmíněných operací nejpřímočařejší. Existuje pro ni metoda s názvem „delete“, která neobsahuje žádné argumenty. Nejprve je vytvořena reference na dokument, který je třeba smazat. Podobně jako u metody pro aktualizaci je použit unikátní identifikátor dokumentu. V bloku dokončení je opět k dispozici proměnná pro reakci na případnou chybu při vykonávání operace. V ukázce je zobrazena metoda pro smazaní dokumentu, který reprezentuje konkrétní List v aplikaci.

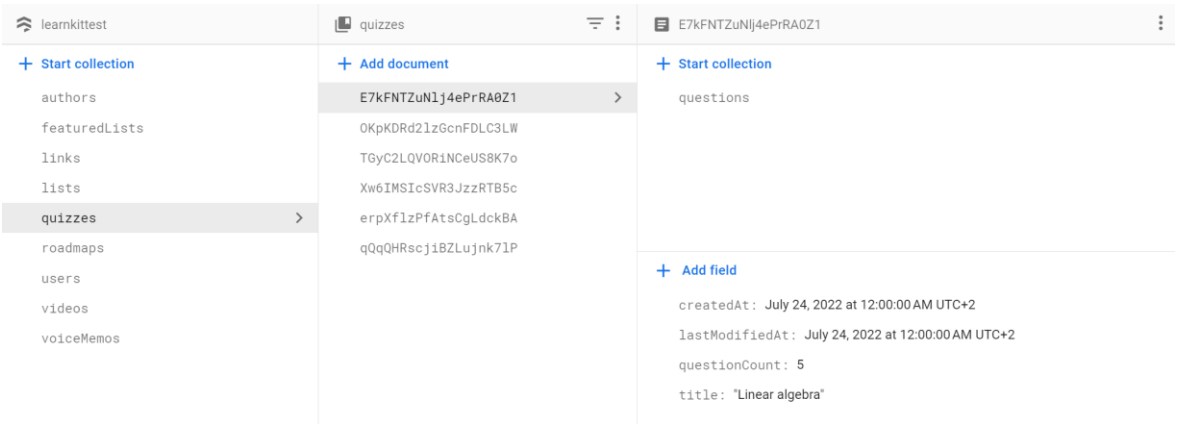
6.6.4 Ukázky datových modelů v prototypu

Základní datovou entitou v prototypu je List, který obsahuje data o vytvořeném seznamu včetně odkazů na jednotlivé materiály. Všechny dokumenty reprezentující Listy jsou v databázi uloženy v kolekci s názvem „lists“. Každý List obsahuje data jako název, kategorii, asociovanou barvou nebo datum vytvoření. Dále také položku „userID“, která udává unikátní identifikátor uživatele, který List vlastní. Poté jsou zde data, které plní funkci odkazů na jednotlivé materiály.



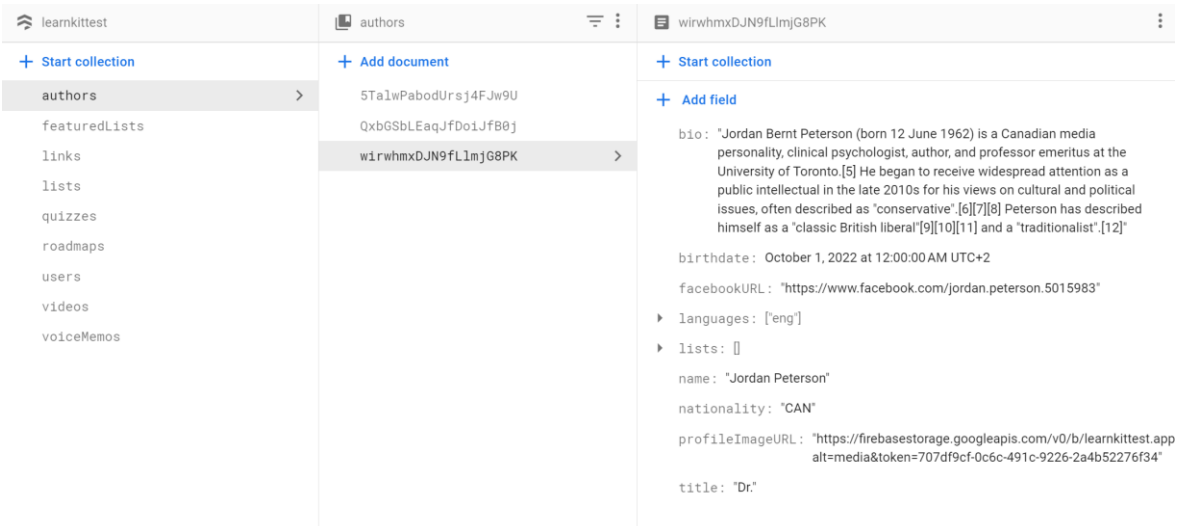
Obrázek 43 Kolekce lists v databázi Firestore

Příkladem je atribut „quizzes“, což je pole datového typu „String“. Každý textový řetězec v tomto poli je jeden unikátní identifikátor dokumentu, který reprezentuje kvíz. Tyto dokumenty jsou uloženy v oddělené kolekci „quizzes“. Podobným způsobem jsou uloženy odkazy na ostatní typy materiálů v Listu. Každý dokument reprezentující kvíz obsahuje data jako název, počet otázek apod. Mimo to je zde subkolekce s názvem „questions“, ve které jsou dokumenty, které reprezentují jednotlivé otázky pro daný kvíz.



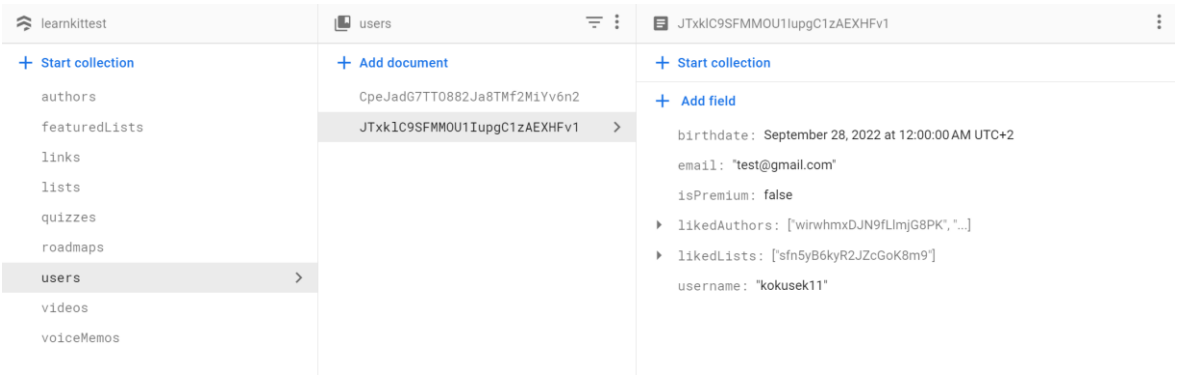
Obrázek 44 Kolekce quizzes v databázi Firestore

Další důležitou entitou v aplikaci je autor. Tyto dokumenty jsou v kolekci „authors“ a každý z nich reprezentuje jednoho konkrétní ověřeného autora v systému. Všichni autoři mají atribut „lists“. Jde opět o pole unikátních identifikátorů, tentokrát pro Listy, které autoři publikovali. Tyto Listy jsou součástí kolekce „featuredLists“.



Obrázek 45 Kolekce authors v databázi Firestore

Pro ukládání dat o jednotlivých uživatelích existuje kolekce „users“. Jednotlivé dokumenty obsahují informace jako uživatelské jméno, datum narození nebo odkaz na oblíbené autory a Listy.



Obrázek 46 Kolekce users v databázi Firestore

6.7 Multimédia aplikace

Všechny lokální multimediální zdroje, které aplikace používá, jsou v projektu ve složce s názvem „Assets“. Je to prostor pro uložení souborů pro video, audio, fonty, barvy, obrázky, loga, ilustrace a další. Pro ukládání obrázků a barev je při vývoji Swift aplikace použit speciální formát souboru Asset Catalog s příponou „xcassets“, který využívá SwiftUI v rámci mnoha funkcionalit.

6.7.1 Audio

Audio soubory jsou v prototypu použity pro přehrávání muziky a zvuků. Tyto soubory ve formátu „mp3“ se nachází ve složkách s názvy „Sounds“ a „Music“.

```
func playSound(name: String, type: String){
    guard let path = Bundle.main.path(forResource: name, ofType: type) else { return }
    let url = URL(fileURLWithPath: path)

    do{
        player = try AVAudioPlayer(contentsOf: url)
        player?.play()
    } catch {
        print(error.localizedDescription)
    }
}
```

Obrázek 47 Metoda pro přehrání audio souboru v prototypu

Metoda s názvem „playSound“ obsahuje dva parametry. Prvním z nich je název přehrávaného zvukového souboru a poté jeho typ. Metoda nejdříve použije oba argumenty ke konstrukci cesty k požadovanému souboru a poté ji převede na URL (uniform resource locator). Posledním krokem pro přehrání zvukového souboru je vytvoření instance třídy „AVAudioPlayer“, které se do konstruktoru předá již zmiňovaná URL. Definice třídy je součástí nativní knihovny AVFAudio. [81] Přehrání zvuku je realizováno pomocí volání metody „play“.

6.7.2 Fonty

Pokud vývojář iOS aplikace nechce použít defaultní font San Francisco, musí si naimportovat vlastní soubory s fonty. V prototypu se vyskytuje font s názvem „Montserrat“, který byl stažen ze služby Google Fonts. Font je pod volnou licencí a lze ho bezplatně používat i v komerčních produktech. [82] Soubory ve formátu „TrueType font“ (TTF), jsou v projektu uloženy ve složce s názvem „Fonts“ a podsložce „Montserrat“, kde se nachází více variant fontu v různých variantách. Aby se nový font dal aplikovat ve frameworku SwiftUI, je nutné upravit konfigurační soubor „Info.plist“. Zde je třeba přidat novou položku s klíčem „Fonts provided by application“. Jde o pole textových řetězců, kde každý řetězec obsahuje název jednoho souboru s fontem včetně koncovky.

Key	Type	Value
Information Property List	Dictionary	(1 item)
Fonts provided by application	Array	(8 items)
Item 0	String	Montserrat-Black.ttf
Item 1	String	Montserrat-ExtraBold.ttf
Item 2	String	Montserrat-Bold.ttf
Item 3	String	Montserrat-SemiBold.ttf
Item 4	String	Montserrat-Medium.ttf
Item 5	String	Montserrat-Regular.ttf
Item 6	String	Montserrat-Light.ttf
Item 7	String	Montserrat-Thin.ttf

Obrázek 48 Seznam fontů v souboru Info.plist

Posledním krokem je samotná aplikace fontu na komponentu pomocí metody „font“ ve, které se uvede textový řetězec reprezentující název fontu:

```
extension View{
    func fontMontserrat(size: CGFloat, weight: FontWeight = .regular) -> some View{
        self
            .font(.custom("Montserrat-" + weight.rawValue, size: size))
    }
}

enum FontWeight: String{
    case black = "Black"
    case extraBold = "ExtraBold"
    case bold = "Bold"
    case semiBold = "SemiBold"
    case medium = "Medium"
    case regular = "Regular"
    case light = "Light"
    case thin = "Thin"
}
```

Obrázek 49 Metoda a enumerace pro použití fontu Montserrat v prototypu

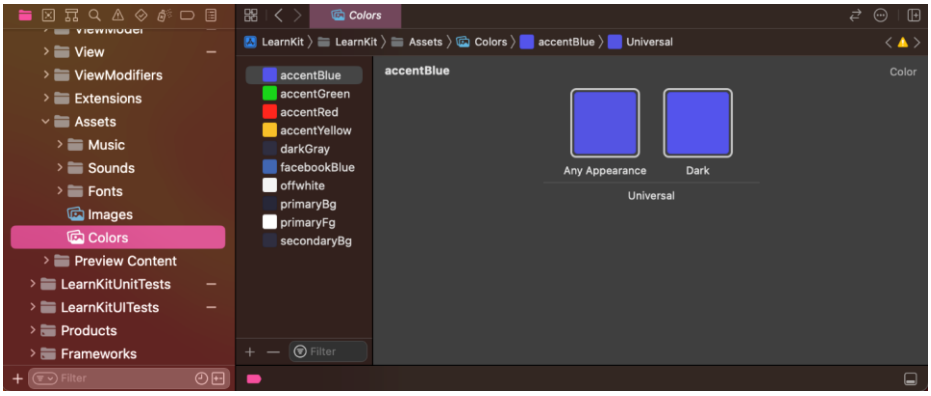
V prototypu je vytvořena funkce s názvem „fontMontserrat“, která má jako parametry velikost a tloušťku písma, která je daná enumerací „FontWeight“. Tuto funkci lze použít na jakoukoliv komponentu. Důvodem tvorby této funkce je snaha vyvarovat se potřeby psaní textového řetězce reprezentující název fontu při každém jeho použití. Při tomto procesu může vývojář udělat chybu a název fontu vepsat nesprávně. Použitím metody je toto riziko eliminováno. Chyba může vzniknout pouze v místě definice funkce.

```
Text(associatedTab.rawValue)
    .opacity(isSelected ? 0 : 1)
    .foregroundColor(.white)
    .fontMontserrat(size: 10, weight: .semiBold)
```

Obrázek 50 Příklad použití fontu Montserrat ve SwiftUI

6.7.3 Barvy

Všechny definice barev, které se vyskytují v designu uživatelského rozhraní jsou součástí souboru „Colors.xcassets“. Barvy lze vyjádřit mnoha způsoby, například pomocí škály RGB (red-green-blue). Tyto hodnoty lze získat z návrhu uživatelského rozhraní v aplikaci Adobe Xd. Pro každou položku lze definovat odlišnou barvu pro světlý a tmavý mód aplikace.



Obrázek 51 Katalog Colors pro definici barev v prototypu

Framework SwiftUI používá pro vyjádření barev strukturu „Color“ [83], kterou lze inicializovat mnoha způsoby. V případě, že jsou barvy uloženy v souboru s příponou „xcassets“, lze použít konstruktor s parametrem datového typu „String“. Textový řetězec reprezentuje název definované barvy v souboru.

```
extension Color{
    static let primaryBg = Color("primaryBg")
    static let primaryFg = Color("primaryFg")
    static let secondaryBg = Color("secondaryBg")
    static let accentBlue = Color("accentBlue")
    static let accentGreen = Color("accentGreen")
    static let accentRed = Color("accentRed")
    static let accentYellow = Color("accentYellow")
    static let darkGray = Color("darkGray")
    static let offwhite = Color("offwhite")
    static let facebookBlue = Color("facebookBlue")
}
```

Obrázek 52 Inicializace barev z katalogu pomocí statických konstant

Všechny používané barvy jsou v prototypu inicializovány jako statické proměnné v rámci rozšíření datové struktury „Color“.

6.7.4 Grafika

Grafické elementy jako ikony, ilustrace nebo fotky jsou v projektu uloženy v souboru „Images.xcassets“ a jsou roztříděny do několika složek pro lepší přehlednost. Ve SwiftUI lze tyto elementy zobrazit pomocí struktury „Image“. [84] Podobně jako u barev má i tato struktura konstruktor s parametrem datového typu „String“, který popisuje název uloženého grafického souboru.

```
Image("LearnKitLogo")
    .resizable()
    .scaledToFit()
    .frame(width: 40)
```

Obrázek 53 Příklad zobrazení loga aplikace ve SwiftUI

V ukázce je příklad použití struktury „Image“ pro zobrazení loga aplikace v části pro vyhledávání obsahu.

7 UKÁZKY TESTOVÁNÍ PROTOTYPU IOS APLIKACE

Fáze testování je nedílnou součástí vývoje mobilní aplikace pro iOS. Testování je určeno k ověření kvality aplikace. Cílem unit a UI testování je nalézt chyby v kódu a opravit je. Součástí jazyka Swift je nativní knihovna XCTest, která obsahuje řadu funkcionalit pro tvorbu unit, performance a UI testů. [85] Následující část obsahuje příklady implementací unit a UI testů. Pojmem unit (jednotkový) se označuje test, který se soustředí na validaci individuálních nezávislých jednotek v softwaru. Unit testy většinou implementují vývojáři v počátečních fázích vývoje. UI (user interface) testy jsou určeny k validaci uživatelského rozhraní. Cílem je zjistit, zda visuální elementy aplikace a jejich interakce s uživatelem splňují požadavky. [86][87]

7.1 Unit testy

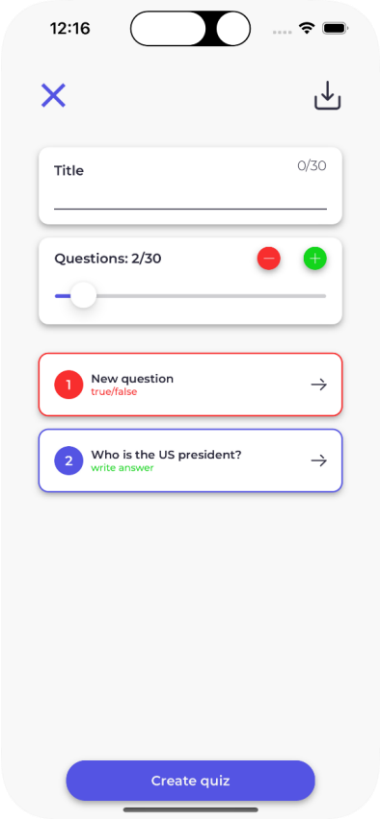
V následujícím kódu je zobrazena třída „LearnKitUnitTests“, která dědí od třídy „XCTestCase“ z knihovny XCTest. V definici třídy lze přepsat dvě metody z rodičovské třídy – „setUpWithError“ a „tearDownWithError“. V nich je možné uvést příkazy, které se vykonají před a po zavolání každého testu v třídě. Testovací případ je v kódu reprezentován jako metoda třídy, jejíž název musí začínat slovem „test“.

```
final class LearnKitUnitTests: XCTestCase {

    override func setUpWithError() throws {
        // Put setup code here. This method is called before the invocation of
    }

    override func tearDownWithError() throws {
        // Put teardown code here. This method is called after the invocation
    }

    func test_validateQuestion_any_type_empty_query(){
        let question = QuestionViewModel()
        question.validateQuestion()
        XCTAssertFalse(question.isValid)
    }
}
```

Obrázek 54 Třída pro realizaci unit testů v prototypu

V ukázce je metoda, která testuje validaci jednotlivých otázek při tvorbě kvízu uživatelem. Každá otázka musí splňovat předem stanovené požadavky, které kontroluje uvedená metoda „validateQuestion“. Tato metoda nastaví členskou proměnnou třídy „QuestionViewModel“

s názvem „isValid“ podle toho, zda splňuje všechna kritéria. Jedním z požadavků je, že každý objekt reprezentující otázku, musí obsahovat neprázdný textový řetězec, který ji popisuje. Uvedený testovací případ zachycuje situaci, kdy je popis vytvářené otázky prázdný. Po zavolaní metody pro validaci otázky musí mít proměnná „isValid“ hodnotu „false“. Výsledek metody „validateQuestion“ je v uživatelském rozhraní indikován při tvorbě nového kvízu. Otázka je označena červeně, nebo modře podle toho, zda je validní.



Obrázek 55 Ukázka validace otázky při tvorbě kvízu

7.2 UI testy

Struktura třídy pro tvorbu UI testů je totožná s třídou pro tvorbu unit testů. Součástí knihovny XCTest je třída „XCUIApplication“, pomocí které lze automaticky vytvořit, monitorovat a ukončit instanci testované aplikace.

```
func test_register_sheet_content(){
    let app = XCUIApplication()
    app.launch()

    let registerNowButton = app.buttons["RegisterNowButton"]
    registerNowButton.tap()

    let titleText = app.staticTexts["Create an account"]
    let titleTextExists = titleText.waitForExistence(timeout: 0.5)

    XCTAssertTrue(titleTextExists)
}
```

Obrázek 56 Příklad UI testu v prototypu

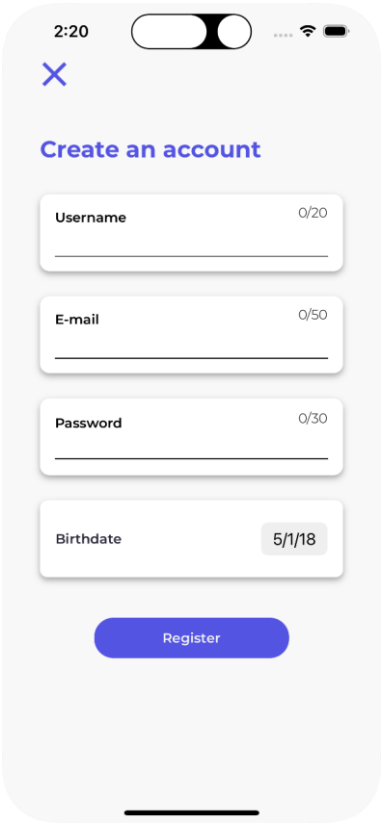
Uvedený příklad UI testu kontroluje, zda se po stisknutí tlačítka pro zobrazení registračního formuláře pro uživatele zobrazí konkrétní text. Testovací případ začíná vytvořením instance třídy „XCUIApplication“ a zavolání metody „launch“ pro spuštění aplikace. Po spuštění se aplikace nachází ve formuláři pro přihlášení s možností stisknout tlačítko pro přesměrování na registrační formulář pro případ, že uživatel žádný účet ještě nevlastní. V rámci objektu „app“ lze vyhledat konkrétní grafické elementy jako například uvedené tlačítko s identifikátorem „RegisterNowButton“ a zavolat na něj metodu „tap“. Uvedený identifikátor je nutno specifikovat v kódu, který tlačítko popisuje, pomocí metody „accessibility“.

```
Button {
    showRegisterSheet.toggle()
} label: {
    Text("Don't have an account? ")
        .foregroundColor(.accentBlue)
    +
    Text("Register now!")
        .foregroundColor(.accentGreen)

}
.accessibility(identifier: "RegisterNowButton")
```

Obrázek 57 Ukázka použití metody accessibility na tlačítko

Další částí testu je nalezení statického textu „Create an account“, který by se měl zobrazit jako nadpis formuláře pro registraci uživatele. V testu je uvedena metoda „waitForExistence“, ve které lze definovat, jak dlouho je povoleno čekat na zobrazení

hledaného elementu. Metoda vrací datový typ „Bool“, který vyjadřuje, zda element existuje, nebo ne. Poslední krok testu ověřuje, zda je tato hodnota vždy „true“.



Obrázek 58 Formulář pro registraci nového uživatele do systému

8 FUNKCIONALITY PRO BUDOUCÍ ITERACE PROTOTYPU

Vzhledem k tomu, že tato práce popisuje vývoj první iterace prototypu, tak neobsahuje zdaleka všechny funkcionality, které by mohl finální produkt obsahovat. Proto jsou v následující části zmíněny nápady na funkcionality, které by mohly být potenciálně implementovány v příštích vývojových iteracích.

8.1 Přidání uživatelů do seznamu přátel

Uživatel by měl možnost vyhledat ostatní uživatele, kteří mají vytvořený účet v aplikaci a zaslat jim žádost o přátelství podobně jako na dnešních sociálních sítích. Tímto propojením by se otevřel prostor pro jejich vzájemnou komunikaci a sdílení dat, což by dalo možnost implementaci mnoha nových funkcionalit na bázi kooperace.

8.2 Sdílení listů vytvořených uživatelem mezi přáteli

Tato funkcionalita je příkladem použití propojení uživatelů v aplikaci. Uživatel by mohl sdílet svůj vytvořený List s jedním nebo skupinou přátel. Nabízí se zde i nastavení práv autora listu pro čtení, editování a mazání, které by se týkaly přátel, se kterými je List sdílen.

8.3 Místnosti pro vyplnění kvízu více uživateli v reálném čase

V této funkcionalitě by uživatel vytvořil místnost, do které by se připojili ostatní uživatelé, například pomocí vygenerovaného QR kódu. Zakladatel místnosti by vybral kvíz z Listu, který sám vytvořil, nebo z listu od ověřeného autora a provedl jeho herní nastavení včetně časového limitu, počtu nápověd apod. Po připojení uživatelů do místnosti by zakladatel spustil kvíz, který by v reálném čase začali všichni účastníci vyplňovat. Po ukončení časového limitu by měli všichni uživatelé v místnosti k dispozici výsledky všech respondentů kvízu. Tato funkcionalita by se mohla používat ve školách, jako alternativní způsob výuky nebo testovaní studentů.

8.4 Aktivace prémiového účtu

Po zaplacení jednorázového poplatku, nebo měsíčního předplatného by uživatel mohl získat přístup k prémiovému účtu, který by obsahoval například verzi aplikace bez reklam nebo prémiové funkcionality, které uživatel s účtem zdarma k dispozici nemá.

8.5 Webová platforma pro ověřené autory

Pro první iteraci prototypu zatím neexistuje žádná platforma pro registraci a správu ověřených autorů, kteří by mohli své Listy publikovat. Aktuálně jsou data o ověřených autorech a jejich Listech do databáze přidány ručně. Vytvoření webové platformy, kde by se autoři mohli registrovat a spravovat své Listy, případně výdělky za své příspěvky, je klíčovou funkcionalitou pro finální verzi aplikace. Výhodou platformy Firebase je, že obsahuje SDK i pro implementaci webových aplikací. Všechny služby, jako je databáze Firestore nebo uložiště Storage, může mobilní a webová aplikace sdílet.

ZÁVĚR

Bakalářská práce popsala proces návrhu a implementace vzdělávací aplikace pro platformu iOS. Nejprve byla provedena literární rešerše technologických nástrojů, které byly při vývoji aplikace použity. Dále byl uveden popis použitých softwarových paradigmat a návrhových vzorů. Součástí dokumentu byl popis průzkumu konkurenčních řešení, ze kterých vyplynuly požadavky na vzhled a funkcionality prototypu. Poté byl vytvořen návrh uživatelského rozhraní v aplikaci Adobe Xd, na jehož základě byla implementována aplikace. Popis implementace prototypu zachytil nejdůležitější body z hlediska vytvoření a nastavení nového projektu v Xcode a integrace služby Firebase. Dále byly důkladně popsány základní CRUD operace pro manipulaci s databází. Na prototypu byly předvedeny ukázky testovaní iOS aplikace. Výsledkem bakalářské práce je první vývojová iterace prototypu vzdělávací aplikace LearnKit. Aplikace splnila definované funkcionální požadavky a uživatel ji může použít pro tvorbu vlastních výukových materiálů. Na závěr byly popsány nápady na rozšíření pro budoucích iterace aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] The Swift Programming Language [online]. Swift 5.7. Cupertino: Apple, 2022 [cit. 2023-05-04]. Dostupné z: https://books.apple.com/us/book/the-swift-programming-language-swift-5-7/id881256329

[2] HOFFMAN, Jon. Mastering Swift 5: Deep dive into the latest edition of the Swift programming language. 5th Edition. Velká Británie: Packt Publishing, 2019. ISBN 1789139864.

[3] The Basics | Documentation. Swift [online]. Cupertino: Apple, c2014-2023 [cit. 2023-05-04]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/thebasics

[4] Automatic Reference Counting | Documentation. Swift [online]. Cupertino: Apple, c2014-2023 [cit. 2023-05-04]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/automaticreferencecounting

[5] Inheritance | Documentation. Swift [online]. Cupertino: Apple, c2014-2023 [cit. 2023-05-04]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/inheritance

[6] Generics | Documentation. Swift [online]. Cupertino: Apple, c2014-2023 [cit. 2023-05-04]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/generics/

[7] Introducing SwiftUI: Building Your First App. In: Apple [online]. Cupertino: Apple, 2019 [cit. 2023-05-04]. Dostupné z: https://developer.apple.com/videos/play/wwdc2019/204/

[8] SwiftUI | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-04]. Dostupné z: https://developer.apple.com/documentation/swiftui

[9] HUDSON, Paul. SwiftUI by Example. Hacking with Swift [online]. Spojené království: Hudson Heavy Industries, c2022 [cit. 2023-05-04]. Dostupné z: https://www.hackingwithswift.com/quick-start/swiftui

[10] View | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-04]. Dostupné z: https://developer.apple.com/documentation/swiftui/view

[11] ViewModifier | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-04]. Dostupné z: https://developer.apple.com/documentation/swiftui/viewmodifier

[12] Initialization | Documentation. Swift [online]. Cupertino: Apple, c2014-2023 [cit. 2023-05-04]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/initialization

[13] Model-View-ViewModel (MVVM). TechTarget [online]. Newton: TechTarget Contributor, 2019 [cit. 2023-05-04]. Dostupné z: https://www.techtarget.com/whatis/definition/Model-View-ViewModel

[14] Managing a Shared Resource Using a Singleton | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-04]. Dostupné z: https://developer.apple.com/documentation/swift/managing-a-shared-resource-using-a-singleton

[15] ROUÉ, Gwendal. MVVM and Dependency Injection. Swift Package Index [online]. Spojené království: SPI Operations Limited, c2023 [cit. 2023-05-04]. Dostupné z: https://swiftpackageindex.com/groue/grdbquery/0.7.0/documentation/grdbquery/mvvm

[16] BERLIN, Josh a René CACHEAUX, FERGUSON, Darren, ed. Advanced iOS App Architecture: Real-World App Architecture in Swift. Fourth Edition. Virginia: Razeware, 2022. ISBN 978-1950325610.

[17] KHAN, Aasif. MVVM and SwiftUI. Appy Pie [online]. Virginia: Appy Pie, 2023 [cit. 2023-05-04]. Dostupné z: https://www.appypie.com/mvvm-swiftui-how-to

[18] EKREN, Esat Kemal. What Is Xcode and How to Use It? Netguru [online]. Poland: Netguru, 2022 [cit. 2023-05-04]. Dostupné z: https://www.netguru.com/blog/what-is-xcode-and-how-to-use-it

[19] JACOBS, Bart. Six Common Questions About Xcode Answered. Cocoacasts [online]. Belgie: Code Foundry, c2016-2023 [cit. 2023-05-04]. Dostupné z: https://cocoacasts.com/six-common-questions-about-xcode-answered

[20] Xcode | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-04]. Dostupné z: https://developer.apple.com/documentation/xcode

[21] LABRAYILOV, Majid. Mastering SwiftUI previews. Swift with Majid [online]. 2021 [cit. 2023-05-04]. Dostupné z: https://swiftwithmajid.com/2021/03/10/mastering-swiftui-previews

[22] SwiftUI Previews: Validating views in different states. Avanderlee [online]. Amsterdam: SwiftLee, 2020 [cit. 2023-05-04]. Dostupné z: https://www.avanderlee.com/swiftui/previews-different-states

[23] HINDI, Daniel. How to Publish an App to the App Store. BuildFire [online]. San Diego [cit. 2023-05-04]. Dostupné z: https://buildfire.com/how-to-publish-an-app-to-the-app-store

[24] Running your app in Simulator or on a device | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-04]. Dostupné z: https://developer.apple.com/documentation/xcode/running-your-app-in-simulator-or-on-a-device

[25] What is Google Firebase? Deepak Bhat [online]. 2022 [cit. 2023-05-04]. Dostupné z: https://www.deepakbhatt.in/what-is-google-firebase/#History_of_Google_Firebase

[26] Build Documentation | Firebase Documentation. Firebase [online]. San Francisco: Google, 2021 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/docs/build

[27] BATSCHINSKI, George. What is Firebase? All the secrets unlocked. Back4app [online]. California, c2022 [cit. 2023-05-04]. Dostupné z: https://blog.back4app.com/firebase

[28] The Good and the Bad of Firebase Backend Services. Altexsoft [online]. California, 2019 [cit. 2023-05-04]. Dostupné z: https://www.altexsoft.com/blog/firebase-review-pros-cons-alternatives

[29] What Are App IDs and Bundle Identifiers. Cocoacasts [online]. Belgie: Code Foundry, c2016-2023 [cit. 2023-05-04]. Dostupné z: https://cocoacasts.com/what-are-app-ids-and-bundle-identifiers

[30] Package Manager. Swift [online]. Cupertino: Apple, c2023 [cit. 2023-05-04]. Dostupné z: https://www.swift.org/package-manager

[31] CARLI, Sullivan De. How to add Firebase to a SwiftUI Project with SPM. Medium [online]. San Francisco: Swift Productions, 2021 [cit. 2023-05-04]. Dostupné z: https://medium.com/swift-productions/how-to-add-firebase-with-spm-swiftui-d083189d510d

[32] Add Firebase to your Apple project | Firebase Documentation. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/docs/ios/setup

[33] Firebase Authentication. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/docs/auth

[34] Firebase Authentication | Introduction. Full-Stack Firebase [online]. 2019 [cit. 2023-05-04]. Dostupné z: https://www.fullstackfirebase.com/firebase-authentication/introduction

[35] Cloud Firestore Data model | Firebase Documentation. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/docs/firestore/data-model

[36] Cloud Firestore | Store and sync app data at global scale. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/products/firestore

[37] Firestore – Transactions and batched writes. AIR Native Extensions [online]. Australia: Distriqt, c2023 [cit. 2023-05-04]. Dostupné z: https://docs.airnativeextensions.com/docs/firebase/firestore/transactions-and-batched-writes

[38] Transactions and batched writes | Firebase Documentation. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/docs/firestore/manage-data/transactions

[39] Perform simple and compound queries in Cloud Firestore | Firebase Documentation. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/docs/firestore/query-data/queries

[40] Get started with Cloud Firestore Security Rules | Firebase Documentation. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/docs/firestore/security/get-started

[41] Cloud Storage for Firebase. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-04]. Dostupné z: https://firebase.google.com/docs/storage

[42] BOHBOT, Kobi. Firebase Storage: What It Is and How It Works. Backg4app [online]. California, c2023 [cit. 2023-05-04]. Dostupné z: https://blog.back4app.com/firebase-storage

[43] Learn how to design interactive prototypes, wireframes, and graphics using XD. Adobe

[online]. San Jose: Adobe, 2021 [cit. 2023-05-04]. Dostupné z: https://helpx.adobe.com/cz/xd/help/adobe-xd-overview.html

[44] SHAUL, Natalie Ben. How to export Adobe XD to HTML. Animaapp [online]. New York: Anima, 2022 [cit. 2023-05-04]. Dostupné z: https://www.animaapp.com/blog/design-to-code/how-to-export-adobe-xd-to-html-2/

[45] Learn how to share your designs and prototypes with stakeholders, developers, or fellow designers. Adobe [online]. San Jose: Adobe, 2021 [cit. 2023-05-04]. Dostupné z: https://helpx.adobe.com/cz/xd/help/share-designs-prototypes.html

[46] Adobe XD Pricing 2023. TrustRadius [online]. Austin: TrustRadius, c2013-2023 [cit. 2023-05-04]. Dostupné z: https://www.trustradius.com/products/adobe-xd/pricing

[47] Adobe XD Tools. JavaTpoint [online]. India: JavaTpoint, c2011-2021 [cit. 2023-05-04]. Dostupné z: https://www.javatpoint.com/adobe-xd-tools

[48] Preview your mobile devices. Adobe [online]. San Jose: Adobe, 2023 [cit. 2023-05-04]. Dostupné z: https://helpx.adobe.com/xd/help/preview-mobile.html

[49] Singleton. Refactoring guru [online]. Ukraine, c2014-2023 [cit. 2023-05-04]. Dostupné z: https://refactoring.guru/design-patterns/singleton

[50] Let's examine the pros and cons of the Singleton design pattern. FreeCodeCamp [online]. San Francisco, 2018 [cit. 2023-05-04]. Dostupné z: https://www.freecodecamp.org/news/singleton-design-pattern-pros-and-cons-e10f98e23d63

[51] Swift Singleton. Programiz [online]. Nepal: Parewa Labs [cit. 2023-05-04]. Dostupné z: https://www.programiz.com/swift-programming/singleton

[52] SOSUN, Batikan. Dependency Injection in Swift. Medium [online]. San Francisco: Sahibinden Technology, 2022 [cit. 2023-05-04]. Dostupné z: https://medium.com/sahibinden-technology/dependency-injection-in-swift-11756a07a064

[53] THORBEN, Janssen. Design Patterns Explained – Dependency Injection with Code Examples. Stackify [online]. California, 2023 [cit. 2023-05-04]. Dostupné z: https://stackify.com/dependency-injection

[54] Dependency Injection. ProfessionalQA [online]. United States: ProfessionalQA, 2017 [cit. 2023-05-04]. Dostupné z: https://www.professionalqa.com/dependency-injection

[55] GILLIS, Alexander S. a Kevin FERGUSON. What is dependency injection in object-oriented programming. TechTarget [online]. Newton: TechTarget, 2023 [cit. 2023-05-05]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/dependency-injection

[56] A Comprehensive Guide to Dependency Injection in C#. ExecuteCommands [online]. 2022 [cit. 2023-05-05]. Dostupné z: https://executecommands.com/guide-dependency-injection-adv-dis-mock-exampl

[57] Dependency Injection. TutorialsTeacher [online]. c2023 [cit. 2023-05-05]. Dostupné z: https://www.tutorialsteacher.com/ioc/dependency-injection

[58] Dependency Injection. Devopedia [online]. 2022 [cit. 2023-05-05]. Dostupné z: https://devopedia.org/dependency-injection

[59] Inheritance | Documentation. Swift [online]. Cupertino: Apple, c2014-2023 [cit. 2023-05-05]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/inheritance

[60] Swift Inheritance. Programiz [online]. Nepal: Parewa Labs [cit. 2023-05-04]. Dostupné z: https://www.programiz.com/swift-programming/inheritance

[61] Protocols | Documentation. Swift [online]. Cupertino: Apple, c2014-2023 [cit. 2023-05-05]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/protocols/

[62] CECI, L. Leading learning and education apps worldwide in 2022, by downloads. In: Statista [online]. New York: Statista, 2023 [cit. 2023-05-04]. Dostupné z: https://www.statista.com/statistics/1284623/top-education-apps-global-by-downloads/

[63] What is Duolingo? Duolingo Help Center [online]. Pittsburgh: Duolingo, 2022 [cit. 2023-05-04]. Dostupné z: https://support.duolingo.com/hc/en-us/articles/204829090-What-is-Duolingo

[64] How much does Duolingo Cost? Duolingo Help Center [online]. Pittsburgh: Duolingo, 2022 [cit. 2023-05-04]. Dostupné z: https://support.duolingo.com/hc/en-us/articles/207362396-How-much-does-Duolingo-cost

[65] About Classroom. Google Support [online]. Mountain View: Google, c2023 [cit. 2023-05-04]. Dostupné z: https://support.google.com/edu/classroom/answer/6020279

[66] EDWARDS, Luke. What is Brainly and How Can It Be Used to Teach? Tips & Tricks. Tech Learning [online]. New York, 2022 [cit. 2023-05-04]. Dostupné z: https://www.techlearning.com/how-to/what-is-brainly-and-how-can-it-be-used-to-teach-tips-and-tricks

[67] Srovnání plánů pro jednotlivce, studenty a učitele a týmy. Adobe [online]. San Jose: Adobe, c2023 [cit. 2023-05-08]. Dostupné z: https://www.adobe.com/cz/creativecloud/compare-plans.html

[68] ADAMS, Sean, Peter DAWSON, John FOSTER a Tony SEDDON. Graphic Design Rules: 365 Essential DOS and Dont's. Revised edition with a new introduction by Sean Adams. New York: Princeton Architectural Press, 2020. ISBN 978-1616898762.

[69] UIStoryboard | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-08]. Dostupné z: https://developer.apple.com/documentation/uikit/uistoryboard

[70] SF Symbols – Apple Developer. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-08]. Dostupné z: https://developer.apple.com/sf-symbols

[71] Property Wrappers in Swift explained with code examples. Avanderlee [online]. Amsterdam: SwiftLee, 2020 [cit. 2023-05-08]. Dostupné z: https://www.avanderlee.com/swift/property-wrappers

[72] Map Cloud Firestore data with Swift Codable. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-08]. Dostupné z: https://firebase.google.com/docs/firestore/solutions/swift-codable-data-mapping

[73] ObservableObject | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-08]. Dostupné z: https://developer.apple.com/documentation/combine/observableobject

[74] What is the Property List or plist in IOS. MyCodeTips [online]. [cit. 2023-05-08]. Dostupné z: https://mycodetips.com/ios/what-is-the-property-list-or-plist-in-ios-3429.html

[75] Core Image | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-08]. Dostupné z: https://developer.apple.com/documentation/coreimage

[76] HUDSON, Paul. GitHub – twostraws/CodeScanner. GitHub [online]. Spojené království: Hudson Heavy Industries, c2021 [cit. 2023-05-08]. Dostupné z: https://github.com/twostraws/CodeScanner

[77] Usage and limits | Firestore. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-08]. Dostupné z: https://firebase.google.com/docs/firestore/quotas

[78] Map Cloud Firestore data with Swift Codable. Firebase [online]. San Francisco: Google, 2023 [cit. 2023-05-08]. Dostupné z: https://firebase.google.com/docs/firestore/solutions/swift-codable-data-mapping

[79] CRUD – definition & overview. Sumo logic [online]. Redwood City, c2023 [cit. 2023-05-08]. Dostupné z: https://www.sumologic.com/glossary/crud

[80] STEVENSON, Doug. Why are the Firebase API asynchronous? Medium [online]. San Francisco: Firebase Developers, 2018 [cit. 2023-05-08]. Dostupné z: https://medium.com/firebase-developers/why-are-firebase-apis-asynchronous-callbacks-promises-tasks-e037a6654a93

[81] AVFAudio | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-08]. Dostupné z: https://developer.apple.com/documentation/avfaudio

[82] ULANOVSKY, Julieta, Sol MATAS, Juan Pablo DEL PERAL a Jacques LE BAILLY. Montserrat – Google Fonts. Google Fonts [online]. San Francisco: Google [cit. 2023-05-08]. Dostupné z: https://fonts.google.com/specimen/Montserrat/about?query=montserrat

[83] Color | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-08]. Dostupné z: https://developer.apple.com/documentation/swiftui/color

[84] Image | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-08]. Dostupné z: https://developer.apple.com/documentation/swiftui/image

[85] XCTest | Apple Developer Documentation. Apple [online]. Cupertino: Apple, c2023 [cit. 2023-05-08]. Dostupné z: https://developer.apple.com/documentation/xctest

[86] Unit Testing | Software Testing. GeeksforGeeks [online]. India [cit. 2023-05-08]. Dostupné z: https://www.geeksforgeeks.org/unit-testing-software-testing

[87] BOSE, Shreya. UI Testing: A Detailed Guide. BrowserStack [online]. Dublin, 2022 [cit. 2023-05-08]. Dostupné z: https://www.browserstack.com/guide/ui-testing-guide

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

iOS	Operační systém pro telefony iPhone od společnosti Apple
macOS	Operační systém pro počítače Macintosh od společnosti Apple
watchOS	Operační systém pro chytré hodinky od společnosti Apple
tvOS	Operační systém pro produkt Apple TV od společnosti Apple
ARC	Automatic Reference Counting – automatické počítání referencí v rámci správy paměti kompilátoru
WWDC	Worldwide Developer Conference – každoroční konference společnosti Apple pro vývojáře
MVVM	Model-View-ViewModel – architektonický vzor pro vývoj počítačového softwaru
API	Application Programming Interface – rozhraní pro programování aplikací
SDK	Software Development Kit – sada nástrojů pro vývoj aplikací na danou platformu
SQL	Structured Query Language – programovací jazyk pro práci s relační databází
NoSQL	Databázový koncept, který používá jiné prostředky než relační databáze
HTML	Hypertext Markup Language – značkovací jazyk pro tvorbu webových stránek
QR kód	Quick Response kód – prostředek pro rychlý přenos libovolné informace do mobilního zařízení
GB	Gigabajt – jednotka digitální informace
MiB	Mebibit – jednotka digitální informace
SVG	Scalable Vector Graphics – formát souboru popisující dvojrozměrnou vektorovou grafiku
PNG	Portable Network Graphics – formát souboru pro bezztrátovou kompresi rastrové grafiky
UI	User Interface – uživatelské rozhraní
SF symboly	San Francisco symboly – knihovna ikon pro vývoj na platformách Apple

XML	Extensible Markup Language – obecný značkující jazyk používaný pro serializaci dat
CRUD	Create, Read, Update, Delete – čtyři základní operace nad objektem v trvalém uložišti
URL	Uniform Resource Locator – řetězec znaků specifikující umístění zdroje na Internetu
TTF	TrueType Font – standard pro popis vektorových počítačových písem
RGB	Red-Green-Blue – způsob míchání barev používaný v barevných monitorech

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH