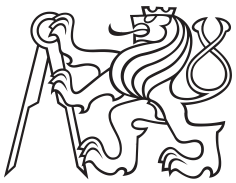
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Kvízová aplikace s automatizovanou podporou generování otázek

Matěj Bárta

Vedoucí: Ing. Ivo Malý, Ph.D.
Obor: Softwarové inženýrství a technologie
Srpen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bárta** Jméno: **Matěj** Osobní číslo: **483867**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Kvízová aplikace s automatizovanou podporou generování otázek

Název bakalářské práce anglicky:

Quiz application with support for automated question generation

Pokyny pro vypracování:

Analyzujte existující mobilní kvízové aplikace s generováním otázek. Zaměřte se na způsob vkládání nových otázek s využitím obrázků, jejich anotaci a generování vhodných alternativních odpovědí. Navrhněte vhodnou architekturu systému pro generování otázek a vyhodnocování odpovědí, kde klientská aplikace poběží na mobilním zařízení. Dále navrhněte uživatelské rozhraní aplikace pro tvorbu i odpovídání otázek pomocí nízkoúrovňových prototypů. Na základě požadavků na aplikaci vyberte vhodnou knihovnu pro polo/automatizaci tvorby a vyhodnocení nových otázek. Výslednou aplikaci na základě jejího návrhu implementujte na platformě iOS pro mobilní telefon. Funkčnost aplikace vyhodnoťte na alespoň 30 objektech z alespoň 4 kategorií.

Seznam doporučené literatury:

1. Firebase Machine Learning, https://firebase.google.com/products/ml
2. iOS Machine Learning, https://developer.apple.com/machine-learning/
3. M. Geldard, J. Manning, P. Buttfield-Addison, T. Nugent. Practical Artificial Intelligence with Swift: From Fundamental Theory to Development of AI-Driven Apps, O'Reilly Media, Inc. 2019

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ivo Malý, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2021** Termín odevzdání bakalářské práce: **13.08.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Ivo Malý, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování

V první řadě bych chtěl poděkovat vedoucímu práce Ing. Ivu Malému, Ph.D. za jeho trpělivost. Dále velké díky patří ČVUT, za vzdělání které mi poskytlo. Na závěr bych chtěl poděkovat Dominikovi P. a Michalovi S., za jejich rady a pomoc při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

Abstrakt

Výstupem bakalářské práce je kvízová mobilní aplikace pro platformu iOS s možností generování otázek. Ty se sestavují z obrázku a odpovědí. Uživatelé mají možnost vytvářet sady otázek jak manuálně, tak generováním, které využívá strojové učení a slovní vyhledávač. Tento dokument obsahuje veškeré analýzy, návrhy, implementace a rozhodnutí, která stojí za vývojem této aplikace a zároveň popisuje použitelnost relativně nových technologií od společnosti Apple.

Klíčová slova: softwarová analýza, mobilní aplikace, Swift, Firebase, Strojové učení, SwiftUI, Combine

Vedoucí: Ing. Ivo Malý, Ph.D.

Abstract

The output of the bachelor's thesis is a quiz mobile application for the iOS platform with the possibility of generating questions. These are composed of a picture and answers. Users can create question sets manually or by generation, which uses machine learning and a word finder. This document contains all the analysis, design, implementation, and decisions behind the development of this application, as well as describes the applicability of relatively new technologies from Apple.

Keywords: software analysis, mobile application, Swift, Firebase, Machine learning, SwiftUI, Combine

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

1.1 Motivace

V dnešní době, kdy trh s mobilními aplikacemi je naprosto přehlcen, je téměř nemožné přijít s nápadem, který by již nebyl stvořen. K téměř každé aplikaci existuje alespoň nějaká alternativa a tak může být obtížné na takovém trhu uspět. Navíc vývoj mobilních aplikací může být pro naprosté nováčky časově náročný a složitý. Proto volba správných technologií je v tomto ohledu klíčová.

Možná je tedy podivuhodné, proč jsem si vybral platformu iOS od společnosti Apple. Ta se sice na trhu prezentuje jako moderní technologická společnost, známa pro svůj simplistický design a uživatelskou přívětivost svých zařízení. Pokud jde ovšem o vývoj, ten dle slov kolegy senior iOS vývojáře pokulhává. Plno funkcionalit chybí, dokumentace jsou nekompletní a fóra moc nefungují. Snaha o zlepšení přišla roku 2019, kdy společnost představila dva nové frameworky. Combine[19], sloužící k zpracování dat v průběhu času a SwiftUI[20], deklarativní nástroj pro tvorbu uživatelského rozhraní. Ty měli tuto situaci napravit.

Jelikož již delší dobu sleduji vývoj těchto technologií, zajímalo mne tedy, jestli se dva roky od představení situace změnila a zda-li je za použití čistě těchto technologií možné vytvořit plnohodnotnou aplikaci.

Pro otestování je tak důležité zvolit téma, které pokryje co nejvíce jejich funkcionalit. Zároveň, by mělo být, alespoň částečně inovativní a užitečné. Na základě těchto předpokladů a krátké debatě s vedoucím práce jsme přišli

na téma kvízové aplikace využívající strojové učení pro polo-automatizaci generování otázek. Strojové učení zažívá v poslední době velký rozvoj a jeho implementace by tak mohla pomoci aplikaci vyniknout. Nutno také podotknout, že aplikace bude v anglickém jazyce.

1.2 Cíle

Cílem této práce je vytvořit analýzu existujících řešení a na základě té stanovit měřítka úspěšnosti aplikace. Dále analyzovat metody vhodné pro generování kvizových otázek a vybrat nástroje pro jejich provedení. V neposlední řadě navrhnout architekturu systému a příslušné uživatelské rozhraní, které bude dodržovat guidelines[1] platformy iOS. To vše implementovat za pomoci SwiftUI a Combine. Nakonec popsat použitelnost těchto technologií a otestovat funkcionalitu na sadě několika objektů.

[1] guidelines - Souhrn konceptů a pravidel pro vytvoření uživatelsky přívětivé aplikace.

Kapitola 2

Analýza

Abych byl schopen aplikaci správně navrhnout a následně implementovat, je důležité provést několik analýz. Ty slouží k ujasnění požadavků, výběru správných nástrojů a metod.

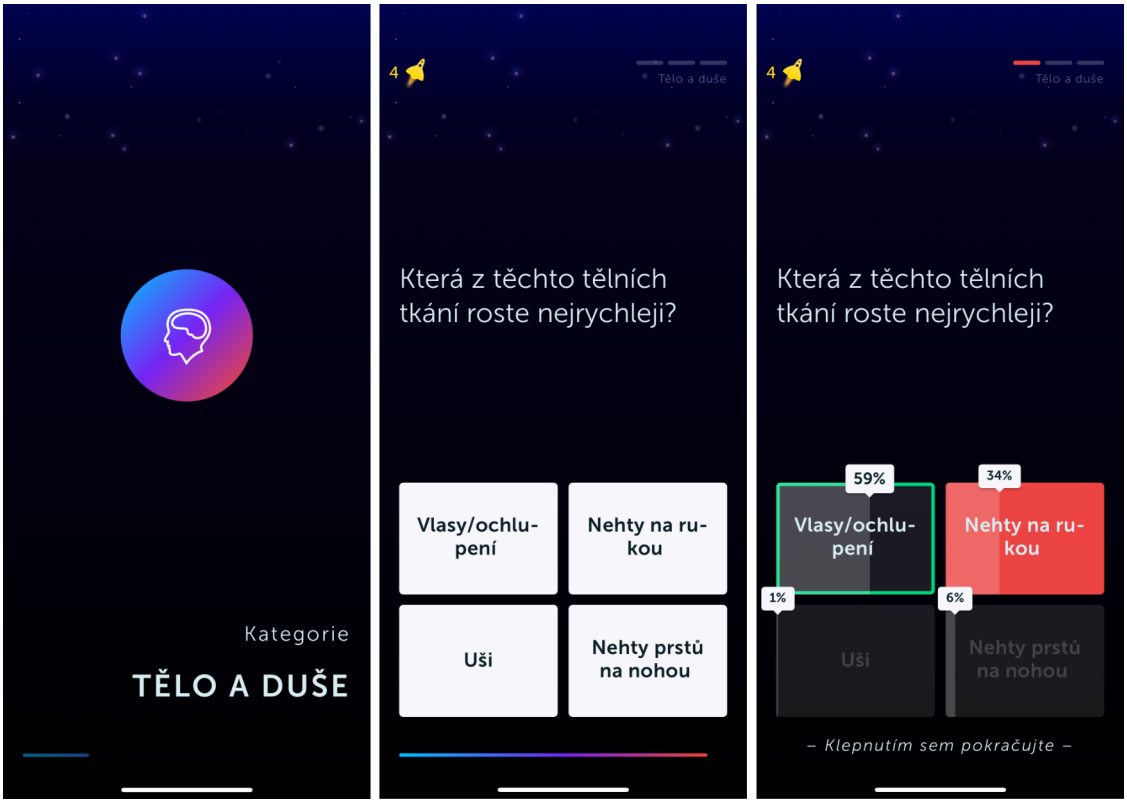
2.1 Existující řešení

Vzhledem k tomu, že už od začátku je aplikace zamýšlena pro operační systém iOS, tak i analýza podobných řešení se odehrávala na jediném místě pro její distribuci, AppStore. Zde se většina kvízových aplikací zaměřuje spíše na kvízové testy, než-li na tvorbu a generování otázek. Hledal jsem tedy v různých kategoriích komponenty, které by se následně dali aplikovat na tento projekt.

2.1.1 Quiz Planet

[1] Za zmínku z kategorie Kvízových aplikací stojí například Quiz Planet. Tato cross-platform aplikace si svou slávu získala přes Facebook games. Zde vyhrála nejpopulárnější hru července 2019. Ke své popularitě jí pravděpodobně pomohl její jednoduchý, čistý design, prostá hratelnost a množství otázek (Vydavatel udává, že počet otázek přesahuje 10 000)

[1] `https://quizplanet.game`



(a):

Obrázek 2.1: Quiz Planet

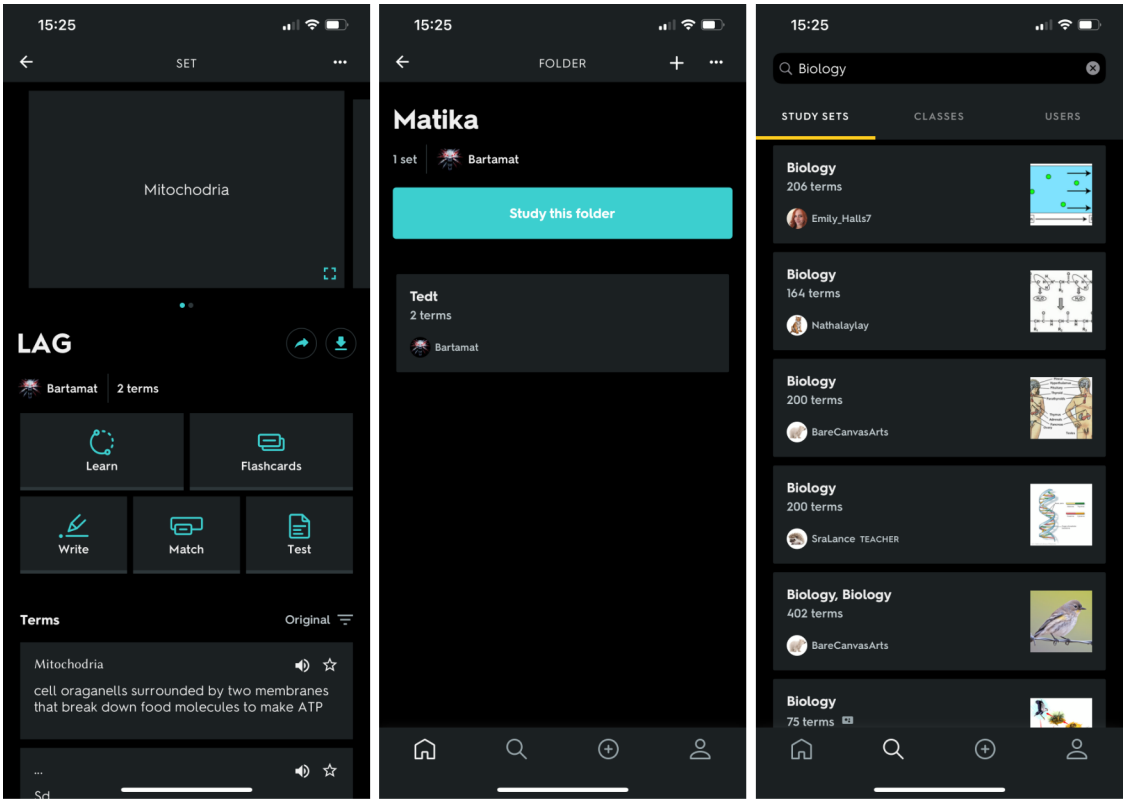
Na obrázku 2.2a v obrazovce na levo, je vidět logo a název kategorie. Tato obrazovka se zobrazí na začátku každého kola. Uživatel je tímto informován, z jaké kategorie budou následující otázky. Prostřední obrazovka reprezentuje kvíz samotný. Tři linky v levém rohu znázorňují skóre uživatele v daném kole. Šedá barva znamená, že otázka ještě není zodpovězena, červená špatný výsledek, zelená správný. Bílé bloky, které představují odpovědi svou kontrastní barvou vyčnívají a poutají na sebe pozornost. Všimněme si také spodní lišty. Ta se s postupem času rozšiřuje a mění barvu čímž signalizuje zbývající čas na zodpovězení otázky.

2.1.2 Quizlet

[2] Další aplikace, stojící za zmíňku patří do kategorie vzdělání. Aplikace byla v počátku napsána Andrew Sutherlandem roku 2005 jako pomůcka pro hodiny

[2] `https://quizlet.com`

francouzštiny.



(a) : Obrazovky zleva: Sada, Složka, Vyhledávání sad ostatních uživ.

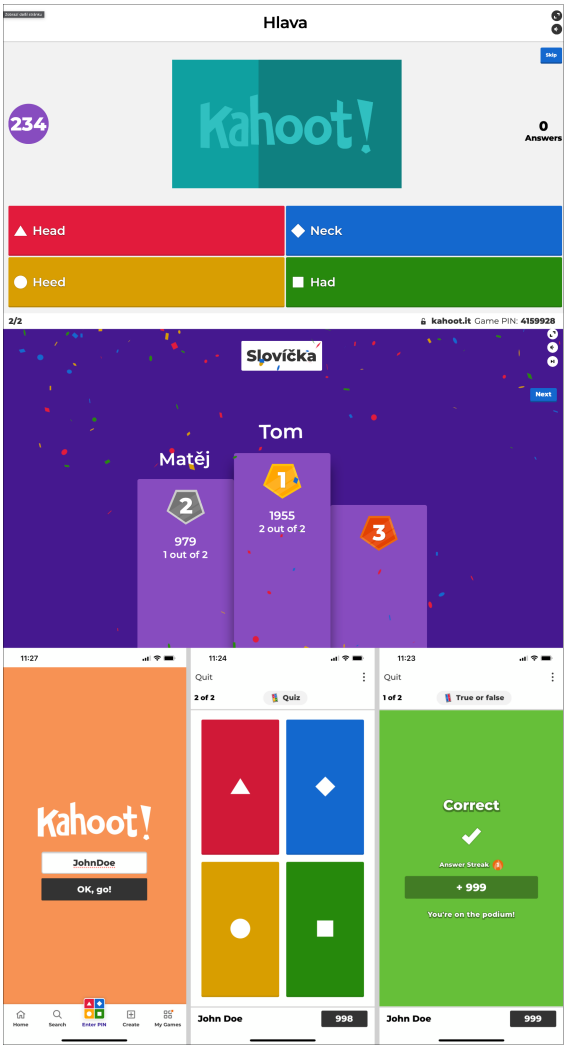
Obrázek 2.3: Quizlet

Na této aplikaci viz. obrázek 2.4a je zajímavý princip kategorizace a sdílení termínů. Uživatel vytváří sady termínů a jejich definic, ty může následně ukládat do svých vlastních složek či tříd. Výhoda pro ostatní, ale nepříjemnost pro tvůrce sady je její sdílení se všemi uživateli. To se sice dá změnit, ale tato akce není příliš intuitivní. Všechny sety jsou tak v podstatě dostupné všem, což vede k obrovskému množství již vytvořeného obsahu.

2.1.3 Kahoot!

[3] Posledním tématem této analýzy je aplikace Kahoot! Ta vznikla roku 2012 jako společný projekt tří studentů Norské Univerzity Vědy a Technologií. Tato aplikace se používá převážně k výuce a proto je ve velkém měřítku využívána ve školách.

[3] `https://kahoot.com`



(a) : Obrazovka shora: Položená otázka (zobrazena pouze na hl. monitoru), výsledky kvízu (také hl. monitor), dále zleva : Volba jména, volba odpovědi, vyhodnocení odpovědi

Obrázek 2.5: Kahoot!

Právě princip a vzhled této aplikace je důvod proč se stala tak populární. Učitelé mohou oživit a přimět studenty aby se zapojili do výuky, pomocí kvízů. Z obrázku 2.6a je vidět, že aplikace má jednoduchý, neformální až by se dalo říci komický vzhled. Studenti se připojí přes svá zařízení a volí si svá jména což navozuje pocit anonymity a student tak necítí tlak ostatních, když odpoví špatně. Zároveň, je ale celý kvíz soutěž, což vyvolává soutěživost a studenti se tak více snaží.

2.1.4 Shrnutí

Z předchozího textu jsem zjistil několik věcí. Hlavním tahounem většiny aplikací je jejich vzhled. U něho je důležité aby zůstal jednoduchý a intuitivní. Přecpaný a zmatený vzhled může mít nejlepší funkce na světě, ale uživatel po několika minutách ztrácí zájem.

Další částí je obsah. U kvízových aplikací platí pravidlo čím více, tím lépe. Uživatel nechce odpovídat na stejné otázky pořád dokola. To se dá vyřešit hned třemi způsoby. První, pro vývojáře ne zrovna příjemný způsob, je neustálá tvorba obsahu. Tím sice zaručíme kvalitu, ale čas je pro programátora cenná komodita. Druhý způsob, je nechat uživatele vytvářet obsah a ten sdílet s ostatními. Kvalita sice nebude zajištěna, ale kvantita bude růst s počtem uživatelů. Poslední způsob je vytvořit automatizovaný systém pro generování obsahu. Vývojář tak může sám ovládat kvalitu a kvantitu. Cena toho způsobu je pak čas a schopnost vyvinout takový systém. Při velkém množství obsahu je také významná kategorizace. Nekategorizovaný obsah by mohl vyvolat změť nepořádku.

V poslední části je důležitá myšlenka aplikace. Ta by měla být cílena na určitou skupinu a mít nějaký účel. V našem případě je cílová skupina rodič s dítětem v první / druhé třídě základní školy, tedy ve věku kdy se učí číst. Z toho lze usoudit, že přidávat časový limit do kvízu by mohlo vést k frustraci dítěte. To by muselo závodit s časem a tak by při čtení mohl přeskakovat písmenka a domýšlet odpovědi. Účelem je tedy vzdělání a zábava.

2.2 Požadavky

Na základě získaných informací z analýzy existujících řešení 2.1, jsem sepsal seznam požadavků, ty jsou pro větší přehlednost rozděleny do dvou kategorií:

2.2.1 Funkční požadavky

1. **Registrace do systému**
   Aplikace umožní registraci uživatele za pomoci osobního emailu.
2. **Přihlášení do systému**
   Aplikace umožní přihlášení uživatele za pomoci osobního emailu.

3. **Odhlášení ze systému**
   Aplikace umožní odhlášení uživatele ze systému.

4. **Resetování zapomenutého hesla**
   Aplikace umožní resetovat heslo uživatele, zasláním odkazu na email.

5. **Vyhledávání uživatelem vytvořených kvízů**
   Aplikace umožní přihlášenému uživateli vyhledávat kvízy, které vytvořil a to pomocí textového vstupu

6. **Seřadit kvízy**
   Aplikace umožní přihlášenému uživateli seřadit kvízy, které vytvořil podle názvu, času poslední úpravy a velikosti přes dropdown menu.

7. **Sdílet kvíz**
   Aplikace umožní přihlášenému uživateli sdílet své kvízy s ostatními uživateli.

8. **Hodnotit kvíz**
   Aplikace umožní přihlášenému uživateli, který není autorem kvízu ohodnotit kvíz na škále 0-5.

9. **Vytvoření nového kvízu**
   Aplikace umožní přihlášenému uživateli vytvoření nového kvízu.

10. **Smazání kvízu**
    Aplikace umožní přihlášenému uživateli smazat kvíz, který vytvořil.

11. **Vytvoření otázky pro kvíz**
    Aplikace umožní přihlášenému uživateli vytvořit novou otázku v kvízu, kde je autorem. K tomu je potřeba obrázek, který lze vybrat z knihovny v telefonu nebo vyfotit, dále pak jedna správná a dvě špatné odpovědi.

12. **Generování odpovědí pro otázku**
    Aplikace vygeneruje odpovědi při tvorbě / editaci otázky.

13. **Smazání otázky**
    Aplikace umožní přihlášenému uživateli smazat otázku.

14. **Editace otázky**
    Aplikace umožní přihlášenému uživateli editovat obrázek a odpovědi otázky.

15. **Vyhledávání otázky**
    Aplikace umožní přihlášenému uživateli vyhledávat otázky listováním v sadě.

16. **Seřazení otázky**
    Aplikace umožní přihlášenému uživateli seřadit otázky podle názvu a času poslední úpravy.

17. **Spuštění kvízu**
Aplikace umožní přihlášenému uživateli spustit kvíz.

18. **Plnění kvízu**
Aplikace umožní přihlášenému uživateli po spuštění plnit kvíz. Uživatel odpovídá na otázku jednou ze tří odpovědí. Při správném výběru odpověď zezelená, při špatném zčervená a správná zezelená. Po zodpovězení všech otázek nastává **Vyhodnocení kvízu**.

19. **Vyhodnocení kvízu**
Aplikace po zodpovězení zobrazí statistiky s počtem správných a špatných odpovědí.

20. **Žebříček kvízů**
Aplikace zobrazí žebříček nejlépe hodnocených kvízů.

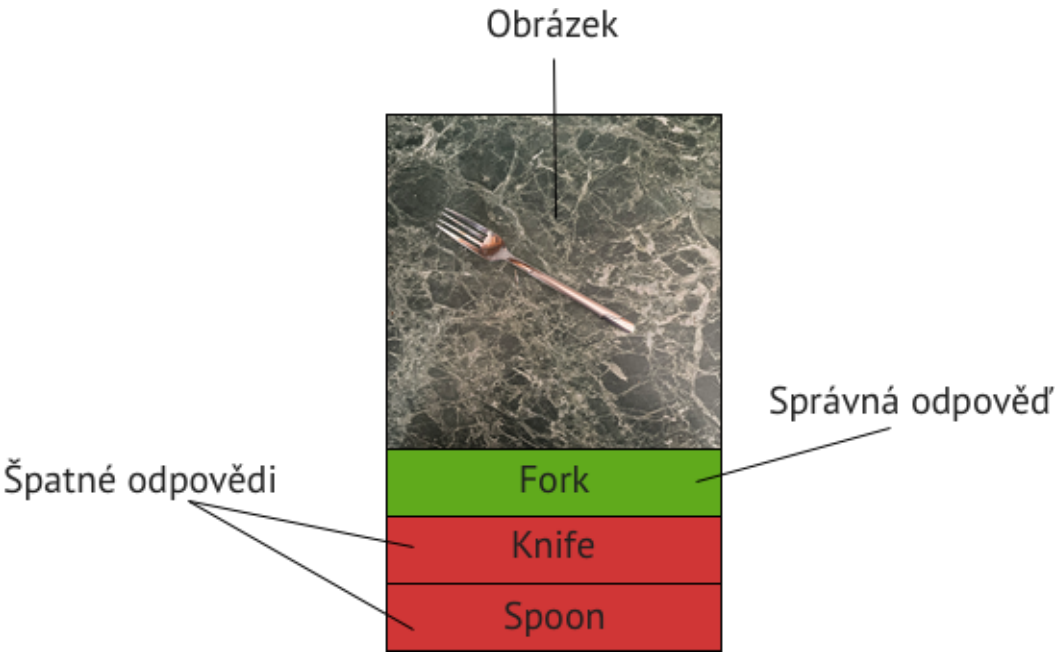
2.2.2 Nefunkční požadavky

1. **Stabilita systému**
Aplikace bude stabilní a nebude zbytečně zatěžovat mobilní zařízení.

2. **Responzivita systému**
Aplikace bude responzivní pro mobilní zařízení v rozmezí velikostí iPhone SE a iPhone 12 Max.

2.3 Metody pro generování otázek

Tvorba kvízu je nedílnou součástí aplikace. Je proto důležité vybrat vhodnou metodu, která by urychlila vytváření kvizových otázek. Otázka by se měla skládat z obrázku a jedné správné odpovědi. Ta popisuje co na obrázku doopravdy je a dvou špatných odpovědí, které souvisí se správnou odpovědí. Ukázková otázka znázorněna na obrázku 2.7.

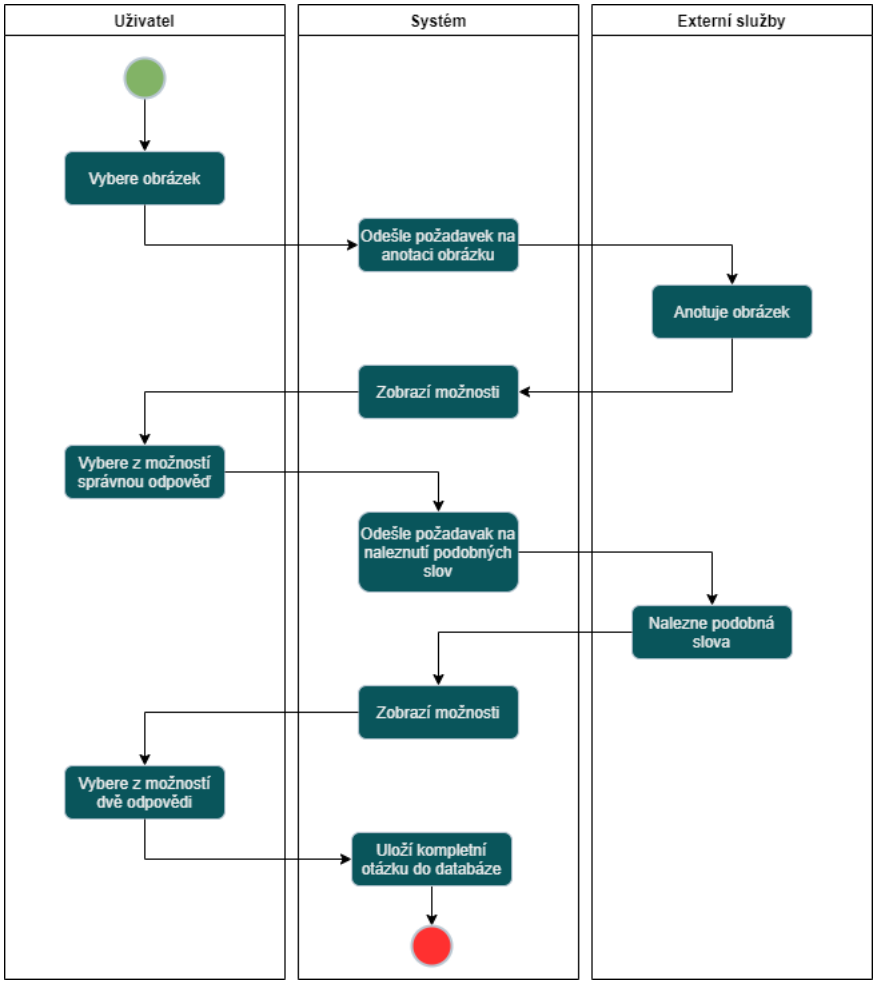


Obrázek 2.7: Ukázková kvizové otázka

Při hledání jsem přišel s dvěma relativně podobnými metodami. Metoda postupného doplňování 2.8 se od metody používající pouze obrázek 2.9 odlišuje interakcí s uživatelem. Použitím interakce uživatel získá kontrolu nad kvalitou otázky, ale proces to znatelně zpomalí. Kromě toho má uživatel také možnost otázky vytvářet bez použití generování. Z toho důvodu jsem se rozhodl použít metodu používající pouze obrázek pro generování otázky.

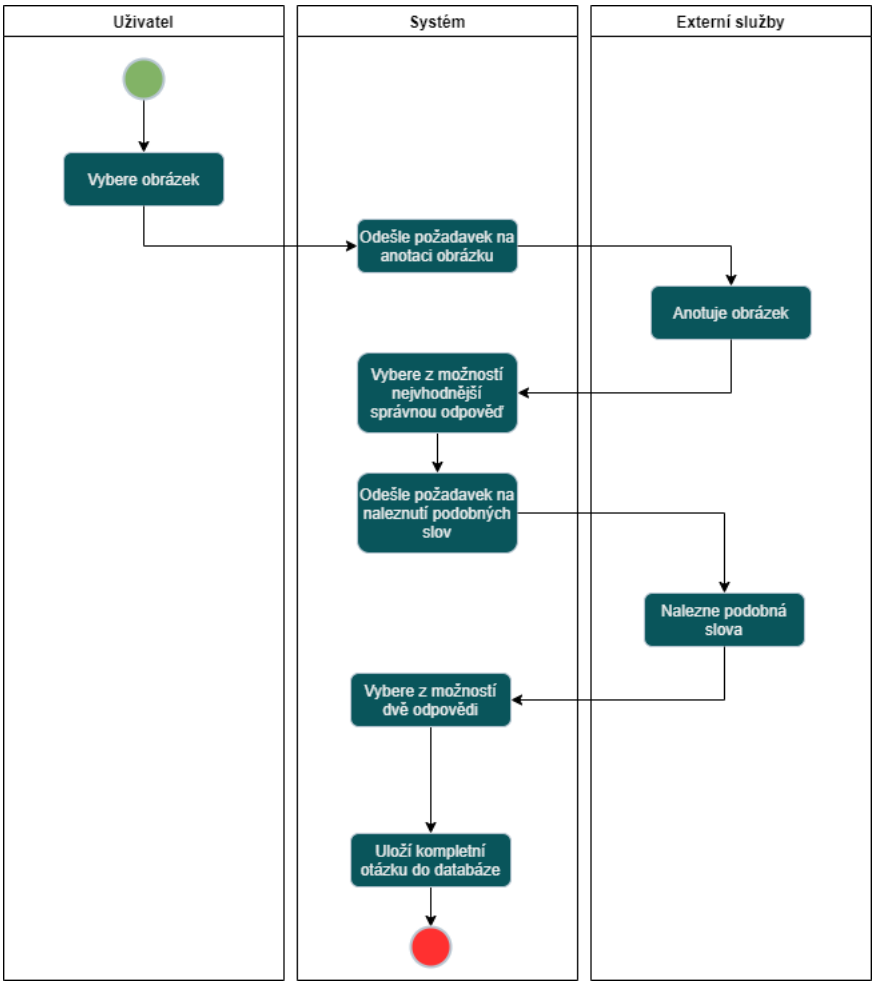
Metody jsou na následujících obrázcích 2.8 a 2.9 popsané pomocí procesních UML diagramů.

2.3.1 Metoda - Postupného doplňování



Obrázek 2.8: Procesní diagram metody postupného doplňování

2.3.2 Metoda - Pouze používající obrázek



Obrázek 2.9: Procesní diagram metody pouze používající obrázek

2.4 Knihovny pro anotaci obrázku

Důležitou částí tohoto projektu je implementace vhodné knihovny pro anotaci obrázků. Při výběru jsem se soustředil na několik faktorů. Prvním a z mého pohledu i tím nejdůležitějším je obtížnost implementace. Jelikož oblast strojového učení je pro mě neznáma, tak je důležité, aby využívání knihovny nevyžadovala tvorbu vlastního modelu, složitou implementaci a nepřehlednou dokumentaci. Druhý faktor je pak funkcionalita. Zde je potřeba, aby knihovna obsahovala kvalitní a rozsáhlý image classification, ten je zásadní pro funkční polo-automatizaci vytváření otázek. Třetím faktorem je pak dostupnost. Vzhledem k tomu, že se jedná o studentský projekt tak hledáme

knihovnu, která je cenově dostupná.

2.4.1 TenserFlow

[12] Tato open-source knihovna od Googlu, patří mezi jednu z nejpoužívanějších knihoven pro strojové učení. Poskytuje celý balíček pro tvorbu a trénování modelu a jeho implementaci.

Specificky pro image labeling na iOS poskytuje on-device computation knihovnu s několika před trénovanými modely. Ty jsou schopny rozpoznat 1000 různých objektů s až 90% přesností. Co se týče dokumentace, ta je poměrně strohá a mířena spíše na uživatele, kteří v oboru strojového učení už mají nějaké znalosti.

2.4.2 CoreML

[18] Open-source knihovna od applu, poskytující stejné funkce jako předchozí TenserFlow s tím rozdílem, že se zaměřuje pouze na on-device computation pro zařízení od apple. Tím má lépe optimalizovaný výkon zařízení a spotřebu energie.

Pro image labeling poskytuje stejné před trénované modely jako TensorFlow, takže schopnost v rozpoznávaní modelů je stejná (1000 různých objektů s až 90% přeností). Dokumentace je dostatečně detailní. Apple navíc poskytuje několik výukových videí.

2.4.3 Cloud Vision

[11] Jak už z názvu vyplývá, jedná se o cloudovou knihovnu zabývající se vizuálním rozpoznáváním. Ta je součástí masivní cloudové platformy od Google.

Rozdíl ve funkcionalitě nastává na první pohled. Cloud Vision nabízí vysokou přesnost v schopnosti rozpoznat přes 10 000 objektů. Navíc Google je znám pro své dokumentace a pro implementaci používá REST api, není tak vyžadována žádná znalost v oboru strojového učení. Google, stejně jako Apple poskytuje výuková videa. Nutno podotknout, že celá služba je zpoplatněna, s výjimkou prvních 1000 výpočtů za měsíc.

2.4.4 Shrnutí

V předchozích sekcích jsem uvedl tři různé nástroje pro strojové učení. Dva z nich (TenserFlow 2.4.1 a CoreML 2.4.2), fungovali spíše jako prostředek pro vývoj a učení vlastních modelů. Oba nabízeli stejné před trénované modely, ale jejich funkcionalita se nemohla rovnat cloudovému výpočtu.

V momentě kdy se podíváme na rozsáhlost v rozpoznávání objektů je tedy jasným favoritem Cloud Vision (2.4.3). Její implementace je poměrně snadná a díky cloudovému výpočtu není potřeba řešit ani optimalizaci výkonu. Jediný problém nastává při ceně služby, jelikož se však jedná o studentský projekt tak 1000 výpočtů měsíčně mi přijde jako relativně dostačující.

2.5 Nástroje pro hledání podobných slov

Posledním krokem pro generování otázky, bylo najít slova související se správnou odpovědí (výsledkem anotace obrázku). Prvotní nápad bylo využít ostatní možnosti z anotace. Po krátkém testování jsme se s vedoucím práce shodli, že výsledky jsou nedostačující, protože odpovědi ve vícero případech nekorelovaly se správným výsledkem. Dalším nápadem tak bylo znovu využít sílu strojové učení. Bohužel jsem nebyl schopen najít knihovnu, která by jedno slovo kategorizovala a vrátila slova podobná. Při hledaní jsem ovšem přišel na knihovny využívající search engine, které mě posléze přivedli k slovníkovým API.

Ty jsou schopna dodat fakta související se slovy, jako například: synonyma, rýmy, kategorie, frekvenci slov a mimo jiné slova související. Nutno podotknout, že ne všechny tyto API tuto funkci obsahovali. Výhodou je také, že většina těchto API využívá RESTové rozhraní, takže jejich použití je poměrně snadné. Jelikož na výběr bylo poměrně z velkého množství a nebylo možné poměřovat jejich funkcionalitu, rozhodl jsem se tedy vybrat DatamuseAPI [7], které nevyžadovalo API klíč a bylo zcela zdarma. Za zmínku, ale stojí například WordsAPI [25].

Kapitola 3

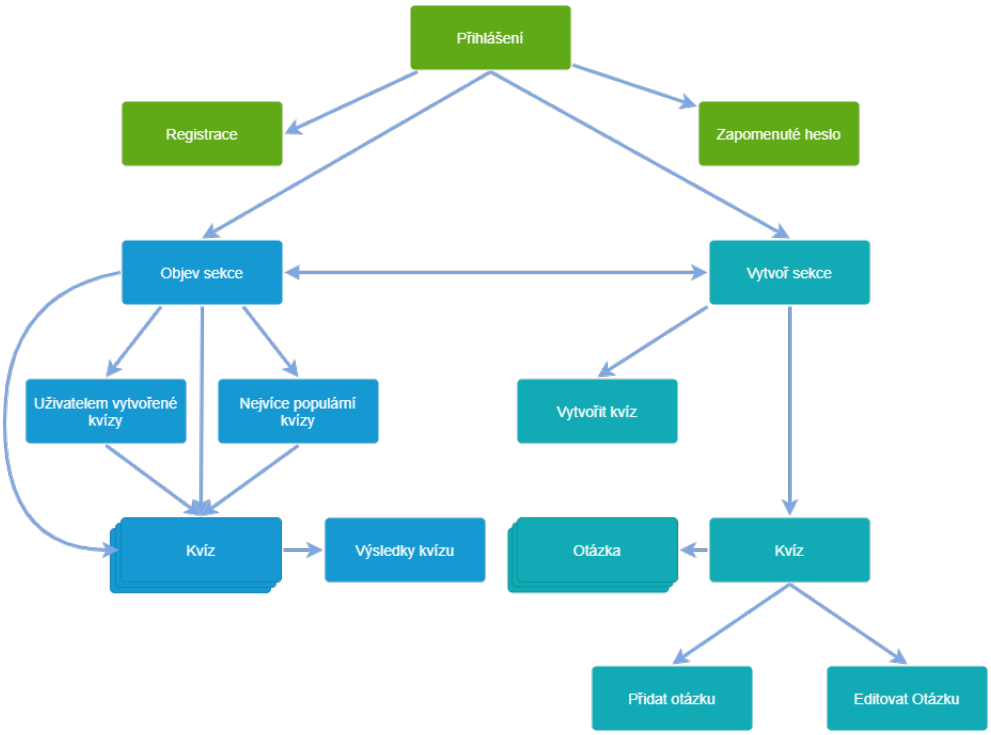
Uživatelské rozhraní

Podstatnou částí každé aplikace je její uživatelské rozhraní. Aby aplikace poskytovaly pozitivní uživatelský zážitek, byly vytvořeny sady pravidel a principů - guidelines, které pomáhají k vytvoření úspěšných designů.[8] Protože je projekt cílen na platformu iOS, řídil jsem se při navrhovaní podle Human Interface Guidelines[1].

3.1 Navigace

Uživatelsky nedoceněnou, ale podstatnou součástí funkčního rozhraní, se často stává navigace. Chaotické přechody mezi obrazovkami mohou způsobit uživatelsky nepříjemný zážitek. Je tak důležité vytvořit strukturu, která nebude nijak upozorňovat a zároveň bude povědomá. Pro lepší orientaci byla zhotovena mapa obrazovek 3.1, kde jsem zkombinoval hierarchickou navigaci s navigací plochou.[2]

[1] Guidelines pro všechny plaftormy od společnosti Apple (iOS, macOS, watchOS, tvOS)



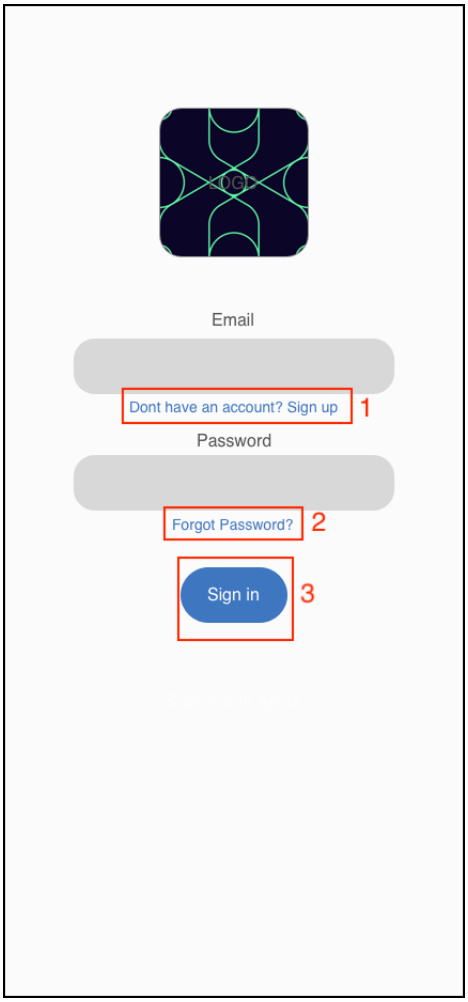
Obrázek 3.1: Mapa obrazovek

3.2 Nízko úrovňový prototyp

Pro tvorbu prototypu jsem vybral nástroj Sketch[2]. Ten se bohužel ukázal jako poměrně nešťastný, jelikož se zaměřuje spíše na vysoko úrovňové prototypy. Tudíž při tvorbě prototypu jsem místy dával příliš velký důraz na detaily což je vidět na finálním prototypu [3].

[2] https://www.sketch.com/
[3] Odkaz na prototyp zde

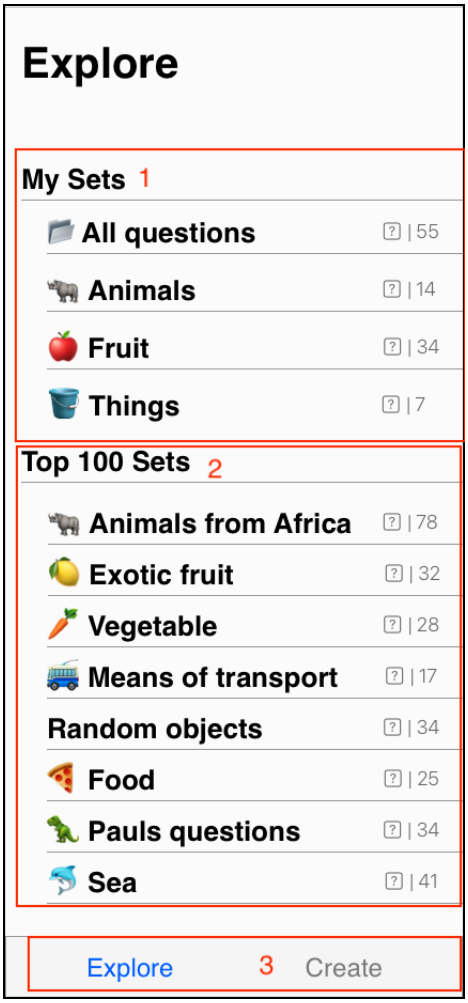
3.2.1 Přihlášení (viz 3.2)



Obrázek 3.2: Přihlašovací obrazovka

1. Tlačítko, které otevře modální okno pro registraci.
2. Tlačítko, které otevře modální okno pro resetování hesla.
3. Tlačítko pro přihlášení - uživatel musí zadat správný email a heslo.

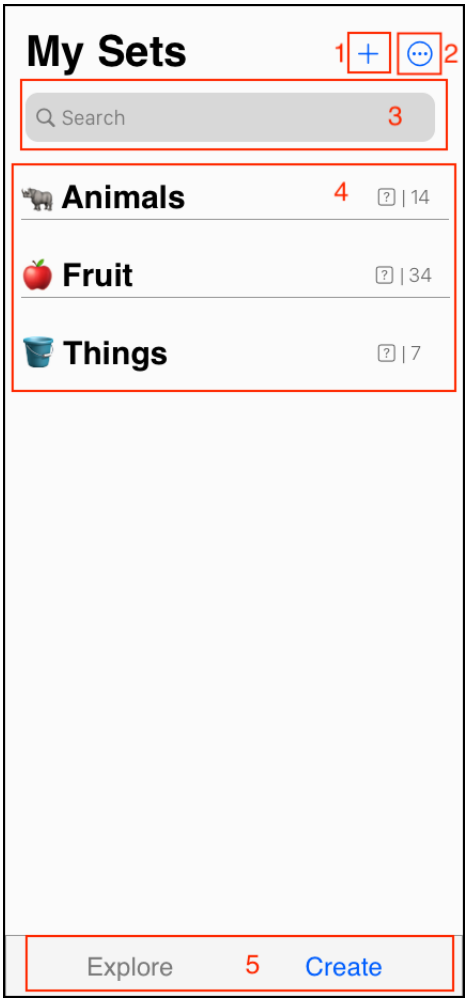
3.2.2 Explore sekce - (viz - 3.3)



Obrázek 3.3: Obrazovka explore sekce

1. List s uživatelem vytvořenými kvízy, kliknutím na jednotlivé položky spustí test.
2. List s nejvíce populárními kvízy, kliknutím na jednotlivé položky spustí test.
3. Tlačítka pro přepnutí mezi Explore sekcí a Create sekcí.

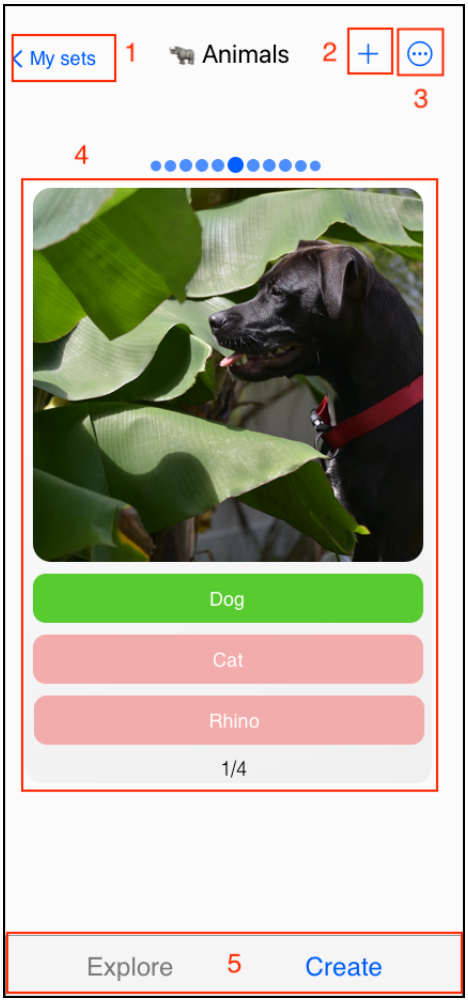
3.2.3 Create sekce - (viz - 3.4)



Obrázek 3.4: Obrazovka create sekce

1. Tlačítko, které otevře modální okno pro vytváření nové sady otázek (kvízu).
2. Tlačítko, které otevře menu s výběrem typu seřazení a možností odhlášení.
3. Textové pole pro vyhledávání mezi vlastními sadami (kvízy).
4. List s uživatelem vytvořenými sadami (kvízy), kliknutím na jednotlivé položky zobrazí detail sady (kvízu). Podržením prstu na názvu jednotlivé položky nabídne smazání sady (kvízu).
5. Tlačítka pro přepnutí mezi Create sekcí a Explore sekcí .

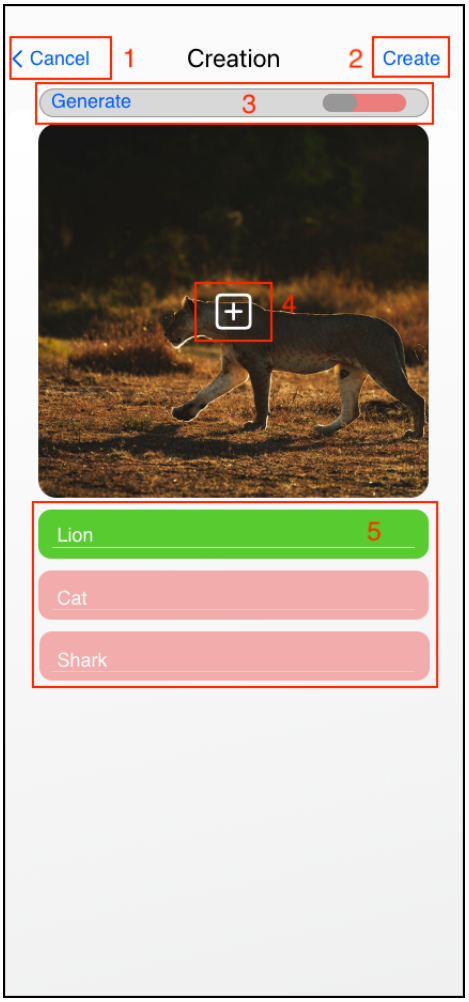
3.2.4 Detail sady (kvízu) - (viz - 3.5)



Obrázek 3.5: Obrazovka detail sady (kvízu)

1. Tlačítko, které přesměruje zpět do Create sekce.
2. Tlačítko, které otevře modální okno pro přidání otázky do aktuálně zobrazené sady.
3. Tlačítko, které otevře menu s výběrem typu seřazení a tlačítky pro editaci, sdílení a mazání aktuálně zobrazené otázky.
4. Aktuálně zobrazená otázka, kliknutím lze listovat mezi otázkami z aktuálně zobrazené sady.
5. Tlačítka pro přepnutí mezi Create sekcí a Explore sekcí .

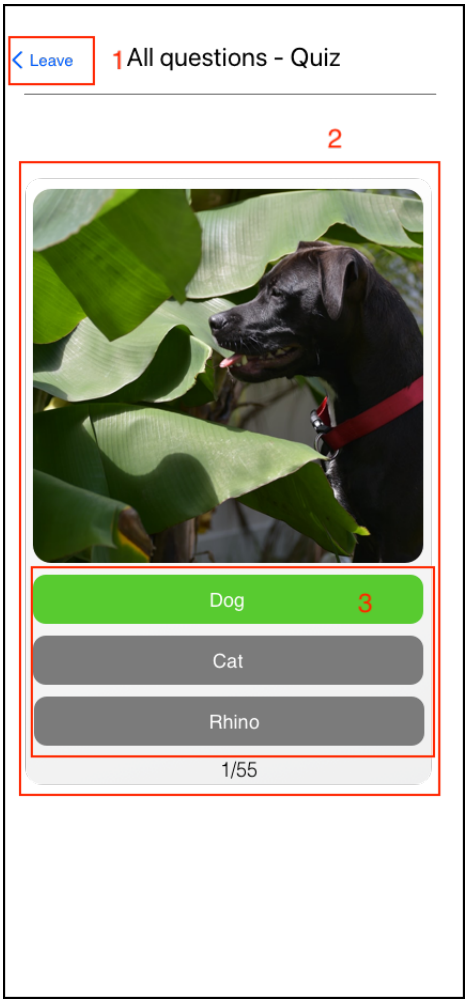
3.2.5 Přidání otázky - (viz - 3.6)



Obrázek 3.6: Obrazovka přidání otázky

1. Tlačítko, které zruší proces vytváření otázky.
2. Tlačítko, které spustí tvorbu otázky.
3. Přepínač pro zapnutí nebo vypnutí generování otázky.
4. Tlačítko, které otevře menu s typem medií pro výběr obrázku (Knihovna, Fotoaparát).
5. Textová pole: Zelená barva pro správnou odpověď, červená barva - špatné odpovědi.

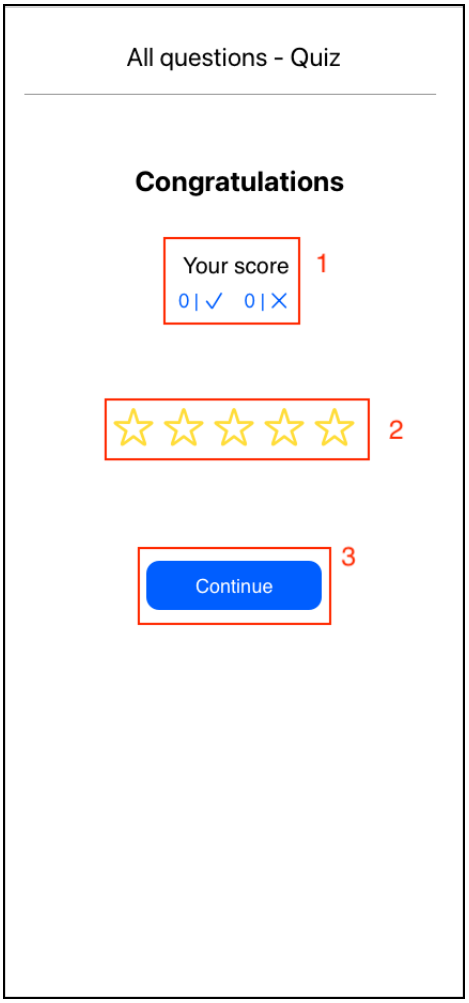
3.2.6 Kvíz - (viz - 3.7)



Obrázek 3.7: Obrazovka kvízu po vybrání správné odpovědi

1. Tlačítko, které zruší vypracovávání kvízu.
2. Aktuální otázka, pro přesunutí na další otázku musí uživatel vybrat jednu z odpovědí a po vyhodnocení kliknout kdekoliv v oblasti otázky.
3. Tlačítka pro výběr odpovědi, při výběru správné otázky tlačítko zezelená. Při výběru špatné zčervená a správná odpověď zezelená.

3.2.7 Vyhodnocení kvízu - (viz - 3.8)

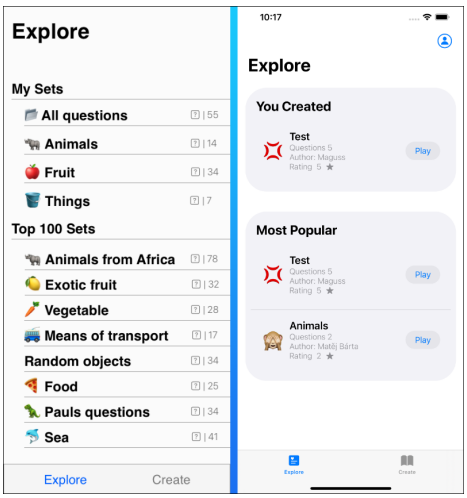


Obrázek 3.8: Obrazovka vyhodnocení kvízu

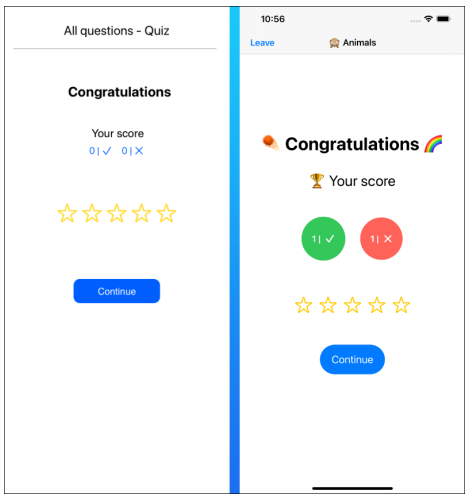
1. Počet správně a špatně zodpovězených otázek.
2. Pokud uživatel není autorem kvízu, může kvíz hodnotit na škále 0-5.
3. Tlačítko pro ukončení kvízu.

3.3 Finální verze

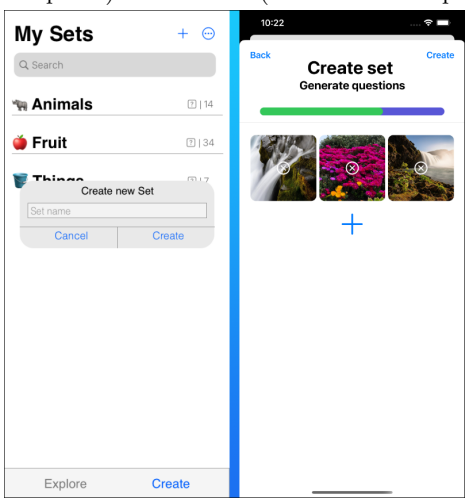
Nutno podotknout, že prototyp se značně liší od finálního produktu. To je důsledkem několika připomínek z řad kolegů, rodiny a vedoucího práce, kteří vývoji přihlíželi a videa [4] zveřejněno po vytvoření prototypu. Zároveň za to může povaha implementace, která se točila kolem nástroje použitého pro její tvorbu. Největší změnou prošlo pravděpodobně vytváření nové sady (kvízu). Ta je interaktivnější a dává uživateli možnost nahrát vícero obrázků najednou (otázky se po přidání vygenerují z obrázků). Porovnání několika obrazovek z prototypu a finální verze je k vidění na obrázku 3.9.



(a) : Explore sekce
(Finální verze napravo)



(b) : Vyhodnocení kvízu
(Finální verze napravo)



(c) : Vytváření nové sady
(Finální verze napravo, krok 3 ze 3)

Obrázek 3.9: Porovnání prototypu a finální verze uživatelského rozhraní

Kapitola 4

Technologie

Tato kapitola slouží jako přehled použitých nástrojů a technologií s jejich krátkým popisem.

4.1 Nástroje

4.1.1 XCode

Officiální IDE[1] od společnosti Apple pro vývoj iOS aplikací. Kromě typických funkcionalit jako lokální buildování, debugger či napojení na verzovací systémy obsahuje vestavěný simulátor všech zařízení od společnosti Apple. S příchodem SwiftUI nabízí takzvané **Previews**. Ty umožňují zobrazit změny v uživatelského rozhraní v reálném čase, bez toho aniž by bylo potřeba projekt zkompilovat. Jako verzovací systém jsem zvolil školní Gitlab[2].[1]

[1] *"Software pro vytváření aplikací, který kombinuje běžné vývojářské nástroje do jediného grafického uživatelského rozhraní."[14]*

[2] Odkaz na projekt - `https://gitlab.fel.cvut.cz/bartamat/quizit`

4.1.2 TestFlight

Služba od společnosti Apple pro beta testování iOS aplikací. Vývojář může určit až 100 členů z týmu pro interní testování nebo až 10 000 externích uživatelů pomocí emailu, či veřejného odkazu. Jediné co je potřeba od testera je stáhnutí aplikace TestFlight z App Storu. Služba zaznamenává veškeré pády aplikace a zpětnou vazbu od testerů. Pro využívání této služby musí být vývojář členem Apple Developer Programu[3]. I když se tento projekt nezabývá testováním aplikace, využil jsem tuto službu k produkci.[3]

4.1.3 Firebase

Platforma od společnosti Google, doporučena vedoucím práce, pro vývoj mobilních a webových aplikací. Ta poskytuje vysokou škálu služeb. Od produktů zaměřených na vývoj a testování, přes autentizaci, až po dokumentové databáze a cloudové funkce. V projektu platformu využívám jako serverovou část. Její služby jsou sice zpoplatněny, ale při minimálním využívání je vše zdarma.[10]

V projektu využívám tyto produkty:

- Authentication - Pro autentizaci uživatele
- Cloud Firestore - Pro ukládání dat o uživateli a kvizových sadách
- Cloud Storage - Pro ukládání obrázků z otázek
- Cloud Functions - Pro volání funkce anotující obrázek

4.2 Technologie

Jak již píši v úvodu práce, vývoj aplikace se odehrával na platformě iOS a soustředí se na využití frameworků SwiftUI a Combine. Z toho vyplývá, že se jedná o nativní aplikaci psanou v jazyce Swift.

[3] https://developer.apple.com/programs/

4.2.1 Swift

Multi-paradigmatický, silně typový programovací jazyk představený roku 2014 založený na jazyce Objective-C, s úmyslem nahradit Cčkové jazyky (Objective-C, C++, C). Jeho vývoj se řídí třemi pravidly: Bezpečnost, Rychlost a Expresivita. Vývojáři se snaží syntaxí co nejvíce napodobit anglický jazyk, což vede k tomu, že syntaxe je jednoduchá na čtení i psaní. Součástí jazyka je garbage collector takže není potřeba se starat o paměť.

Za zmínku také stojí funkce jako pojmenované parametry, více návratových hodnot nebo optionals. Jelikož swift defaultně nedovoluje objektům nulový datový typ (objekt neprojde skrz kompilaci), zvyšuje tím bezpečnost kódu. Občas jsou, ale takové typy potřeba a k tomu slouží optionals, které umožňují bezpečné zacházení.[16][17]

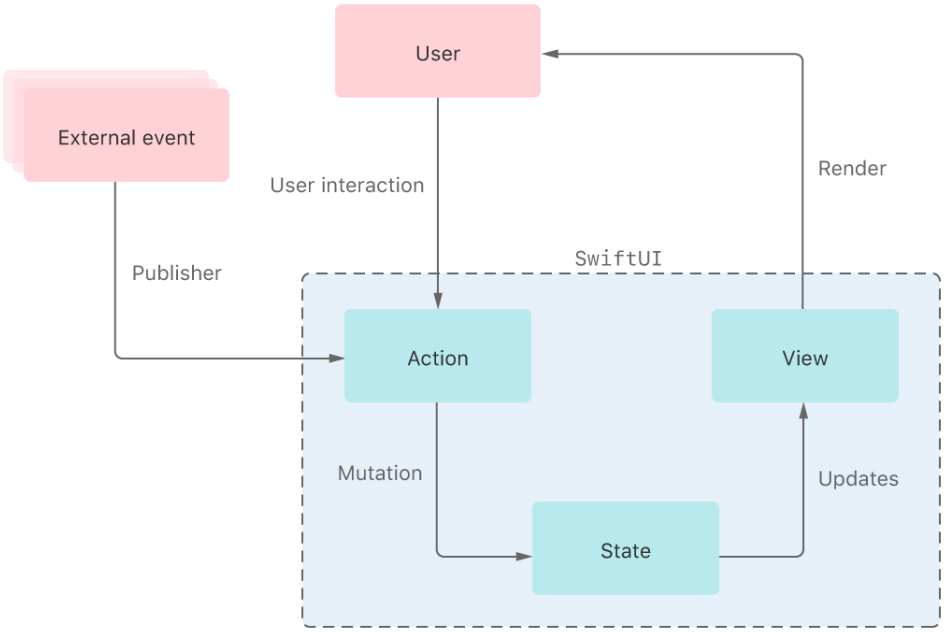
4.2.2 SwiftUI

Sada nástrojů pro tvorbu uživatelského rozhraní. Oproti UIKitu je díky své deklarativní syntaxi přehlednější a snazší na implementaci. Ta probíhá skládáním a kombinováním stavebních bloků (**Views**[4], **Controls**[5]) s **container strukturami**[6] a následného modifikování jejich vzhledu či funkčnosti pomocí speciálních funkcí: **modifikátorů**. [20]

[4]Těmi jsou například texty, obrázky nebo různé tvary.

[5]Tlačítka, přepínače, odkazy a mnoho dalšího.

[6]Vertikální nebo horizontální zásobníky, skupiny a podobné views, která zastávají funkci kontejneru.



Obrázek 4.1: Diagram popisující jak SwiftUI reagující na externí události a vstupy uživatele, obrázek převzat z oficiální dokumentace. [21]

Další z výhod je také nastavení datové závislosti u jednotlivých Views. Ta zaručí, že pokud dojde ke změně dat, změní se i View (4.1). Ke kontrolování změn se používají property wrappers. Lokální změny používají **State** a **Binding**. Změny dat v referenčním modelu používají **ObservedObject** či **StateObject**. Pro distribuci dat skrze aplikaci lze využít **Enviroment**, do kterého se nejprive uloží data. Pro přístup a manipulaci se používá **EnvironmentObject**. Data z referenčního modelu musí splňovat protokol **ObservableObject**. [21]

SwiftUI plně komunikuje s UIKit, AppKit a WatchKit frameworky.

4.2.3 Combine

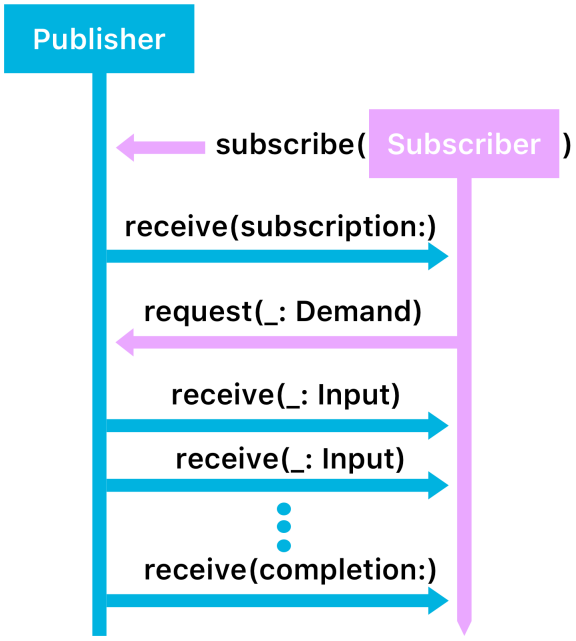
Poměrně komplexní swiftový framework zaměřený na asynchronní práci s hodnotami. Deklaruje protokoly **Publisher** a **Subscriber**. **Publisher** dodává hodnoty v průběhu času, skládá se z dvou asociovaných typů (**Output**[7] a **Failure**[8]) a na základě událostí může generovat:

[7] Typ hodnoty, kterou publisher vrací.
[8] Typ chyby, na kterou může publisher narazit.

- Výstupní hodnotu typu **Output**
- Chybu typu **Failure**
- Úspěšné dokončení

Na **Publishery** lze aplikovat **operátory**[9]. Ty pracují s jejich hodnotami a je možno je řetězit, svou funkcionalitou tak připomínají streamy.

Na konci tohoto řetězce **Subscriber** přijímá data. Ten se musí skládat ze stejných asociovaných typů (**Output** a **Failure**) jako **Publisher**, který vydává hodnoty pokud o to **Subscriber** explicitně požádá. Tento proces je znázorněn na diagramu 4.2.



Obrázek 4.2: Diagram popisující komunikaci mezi Publisher a Subscriber, obrázek převzat z článku o frameworku Combine.[24]

Mimo jiné Combine poskytuje vestavěné **Publishers**, které lze připnout na většinu proměnných, pomocí anotace **@Published**. Toho se ve velkém využívá ve SwiftUI. [19][24]

[9]Například pro úpravu, odstranění nebo přidání dat

4.2.4 WWDC21

Jelikož jsou všechny použité technologie od společnosti Apple, stojí za zmínku jejich každoroční konference pro vývojáře - World Wide Developer Conference (zkráceně WWDC). Ta se tento rok konala 3.6 a přinesla plno změn právě pro asynchronní programování. Obrovskou novinkou pro Swift bylo představení asynchronních funkcí používajících klíčová slova **async** / **await** (použití velice podobné jako v javascriptu). To byl pro komunitu sice velice příjemný, ale šokující krok, vzhledem k nedávnému představení frameworku Combine. Zde se perfektně ukázala mentalita Swift vývojářů, kteří se nebojí neustále modernizovat.[5]

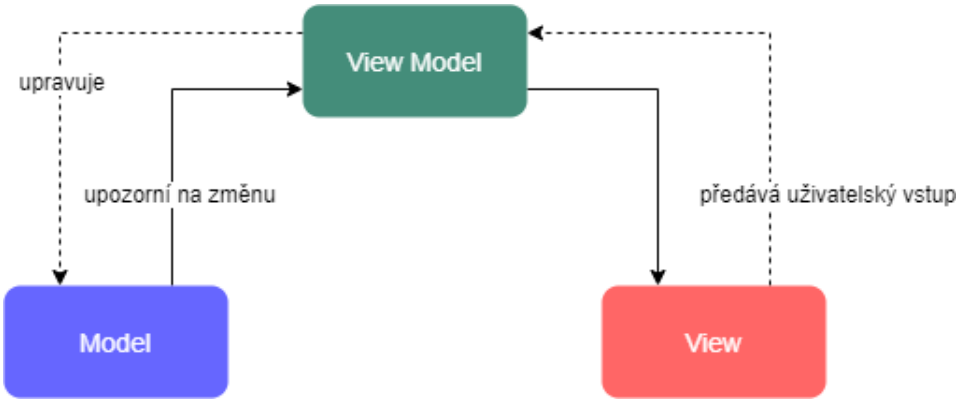
Mezi novinkami pro SwiftUI bylo například view pro načítání vzdálených obrázků (AsyncImage), přidání nových gest pro list nebo modifikátor pro implementování vyhledávání. Apple také prozradil, že pro vývoj svých aplikací používá SwiftUI. Je potřeba však podotknout, že všechny představené novinky jsou funkční teprve od iOS 15.[6]

4.2.5 Architektura MVVM

Architektonický vzor snažící se separovat logiku aplikace od uživatelského rozhraní (diagram 4.3). Jeho cílem je vytvoření větší přehlednosti a čitelnost kódu. Skládá se ze tří komponentů:

- **Model** Data a logika celé aplikace, nemá nic společného s uživatelským rozhraním.
- **View** Uživatelské rozhraní aplikace, deklaruje aktuální stav modelu pomocí ViewModelu a reaguje na vnější události.
- **ViewModel** Spojuje View s Modelem, dělá prostředníka.

Díky nástrojům, které SwiftUI poskytuje[10] využití MVVM architektury dává o to větší smysl.[22]



Obrázek 4.3: Diagram architektury MVVM, obrázek převzat z videa popisující MVVM ve SwiftUI. [22]

[10]Property wrappers 4.2.2

Kapitola 5

Implementace

Poslední částí tohoto projektu je implementace aplikace. V této kapitole tedy komentuji podstatné nebo zajímavé části, problémy na které jsem narazil a odůvodňuji rozhodnutí, která vedla k finální verzi.

5.1 Struktura projektu

Projekt je pro větší přehlednost strukturován do následujících skupin:

- **Work data** - Obsahují testovací data, převážně využívané pro zobrazování Previews
- **Extensions** - Swift umožňuje rozšiřovat funkcionalitu již existujícím třídám, strukturám a protokolům. Zde se nachází rozšíření pro třídy Array a View.
- **Services** - Veškeré externí služby využívané ve ViewModelech. Od firebase storage a firestore database po google vision a datamuse api.
- **App** - Obsahuje jediný soubor, který deklaruje aplikaci a konfiguruje firebase
- **Models** - Obsahuje struktury, které reprezentují data. Na tyto struktury jsou následně mapovány dokumenty uložené v firestore databázi.

- **ViewModels** - ViewModely pro jednotlivá Views, neobsahují žádné reference na Views
- **Views** - Obsahuje obrazovky a modální okna
  - **Styles** - SwiftUI umožňuje vytváření vlastních stylů. Zde se nachází styly pro tlačítka a textová pole.
  - **Components** - Obsahuje komponenty z kterých se skládají některá Views

5.2 Asynchronní funkce

Aplikace obsahuje plno procesů, které stahují a odesílají různé typy dat. Tyto procesy jsou mnohdy asynchronní povahy a je tak potřeba zvládat jejich chování. Firebase SDK ve svých funkcí používá **closure s escaping completion handlerem**[15] (ukázka na funkci načítající populární kvízy C.2). To je poměrně jednoduché na použití, ale při neopatrném chování může vést k problémům typu **Pyramid of Doom**[1]. Pro zachování pomyslné jednoty tento princip používám i v ostatních částech, přestože existuje alternativní řešení používající Combine.

Combine byl použit na propojení **Firestore SnapshotListeneru** (více o něm v sekci 5.4.2) na proměnnou ve ViewModelu. Nejdříve jsou data z listeneru uložena do proměnné v Set/Question Repository anotované jako **@Published**. Proměnná je ve ViewModelu pomocí několika operací přemapována a uložena do finální proměnné, která je poskytována View. Zároveň je její reference uložena do proměnné **AnyCancellable**, která při deinicializaci zruší všechny reference (ukázka mapování C.5).

5.3 SwiftUI

Programování uživatelského rozhraní bylo poměrně jednoduché, jelikož o rozmístění jednotlivých Views se stará samotné SwiftUI. To ve zkratce funguje následovně: Container Views nabízejí prostor pro Views. Ta si sama zvolí prostor, který potřebují. Container Views je poté umístí podle svých pravidel (HStack umísťuje horizontálně, VStack vertikálně, atd.)[23]. Deklarované View k vidění na ukázce C.6).

[1] Blok kódu, který obsahuje příliš vnořených odsazení, které ho dělají naprosto nečitelným.

5.3.1 Problémy

Problémy, většinou nastávaly při přidávání funkcionality nebo naprosté absenci daného komponentu. Na problémy s funkcionalitou jsem narazil například ve chvíli, kdy jsem chtěl přidat gesto (odstranění kvizové sady) na řádek v listu (ten odkazoval na detail sady). Problém byl v oblastech, kde tlačítko reagovalo na jednotlivá gesta. Ve finále se mi podařilo, aby celá plocha řádku přesměrovala na detail a při delším kliknutí na název kvízu se spustil proces odstranění sady.

Největší problém nastal ve chvíli, kdy jsem se snažil implementovat fotoaparát a výběru fotek z knihovny telefonu. Zde jsem byl přinucen vytvořit si vlastní komponent za pomoci UIKitu, jelikož SwiftUI nic takového neposkytuje. K tomu jsem použil protokol **UIViewRepresentable**, který zaobalí UIKit view pro použití ve SwiftUI. V UIKit view jsem implementoval ViewController, který vrací **UIImagePickerController**. Na základě toho, zda uživatel vybral kameru či knihovnu se pak zobrazí korektní modální okno. Pro předávání dat a stavu slouží třída Coordinator, která implementuje protokoly **UIImagePickerControllerDelegate a UINavigationControllerDelegate**. Celý komponent ImagePicker je k vidění v ukázce C.7 a C.8.

5.4 Firebase

Serverovou částí je platforma Firebase. Ta obstarává autentizaci, spravuje data a slouží jako prostředník pro Google Vision API. Použití je poměrně jednoduché, stačí vytvořit projekt na jejich webových stránkách a nainstalovat firebase SDK do projektu v XCode.

5.4.1 Autentizace

Díky službě **Authentication**[2], kterou Firebase poskytuje, nebyla potřeba řešit logika ohledně registrace, přihlašování a ověřování uživatele. Z širokého výběru možných metod autentizace, jsem se rozhodl pro základní email a heslo.

ViewModel starající se autentizaci, drží data o uživateli a odpověď vracenou z funkcí pro přihlášení a registraci (ukázka funkce pro přihlášení C.4).

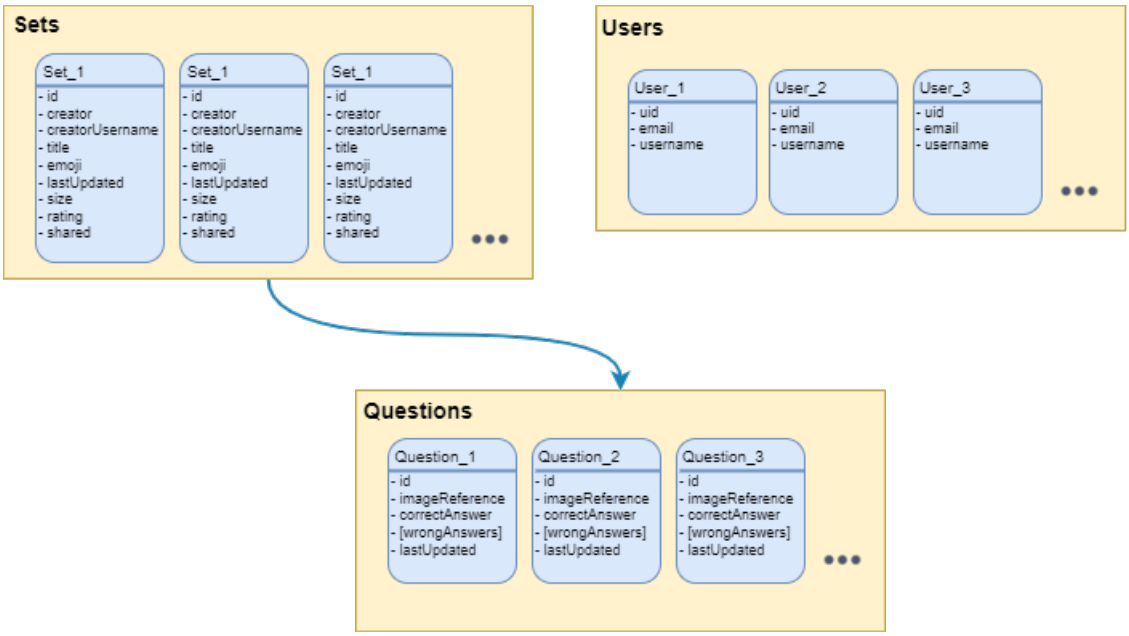
[2]https://firebase.google.com/docs/auth

Následně je uložen jako **enviromentObject** do životního cyklu aplikace, kde k němu přistupují Views jako LoginView či FinishedQuiz.

5.4.2 Správa dat

Firestore

Jednu z nejdůležitějších prací zastává služba **Firestore**[3]. Jedná se o dokumentově orientovanou databázi, ukládající data do dokumentů (velice podobná struktura JSON objektům), které jsou organizovány do kolekcí [13]. Diagram 5.1 popisuje datový model, použitý ve Firestoru. V ukázce C.1 je k vidění model aplikace, na který jsou mapovány dokumenty. Mapování nebylo nijak složité, díky **Codable** protokolu (mapování je k vidění v ukázce C.2).



Obrázek 5.1: Diagram datového modelu

Firestore API, nabízí kromě základních CRUD funkcí vylepšené čtení pomocí **SnapshotListeneru**. Ten naslouchá změnám v dokumentu nebo kolekci a automaticky posílá aktualizovaná data. Tuto funkci používám například v Create sekci a detailu kvizové sady, kde by uživatel měl mít neustále aktuální data. Načítání dat pomocí **SnapshotListeneru** je k vidění na ukázce C.3.

[3] https://firebase.google.com/docs/firestore

Nejvíce problémová byla funkce smazání celé kvizové sady i s otázkami. Firestore neumožňuje mazání celé kolekce a při mazání dokumentu s pod-kolekcemi, zanechá pod-kolekce nedotknuté (dokument se stále objevuje v konzoli, ale nezobrazí se ve výsledcích query). Jediný způsob jak smazat kolekci s otázkami, tak bylo manuálně načíst a smazat všechny dokumenty.

Firestore Storage

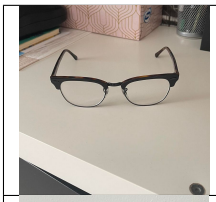
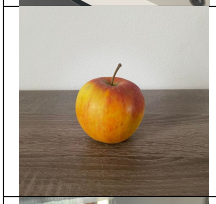
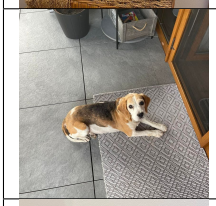
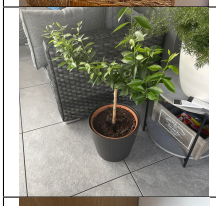
Stejně důležitá je i služba **Storage** [4], sloužící k ukládání fotek, videí a dalšího uživatelského obsahu [9]. Ta je použita k ukládání a načítání obrázků, které jsou součástí otázky. Funkce ukládající obrázek na server, při úspěšném uložení vrátí referenci (adresu) obrázku. Reference slouží k propojení otázky z Firestoru a obrázku z Storage. Při ukládání jsem využil officiální rozšíření **Resize Image**[5], které automaticky po nahrání obrázku vytvoří verzi velikosti 800x800.

5.5 Generování otázky

Poměrně komplexní proces složený z celkem pěti funkcí. Obrázek je nejdříve uložen na server pomocí funkce **uploadImage**, po úspěšném nahrání ho funkce **annotateImage** anotuje. Z nejlepšího výsledku se stává správná odpověď. Následně funkce **findRelatedWords** vyhledá podobná slova. Z náhodných dvou se stávají špatné odpovědi. Pokud funkce **annotateImage** nebo **findRelatedWords** nebyla schopna vrátit výsledek, jako odpoveď se použije "Unable to generate Answer". Pokud proběhne bez problému, uloží se otázka pomocí funkce **createQuestion** do databáze a funkce **updateSize** zvýší počet otázek ve vybrané sadě. Kód funkce je k vidění na ukázce C.9.

Kvalitu vygenerovaných otázkách jsem testoval na 30 mnou vyfocených objektech. Při focení jsem se snažil měnit co nejvíce pozadí aby se testování přiblížilo co nejvíce reálnému světu. Výsledek je vidět v tabulce **??**, **??**, **??** a **??**. Z tabulky je vidět, že anotace obrázku je ve většině případů správná. Naopak související slova jsou buďto naprosto mimo nebo vůbec neexistují.

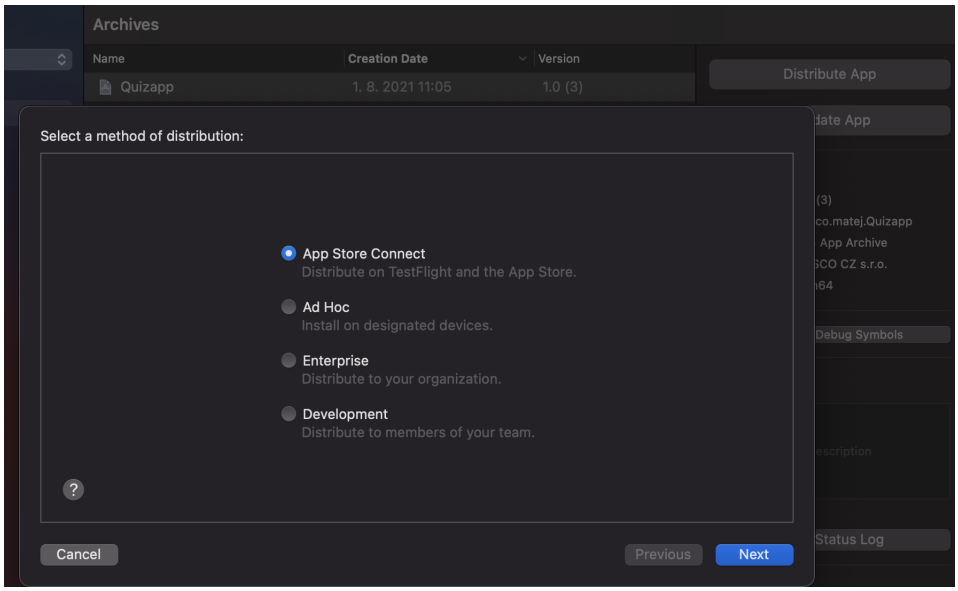
[4] https://firebase.google.com/docs/storage
[5] https://firebase.google.com/products/extensions/storage-resize-images

Obrázek	Generovaná správná odpověď	Generované špatné odpovědi
	Glasses	Specs, Eyeglasses
	Apple	Malus Pumila, Orchard Apple Tree
	Mug	Shlemiel, Patsy
	Tableware	-, -
	Dog	Firedog, Hound
	Tableware	-, -
	Flowerpot	-, Pot
	Waste container	-, -

Tabulka 5.1: Otázky s vygenerovanými odpovědmi

5.6 Nasazení na TestFlight

Jelikož je XCode napojen na službu App Store Connect, jejíž součástí je TestFlight, nasazení bylo poměrně jednoduché. V první řadě bylo potřeba vytvořit aplikaci v App Store Connect pomocí bundleID, které musí být stejné jako bundleID v XCode projektu. Dalším krokem bylo vygenerování archivu aplikace v XCodu a jeho následovné distribuování na App Store Connect (obrázek 5.2). Poslední krok se odehrával v App Store Connect, kde bylo potřeba přiřadit externí testovací skupinu a odeslat aplikaci na revizi.



Obrázek 5.2: XCode - archive a jeho distribuce

Kapitola 6

Závěr

Všechny cíle této práce byly do jisté míry splněny a tak považuji práci za úspěšnou. Přesto nejsem s výsledkem naprosto spokojen. Analýzy splňují svůj účel a frameworky SwiftUI a Combine byly otestovány do takové míry, že jsem schopen si o nich udělat vlastní názor. Co se týče aplikace samotné, ta splňuje veškeré požadavky, které jsem na začátku definoval. Nicméně jsou zde stále věci, které by bylo potřeba doladit či pozměnit.

6.1 Frameworky SwiftUI a Combine

Jelikož mobilní vývoj pro mě byl naprosto nová zkušenost, bylo zde ohromující množství informací, které bylo potřeba si nastudovat. Použitím těchto frameworků jsem si tento fakt příliš neulehčil. Vzhledem k jejich mládí bylo občas obtížné hledat řešení na problém, který zrovna řešíte.

Nicméně práce se SwiftUI byla relativně jednoduchá a intuitivní. Jeho největším problém je tak stále jeho mládí, jelikož plno důležitých komponent pořád chybí. To se dá řešit UIKitem, se kterým bez problému pracuje, přesto to není pro nově příchozí programátory ideální řešení. Apple však přichází každý rok s dalšími novinkami a je tak otázka času kdy SwiftUI naprosto nahradí UIKit.

Co se týče Combine, tak uvedením async / await do swiftu dal Apple jasně najevo, že nemá v plánu tento framework dále rozvíjet.

6.2 Budoucí plány

V budoucnu bych se chtěl k vývoji znovu vrátit a opravit či předělat určité části. Dále aplikaci otestovat na reálných uživatelích a přidat několik funkcionalit. Jednou z nich je přidání fulltextového vyhledávání, jelikož firestore tuto funkci nenabízí.

Literatura

[1] Apple. Xcode. https://developer.apple.com/documentation/xcode, 2003.

[2] Apple. ios human interface guidelines. https://developer.apple.com/design/human-interface-guidelines/ios, 2007.

[3] Apple. Testflight. https://developer.apple.com/testflight/, 2014.

[4] Apple. Discoverable design. https://developer.apple.com/videos/play/wwdc2021/10126, 2021.

[5] Apple. Meet async/await in swift. https://developer.apple.com/videos/play/wwdc2021/10132/, 2021.

[6] Apple. What's new in swiftui. https://developer.apple.com/videos/play/wwdc2021/10018/, 2021.

[7] Datamuse. Datamuseapi. https://www.datamuse.com/api/, 2016.

[8] I. D. Foundation. What are design guidelines? https://www.interaction-design.org/literature/topics/design-guidelines, -.

[9] Google. Firebase storage. https://firebase.google.com/docs/storage, -.

[10] Google. Firebase. https://firebase.google.com/, 2014.

[11] Google. Cloud vision. https://cloud.google.com/vision, 2015.

[12] Google. Tensorflow. https://www.tensorflow.org/lite/, 2015.

[13] Google. Firestore. https://firebase.google.com/docs/firestore/data-model, 2019.

[14] R. Hat. What is an ide? https://www.redhat.com/en/topics/middleware/what-is-ide, -.

[15] A. Inc. and open-source contributors. Closures. https://docs.swift.org/swift-book/LanguageGuide/Closures.html, -.

[16] A. Inc. and open-source contributors. Swift. https://swift.org/about/#swiftorg-and-open-source, 2014.

[17] A. Inc. and open-source contributors. Swift. https://developer.apple.com/swift/, 2014.

[18] A. Inc. and open-source contributors. Core ml. https://developer.apple.com/documentation/coreml, 2017.

[19] A. Inc. and open-source contributors. Combine. https://developer.apple.com/documentation/combine/, 2019.

[20] A. Inc. and open-source contributors. Swiftui. https://developer.apple.com/documentation/swiftui/, 2019.

[21] A. Inc. and open-source contributors. Swiftui - state and data flow. https://developer.apple.com/documentation/swiftui/state-and-data-flow, 2019.

[22] S. University. Mvvm and the swift type system. https://www.youtube.com/watch?v=4GjXq2Sr55Q, 2020.

[23] S. University. Reactive ui + protocols + layout. https://www.youtube.com/watch?v=SIYdYpPXil4&ab_channel=Stanford, 2020.

[24] R. Wenderlich. Combine: Getting started. https://www.raywenderlich.com/7864801-combine-getting-started, 2020.

[25] WordsAPI. Wordsapi. https://www.wordsapi.com/, 2015.

Příloha A

Návod k instalaci

K otestování aplikace je potřeba zařízení **iPhone** s nainstalovanou aplikací **TestFlight**. Pro prohlížení kódu doporučuji IDE **XCode**.

Důležité odkazy:

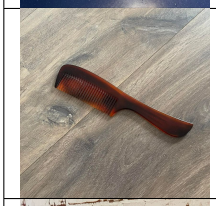
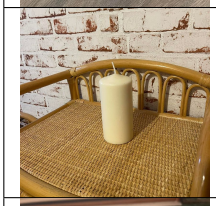
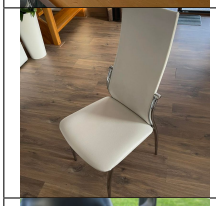
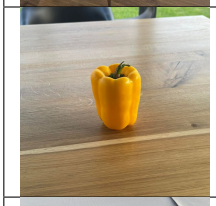
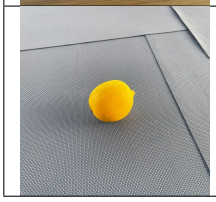
- Aplikace - `https://testflight.apple.com/join/golue9be`
- Zdrojový kód - `https://gitlab.fel.cvut.cz/bartamat/quizit`

Příloha B

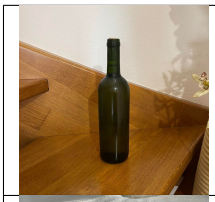
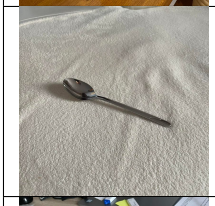
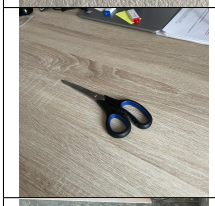
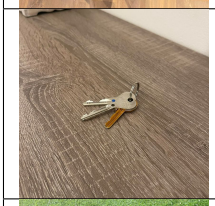
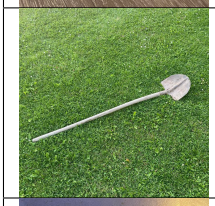
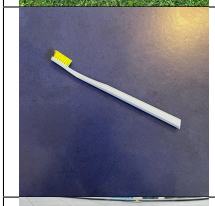
Tabulky

Obrázek	Generovaná správná odpověď	Generované špatné odpovědi
	Cat	Big Cat, Regorge
	Glove	Gloves, Baseball Glove
	Hammer	Hammering, Pounding
	Hat	Lid, Chapeau
	Ladder	Ravel, Run
	Bicycle wheel	-, -

Tabulka B.1: Otázky s vygenerovanými odpovědmi část - 2

Obrázek	Generovaná správná odpověď	Generované špatné odpovědi
	Tap	Strike, Hydrant
	Comb	Disentangle, Comb Out
	Cangle	Cd, Wax Light
	Vegetable	Vegetative, Vegetal
	Knife	Stab, Tongue
	Chair	Chairwoman, Moderate
	Bell pepper	-, -
	Lemon	Gamboge, Lemon Yellow

Tabulka B.2: Otázky s vygenerovanými odpovědmi část - 3

Obrázek	Generovaná správná odpověď	Generované špatné odpovědi
	Packaged goods	-, -
	Spoon	Spoonful, Smooch
	Scissors	Pair Of Scissors, -
	Packaged goods	-, -
	-	-, -
	-	-, -
	Toothbrush	Soup-Strainer, -
	Stool	Bm, Toilet

Tabulka B.3: Otázky s vygenerovanými odpovědmi část - 4

Příloha C

Ukázky kódu

C.1 Firebase

```
1  struct Question: Codable, Identifiable {
2      var id: String
3      var imageRef: String
4      var correctAnswer: String
5      var wrongAnswers: [String]
6      var lastUpdated: Date
7  }
8
9  struct QuestionSet: Codable, Identifiable {
10     var id: String
11     var title: String
12     var emoji: String
13     var size: Int
14     var creator: String
15     var creatorUsername: String
16     var lastUpdated: Date
17     var rating: Int
18     var shared: Bool
19 }
20
21 struct User: Codable {
22     var uid: String
23     var username: String
24     var email: String
25     var favorite: [String]?
26
27 }
```

Ukázka kódu C.1: Použitý model v aplikaci

```
1  func getPopularSets(amount: Int, completion: @escaping
       ([QuestionSet]) -> Void) {
2      db.order(by: "rating").limit(to: amount).getDocuments {
       (query, error)  in
3          guard let documents = query?.documents else {
4              print("Error getting popular sets - \(error!)")
5              return
6          }
7          let sets = documents.compactMap { document ->
       QuestionSet? in
8              try? document.data(as: QuestionSet.self)
9          }
10         completion(sets)
11     }
12 }
```

Ukázka kódu C.2: Funkce pro načtení populárních kvízů

```
1  func loadUserSetsListener(by order: String, descending: Bool) {
2          db.whereField("creator", isEqualTo: uid).order(by:
       order, descending: descending).addSnapshotListener {
       (querySnapshot, error) in
3              guard let documents = querySnapshot?.documents else
       {
4                  print("Error when loading user sets -
       \(error!))")
5                  return
6              }
7              self.sets = documents.compactMap { document ->
       QuestionSet? in
8                  try? document.data(as: QuestionSet.self)
9              }
10         }
11     }
```

Ukázka kódu C.3: Funkce pro načtení uživatelem vytvořených kvízů pomocí SnapshotListeneru

```
1  func signIn(email: String, password: String) {
2      self.response = .loading
3
4      Auth.auth().signIn(withEmail: email, password: password) {
       (result, error) in
5          if error != nil {
6              print("Error signing in - \(error!)")
7
8              if let errorCode = AuthErrorCode(rawValue:
       error!._code) {
9                  self.errorHandler(errorCode)
10             }
11         } else {
12             self.response = .normal
13         }
14     }
15 }
```

Ukázka kódu C.4: Funkce pro přihlášení

C.2 Combine

```
1     @Published var questionSetViewModels: [SetRowViewModel] = []
2     private var setRepository = SetRepository()
3     private var cancellables: Set<AnyCancellable> = []
4
5     init() {
6         setRepository.$sets.map { sets in
7             sets.map(SetRowViewModel.init)
8         }
9         .assign(to: \.questionSetViewModels, on: self)
10        .store(in: &cancellables)
11    }
```

Ukázka kódu C.5: Mapování kvizové sady na pole list ViewModelů

C.3 SwiftUI

```
struct ExploreView: View {

    @State private var showCreatedList = false
    @State private var showPopularList = false

    @EnvironmentObject var auth: Authentication
    @StateObject private var viewModel = ExploreViewModel()

    var body: some View {
        NavigationView {
            VStack {
                ScrollView {
                    ExploreSection(header: "You Created",
    listRowVMs: viewModel.ownedSets)
                        .padding(15)
                        .onTapGesture {
                            showCreatedList.toggle()
                        }
                        .fullScreenCover(isPresented:
    $showCreatedList) {
                            CreatedListView(showView: $showCreatedList)
                        }
                    ExploreSection(header: "Most Popular",
    listRowVMs: viewModel.popularSets)
                        .padding(15)
                        .onTapGesture { showPopularList.toggle() }
                        .fullScreenCover(isPresented:
    $showPopularList) {
                            PopularListView(showView: $showPopularList)
                        }
                }
            }
            .onAppear { viewModel.getPeep() }
            .navigationTitle("Explore")
                .navigationBarItems(
                    trailing:
                    Menu {
                        Button(action: { auth.signOut() }) {
                            Text("Sign out")
                            Spacer()
                        }
                    }
                    label: { Image(systemName: "person.circle") }
                    .font(Font.system(size: 28, weight: .light,
    design: .default)))
        }
    }
}
```

Ukázka kódu C.6: Explore sekce

```
1  struct ImagePicker: UIViewControllerRepresentable {
2
3      @Binding var showPicker: Bool
4      @Binding var imagesDataSet: Set<ImageData>
5      var useCamera: Bool
6      var multiple: Bool
7
8
9      func makeCoordinator() -> Coordinator {
10         Coordinator(parent: self)
11     }
12
13     func makeUIViewController(context: Context) ->
       UIImagePickerController {
14         let controller = UIImagePickerController()
15         controller.allowsEditing = true
16         if useCamera {
17             controller.sourceType = .camera
18         } else {
19             controller.sourceType = .photoLibrary
20         }
21         controller.delegate = context.coordinator
22
23         return controller
24     }
25
26     func updateUIViewController(_ uiViewController:
       UIImagePickerController, context: Context) {
27
28     }
29 }
```

Ukázka kódu C.7: komponent ImagePicker

```
1  class Coordinator: NSObject, UIImagePickerControllerDelegate,
       UINavigationControllerDelegate {
2
3      //Get parent view context to update image data
4      var parent: ImagePicker
5
6      init(parent: ImagePicker) {
7          self.parent = parent
8      }
9
10     func imagePickerController(_ picker:
       UIImagePickerController, didFinishPickingMediaWithInfo
       info: [UIImagePickerController.InfoKey : Any]) {
11         //Getting image and closing
12         if !parent.multiple {
13             parent.imagesDataSet.removeAll()
14         }
15         if let imageData =
       (info[UIImagePickerController.InfoKey.editedImage] as?
       UIImage)?.jpegData(compressionQuality: 0.45) {
16             parent.showPicker.toggle()
17             let id = UUID().uuidString
18             parent.imagesDataSet.insert(ImageData(id: id, data:
       imageData))
19         }
20         else if let imageData =
       (info[UIImagePickerController.InfoKey.originalImage] as?
       UIImage)?.jpegData(compressionQuality: 0.45) {
21             parent.showPicker.toggle()
22             let id = UUID().uuidString
23             parent.imagesDataSet.insert(ImageData(id: id, data:
       imageData))
24         } else {
25             parent.showPicker.toggle()
26         }
27
28     }
29
30
31
32     func imagePickerControllerDidCancel(_ picker:
       UIImagePickerController) {
33             //Closing the view if cancelled
34         parent.showPicker.toggle()
35     }
36
37 }
```

Ukázka kódu C.8: Součást komponentu ImagePicker - třída Coordinator

C.4 Generování otázky

```
1  func add(completion: @escaping (Bool?) -> Void)  {
2      imageRepository.uploadImage(setID, image: imageData.first!)
       { imageRef in
3          if imageRef != nil {
4
       self.imageRecognition.model.annotateImage(imageRef!.fullPath)
       { imageAnnotation in
5
       self.wordRelatibility.model.findRelatedWords(of:
       imageAnnotation!) { relatedWords in
6                      self.questionRepository.createQuestion(
7                          setID: self.setID,
8                          imageRef: imageRef!,
9                          correctAnswer: imageAnnotation!,
10                         wrongAnswers:
       Array(relatedWords!.prefix(2)))
11                     self.setRepository.updateSize(decrement:
       false, id: self.setID)
12                 }
13             }
14             completion(true)
15         } else {
16             completion(false)
17         }
18     }
19 }
```

Ukázka kódu C.9: Funkce pro vygenerování jedné otázky