Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Vývoj mobilní aplikace pro iOS

Bc. David Petřina

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. David Petřina

Systémové inženýrství a informatika
Informatika

Název práce

Vývoj mobilní aplikace pro iOS

Název anglicky

Mobile application development for iOS

Cíle práce

Cílem práce je návrh, analýza a implementace aplikace pro mobilní zařízení s operačním systémem iOS za použití frameworků SwiftUI, Core ML a Create ML. Dílčími cíli je návrh architektury aplikace, vytvoření UI specifikace a vytvoření datového modelu, který bude potřeba pro rozpoznávání objektů v rámci Machine Learning.

Metodika

Diplomová práce se zabývá problematikou vývoje aplikace pro mobilní zařízení s operačním systémem iOS v nativním prostředí. V analytické části práce jsou studovány odborné informační zdroje potřebné k vývoji aplikace. Zejména se jedná o nastudování dokumentace k frameworkům SwiftUI, Create ML a Core ML. V praktické části je provedena analýza požadavků, návrh architektury aplikace, specifikace aplikace pomocí wireframů a scénářů, která je provedena v softwaru Justinmind. Dále je vytvořen datový model pomocí frameworku Create ML, který je následně použitelný pro Core ML. Vývoj mobilní aplikace probíhá v prostředí nástroje Xcode s programovacím jazykem Swift.

Doporučený rozsah práce

60–80 stran

Klíčová slova

iOS; SwiftUI; Core ML; Create ML; Swift; mobilní aplikace; Xcode

---

Doporučené zdroje informací

Apple Developer [online]. Apple, 2019 [cit. 2019-07-10]. Dostupné z: https://developer.apple.com/

LACKO, Ľuboslav. Vývoj aplikací pro iOS. Přeložil Martin HERODEK. Brno: Computer Press, 2018. ISBN 978-80-251-4942-3.

Raywenderlich.com [online]. Razeware, 2019 [cit. 2019-07-10]. Dostupné z: https://www.raywenderlich.com/

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Michal Stočes, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 11. 10. 2019

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 14. 10. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 24. 02. 2020

Čestné prohlášení

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

V Praze dne 06. 04. 2020 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Michalu Stočesovi, Ph.D. za odborné vedení diplomové práce a poskytnuté rady při jejím vypracovávání.

Vývoj mobilní aplikace pro iOS

Abstrakt

Tato práce se zabývá vývojem mobilní aplikace pro operační systém iOS. Hlavním cílem bylo vytvořit mobilní aplikace, která bude schopna pomocí strojového učení rozpoznávat psí plemena na základě uživatelem zvolené fotografie. V teoretické části byla provedena komparace nejčastěji využívaných aplikačních architektur pro tvorbu aplikací pro iOS. Byly specifikovány a charakterizovány knihovny potřebné k implementaci aplikace a spolu s tím více charakterizováno strojové učení se zaměřením na mobilní zařízení. Na závěr teoretické části byla charakterizována UI specifikace a její důležitost tvorby.

V počátečních krocích praktické části byla vytvořena analýza požadavků definující požadované chování a vlastnosti mobilní aplikace. V prostředí aplikace Create ML byl vytvořen model pro strojové učení pro rozpoznávání psích plemen. Jeho validační přesnost 66 % byla zvolena jako dostačující pro implementaci do mobilní aplikace. Před vývojem aplikace byla vytvořena UI specifikace potřebná pro následný vývoj. Pro implementaci bylo zvoleno nativní prostředí Xcode a hlavní framework SwiftUI.

Aplikace umožňuje uživateli vybrat fotografii z knihovny nebo pořídit vlastní fotografii a následně ji vyhodnotit. Dále je zobrazen seznam psích plemen, které byly v rámci trénování modelu využity a ke každému z nich je možné zobrazit základní informace.

Klíčová slova: iOS, Xcode, Swift, SwiftUI, mobilní aplikace, strojové učení, Apple, UI specifikace

Mobile application development for iOS

Abstract

This thesis deals with mobile application development for iOS operating system. The main goal was to create a mobile application that will be able to classify a dog breed based on a user selected photo using a machine learning algorithm. The theoretical part compares the most commonly used application architectures for creating iOS applications. Frameworks needed for an implementation were specified and characterized and machine learning focused on mobile applications was further characterized. At the end of the theoretical part UI specification was characterized with its importance of creation.

In the introductory steps of the practical part, a software requirements specification was created defining the desired behavior and properties of the mobile application. A machine learning model for recognizing dog breed was created in an application Create ML. Its validation accuracy of 66 % was chosen to be sufficient for implementation in the mobile application. Prior to application development, the UI specification was created for subsequent development. For the implementation was chosen the native development environment Xcode and programming language Swift with its framework SwiftUI.

The application allows the user to take a photo or select from a photo library which is then evaluated. Furthermore, a list of dog breeds that have been used in the training model is shown and for each of them it is possible to display basic information.

Keywords: iOS, Xcode, Swift, SwiftUI, mobile application, machine learning, Apple, UI specification

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

MVC – Model View Controller

MVP – Model View Presenter

MVVM – Model View ViewModel

VIPER – View Interactor Presenter Entity Router

ML – Machine Learning

UI – User Interface

1 Úvod

V současné moderní době, kdy jsou mobilní telefony nedílnou součástí životů lidí, se dynamickým způsobem vyvíjí jejich užívání. Jak jde technologie kupředu, posouvají se také i mobilní telefony. Neslouží již jen jako prostředek k vzájemnému dorozumění, ale mění se v osobního kapesního společníka, který do jisté míry zastupuje funkci stolních či přenosných počítačů.

Aplikací pro mobilní zařízení jsou již miliony a zasahují do všech možných kategorií. Přijít tedy s něčím novým je pro vývojáře aplikací, respektive jejich zadavatele, velmi složité. Většina možných problému a zadání má již své řešení v podobě mobilní aplikace. V současné době není pro úspěšné publikování aplikace stěžejní její nápad, konkrétně její funkcionalita a podstata, ale samotný způsob, jakým je obsah a chování aplikace prezentováno. Už pár let je k dispozici například rozšířená realita, která umožňuje uživateli hlubší požitek z aplikace tím, že má v reálném čase pomocí kamery zařízení zprostředkovány různé objekty a interakce.

Současné aplikace jsou dokonce i adaptované na zobrazování obsahu na základě uživatele a jeho historických úkonů s danou aplikací. Příkladem toho jsou aplikace pro přehrávání hudby nebo filmů a seriálů. Na základě chování uživatele, jeho předešlých výběrů písniček či pořadů navrhují, co by se mu mohlo líbit a doporučují tak nový obsah. Stejně tak existují aplikace využívané pro sociální sítě, které při fotografování analyzují obličej uživatele a přidávají filtry navíc, které pasují k jeho vizáži.

Tyto všechny funkcionality jsou dosaženy pomocí Machine Learning, tedy v češtině strojového učení. Na základě připraveného modelu, kterému bylo dodáno velké množství různorodých dat, je možné vyhodnocovat nebo i doporučovat výsledné obsahy. Stroj se tak učí stále novým věcem, zejména se učí rozhodovat stejným způsobem jako člověk.

Strojové učení bylo pro svoji významnost v dnešním světě umělé inteligence a pro svoji moc zvoleno jako téma této diplomové práce. Hlavním cílem bylo navrhnout a implementovat řešení strojového učení pro zařízení s operačním systémem iOS. Konkrétním zaměřením strojového učení je klasifikování psích plemen na uživatelem zvolené fotografii. Aplikace je rovněž vhodná i jako databáze psích plemen s uvedením základních informací o plemeni.

V obchodě aplikací App Store pro zařízení s operačním systémem iOS nebyla nalezena aplikace pro strojové učení klasifikující psí plemena v české lokalizaci, což vytváří příležitost, jak tuto mezeru vyplnit.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem práce je návrh, analýza a implementace aplikace pro mobilní zařízení s operačním systémem iOS pro účely strojového učení za použití frameworků SwiftUI, Core ML, Create ML a Vision. Dílčími cíli práce je návrh architektury aplikace, vytvoření UI specifikace a vytvoření modelu pro strojové učení, který bude schopen klasifikovat psí plemena na základě vybraných fotografií.

2.2 Metodika

Metodika diplomové práce je založena na studiu a rozboru odborných informačních zdrojů a volně dostupných dokumentací potřebných k vývoji aplikací v nativním prostředí. Pro implementaci strojového učení v integrovaném vývojářském prostředí Xcode je nutné studium dokumentace k frameworkům SwiftUI, Create ML, Core ML a Vision, díky kterým je možné vytvoření mobilní aplikace pomocí deklarativního způsobu programování a které jsou nezbytné pro její tvorbu s vlastnostmi strojového učení.

Je provedena charakteristika nejčastěji využívaných aplikačních architektur pro iOS aplikace, mezi které se řadí MVC, MVP, MVVM a VIPER a následně vybraná nejvhodnější, která bude respektovat a zachycovat její problematiku a rozsáhlost.

Pro fázi analýza je vytvořena analýza požadavků zachycující očekávané či požadované chování aplikace a její obecné vlastnosti. Pro fázi návrh je vytvořena UI specifikace obsahující wireframes (logické návrhy obrazovek), use cases (případy užití) a scénáře. Wireframes jsou vytvořeny v online nástroji Balsamiq, který nabízí své služby bez nutnosti vlastnění placené licence.

K vývoji aplikace pro strojové učení na rozpoznávání fotografií je vytvořen model strojového učení pomocí nástroje Create ML od Apple. Podkladová data pro vytvoření modelu jsou získána veřejně dostupné databáze na webových stránkách Stanfordfordské univerzity.

3 Teoretická východiska

3.1 Mobilní operační systém iOS

iOS je operační systém určený pro mobilní telefony od společnosti Apple. Je založen na principech operačního systému Mac OS X, který je využíván na stolních a přenosných zařízení a ze kterého si převzal většinu funkcionalit, které obohatil o nové možnosti jako je například podpora dotykového ovládání.

Hned za operačním systémem Android zaujímá iOS druhé místo v žebříčku nejpoužívanějších operačních systémech pro mobilní telefony. Oproti ostatním operačním systémům se iOS liší v mnoha věcech, jednou z nedůležitějších je bezesporu bezpečnost. Aplikace jsou v telefonu drženy odděleně a nejsou schopny spolu samovolně komunikovat a interagovat, což ve výsledku může zabránit rozšíření nežádoucích jevů jako jsou například viry. Pro schválení aplikací před jejich publikací na App Store jsou velmi důkladně testovány a prochází velmi striktními pravidly a z toho důvodu je velmi nízká pravděpodobnosti nainstalování aplikace s nežádoucím obsahem. [1]

3.1.1 Human Interface Guidelines

Pro vyvinutí aplikace, respektive její schválení před publikací na App Store, je nutné dodržovat pravidla a standardy, které odlišují aplikace pro iOS od ostatních platforem. Tyto standardy jsou definovány třemi hlavními tématy: [2]

Clarity (jasnost, srozumitelnost) – napříč systémem by měl být text čitelný v každé velikosti, ikony přehledné a lehce srozumitelné, ozdoby jemné a vhodné. Negativní prvky místa a prostoru, barvy, písma, grafiky a prvků rozhraní by měly mírně zvýrazňovat důležitý obsah a zprostředkovávat interaktivitu. [2]

Deference (úcta, respekt) – plynulý pohyb a krásné rozhraní by mělo pomáhat lidem porozumět obsahu a interagovat s ním, aniž by jim to mělo dělat jakýkoli problém. Obsah by měl vyplňovat celou obrazovku, zatímco průsvitnost a rozmazání by mělo naznačovat na soustředěný prvek. Minimální využívání ohraničení, barevných přechodů a vržených stínů by mělo udržovat rozhraní vzdušné, přičemž je zajištěno, že obsah je prvořadý. [2]

Depth (hloubka) – odlišné vizuální vrstvy a realistický pohyb zprostředkovávají hierarchii, ovlivňují životnost a ulehčují pochopitelnost. Doteky a objevitelnost zesilují

požitek a umožňují přístup k funkcionalitám a dalšímu obsahu bez ztráty kontextu. Přechody poskytují smysl hloubky při procházení obsahem. [2]

Pokud vývojáři chtějí maximalizovat dopad a dosah vyvinutých aplikací, je nutné, aby dodržovali jejich následující zásady: [2]

Estetická integrita – reprezentuje, jak dobře je integrováno zobrazení a chování aplikace k jejím funkcím. Pro uvedení příkladu aplikace, která pomáhá lidem provést důležitý úkol, může zobrazovat pouze nenápadné grafické prvky, standardní ovládací prvky a předvídatelné chování. Naopak tomu například u her je vhodné uživatelské rozhraní aplikace podmanivým vzhledem, který v uživateli vyvolá zábavu a vzrušení při jejím používání. [2]

Konzistence – konzistentní aplikace implementuje standardy a paradigma pomocí systémových prvků uživatelského rozhraní, ikon a textových stylů. Aplikace by měla obsahovat funkce a chování v takové podobě, v jaké je uživatel bude očekávat.

Přímá manipulace – pomáhá koncovému uživateli pochopení chodu aplikace. Pokud je obrazovka otočena nebo je použita interakce ve formě gesta pomocí prstů, na základě těchto akcí je ihned zobrazen odpovídající výsledek v rámci obsahu. [2]

Zpětná vazba – je důležité informovat uživatele o výsledcích, kterých dosáhli na základě jejich interakcí. V iOS aplikacích je využíváno například rotujícího kolečka pro zobrazení časové prodlevy, než se vykoná dlouhotrvající proces, lehkého zvýraznění interagovaných prvků po jejich kliknutí či využití animací a zvuků, které pomáhají objasnit výsledky akcí. [2]

Metafory – uživatelé se naučí ovládání aplikace mnohem rychleji, pokud její objekty a akce budou připomínat jim známé zkušenosti z reálného světa. V iOS je práce s metaforami snadná z důvodu pohybu uživatelů v aplikacích pomocí gest. Pohybují s obrazovkami pro zobrazení obsahu, který se pod nimi skrývá nebo přetahují a posouvají obsah. [2]

Uživatelské ovládání – napříč systémem iOS jsou v hlavní řídící roli uživatelé a ne aplikace, ty mohou navrhnout postup nebo varovat před nebezpečnými důsledky, ale obvykle je chybou, kdy aplikace převezme kontrolu nad uživatelem. Nejlepší aplikace najdou správnou rovnováhu mezi povolením interakcí uživatelů a vyhýbáním se nežádoucím výsledkům. Pro zajištění pocitu kontroly uživatele nad aplikací je možné využít interakčních

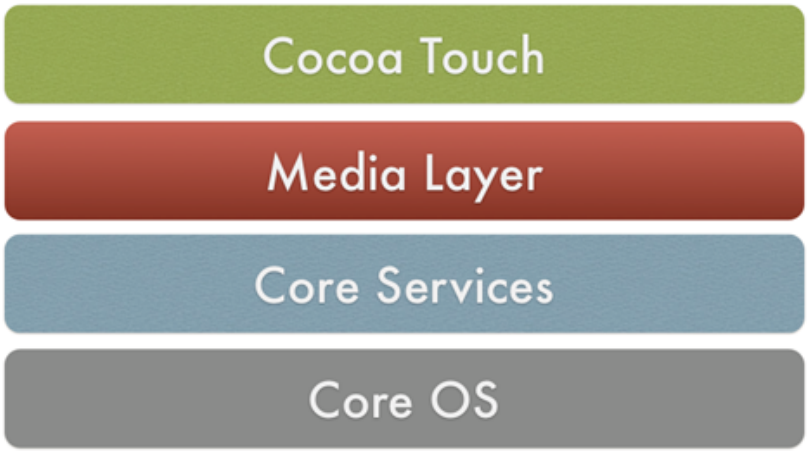
prvků, které jsou známé a předvídatelné, informují o nežádoucích akcí a usnadňují zrušení operací, i když již probíhají. [2]

3.1.2 Architektura iOS

Operační systém iOS hodně vychází z operačního systému Mac OS X, který je součástí stolních a přenosných počítačů od společnosti Apple Inc. Z tohoto operačního systému si iOS vzal jen funkcionality, které je schopen využít v mobilní verzi a zároveň byla přidána podpora pro dotykové ovládání. Struktura iOS se člení na čtyři abstraktní vrstvy, které zajišťují funkčnost a poskytují vývojářům API a frameworky potřebné k vývoji aplikací.

Na nejvyšší úrovni funguje iOS jako prostředník mezi základním hardwarem zařízení a aplikacemi, které jsou vývojáři vytvořeny. Aplikace nekomunikují napřímo s hardwarem, ale učiní tak prostřednictvím kolekce dobře definovaných systémových rozhraní. Tato rozhraní usnadňují tvorbu aplikací, které nepřetržitě pracují na zařízeních s různými hardwarovými schopnostmi. Nižší vrstvy poskytují základní služby, na kterých všechny aplikací běží a na které spoléhají a vyšší vrstvy nabízejí služby úzce související s grafikou a rozhraním. [3]

V této kapitole jsou uvedeny všechny čtyři vrstvy a ke každé jsou zmíněné příklady, které jsou v rámci každé obsaženy.



Obrázek 1 - Vrstvy operačního systému iOS[4]

3.1.2.1 Vrstva Cocoa Touch

Tato vrstva je na vrcholu architektury operačního systému a v první řadě je zodpovědná pro vzhled aplikací a jejich schopnosti reakce na akce od uživatelů. Je založena

na vrstvě Cocoa, která je dostupná pro operační systém Mac OS X a zároveň je rozšířena a upravena pro potřeby mobilních telefonů iPhone. Napsána je v programovacím jazyce Objective-C. [3]

Mezi hlavní rysy a technologie stojí za zmínění Multitasking, který je již od verze iOS 4 dostupný pro zařízení podporující tento operační systém a v praxi umožňuje po ukončení aplikace její přenesení na pozadí, kde buďto pozastaví svojí činnosti nebo dostane určitý čas pro dokončení právě probíhajícího úkonu (například hudební přehrávač, který na pozadí vykonává svoji úlohu). Tato technologie je navržena k maximalizaci výdrže baterie. [5] Rozpoznávání gest umožňuje chování aplikací po provedení předem nadefinovaného gesta, kde mezi základní patří dotek, sevření a rozevření prstů, přetahování objektů, rotace, dlouhý stisk či tvrdší stisk. V neposlední řadě stojí za uvedení technologie pro Push a Local notifikace. Push notifikace slouží k upozornění uživatele na nové informace i v případě, kdy aplikace zrovna není spuštěna. Local notifikace mohou být uživateli posílány prostřednictvím aplikací, které běží na pozadí a mohou ho tak upozornit na důležité informace. Například jsou toto notifikace posílané od navigační aplikace, která běží na pozadí a uživatele informuje na blížící se odbočku, i když zrovna může mít spuštěnou jakoukoli jinou aplikaci. [3]

Mezi frameworky, které jsou zastoupeny ve vrstvě Cocoa Touch se řadí:

- **UIKit** – hlavní framework, který je potřebný při vývoji mobilní aplikace pro zařízení s iOS. Vychází z velké části z frameworku AppKit, který je součástí Mac OS X, jen je přizpůsobeno pro práci s mobilními telefony či tablety. S jeho pomocí jsou nadefinované veškeré grafické prvky, se kterými uživatel může interagovat nebo například zajištěna správa baterie, fotoaparátu či interakce v rámci galerie fotografií. [6]
- **SwiftUI** – nově představený framework, o kterém je věnována kapitola níže.
- **MapKit** – poskytuje naprogramované uživatelské rozhraní, díky kterému je možné integrovat zobrazení map ve vlastních aplikací a nastavení vlastních parametrů při zobrazování záznamů v mapě. [7]
- **GameKit** – nabízí schopnost vytvoření mobilní aplikace, která bude propojovat uživatele (respektive hráče) a poskytovat jim vzájemné interakce. V rámci tohoto frameworku byla dříve implementována služba *Game Center*, která nabízela uživatelům přehled o dosažených a získaných úspěších

(Achievements) a zobrazení žebříčků (Leaderboards) pro jednotlivé herní aplikace. Od verze iOS 10 byla tato služba odstraněna a při vývoji aplikací je potřeba tyto jednotlivé prvky implementovat do her napřímo. [8][9]

3.1.2.2 Media vrstva

V této vrstvě iOS architektury jsou obsaženy grafické, zvukové a video technologie, které jsou potřebné k implementaci multimediálních funkcionalit v aplikacích.

Grafické technologie

Nejvíce využívaným způsobem, jak v aplikaci dosáhnout vysoké kvality grafického zobrazení je použití standardních pohledů a nastavení UIKit frameworku. Touto cestou je přenecháno systému renderování uživatelského rozhraní jednotlivých aplikací daných grafických prvků. Vývojáři také mohou využít vlastní cestu vytváření grafického rozhraní aplikací pomocí dodání vlastních obrázků, které mohou renderovat pomocí frameworků obsažených v UIKit. Tyto frameworky poskytují vykreslování obrazu, který je následně v aplikaci zobrazován v podobě kvalitního obrazu napříč různými rozlišeními. Níže jsou uvedeny jednotlivé frameworky, které se v rámci Media Layer architektury iOS mohou použít. [10][11]

- **Core Graphics** – jinak označováno také Quarty 2D a poskytuje možnost renderování vektorů a obrázků.
- **Core Animation** – je součástí frameworku Core Graphics a umožňuje vytvořit plynulé animace s pokročilejšími efekty.
- **SpriteKit** – nabízí nástroje a metody, díky kterým je možné tvořit a renderovat animované obrázky. Tento framework se využívá například u herních aplikací.
- **OpenGL** – technologie pro tvorbu a animace 2D a 3D grafiky v reálném čase. Tento framework je ve většině případů využit u tvorby her.
- **Cocoa text system** – tyto služby umožňují vytváření, upravování, zobrazování a uchovávání textu.
- **Core Text** – slouží k vykreslování textu v aplikacích.
- **Image Capture Core** – pomocí tohoto frameworku je nastaven přístup a oprávnění aplikace k fotoaparátu.
- **Image I/O** – pomáhá při čtení a zápisu grafických formátů dat.

Zvukové technologie

Tyto technologie umožňují přehrávání a nahrávání zvukových záznamů a používání vibrací. Součástí těchto technologií jsou následující frameworky: [11]

- **AV Foundation** – podporuje zvukové přehrávání, editování, analyzování a nahrávání.
- **OpenAL** – framework implementující 3D audio napříč platformami.
- **Core Audio** – obsahuje baliček frameworků, které podporují nahrávání, přehrávání, mixování, editování, převod formátů či synchronizaci.

Video technologie

V rámci těchto technologií je možné přehrávání video souborů buďto z nainstalovaných aplikací nebo z webového prostředí a nahrávání video záznamu. Následující frameworky se řadí do této skupiny: [11]

- **AVKit** – podporuje přehrávání vizuálního obsahu v aplikaci spolu s možnostmi při přehrávání videa.
- **AV Foundation –** podporuje přehrávání, nahrávání, čtení, zakódování, zapisování a upravování audiovizuálního obsahu.
- **Core Media –** poskytuje správu audiovizuálních medií.

3.1.2.3 Vrstva Core Services

Vrstva Core Services by se do češtiny mohla volně přeložit jako „hlavní služby“ a samotný název vypovídá o její funkci – poskytuje aplikacím systémové služby, ale nemá přímý vliv na uživatelské rozhraní. [12]

- **iCloud uložiště –** funkcionalita, která umožňuje aplikaci zapsat uživatelské dokumenty a data na cloudové služby uživatele, který následně má tyto dokumenty a data dostupné z jakéhokoli zařízení s operačním systémem macOS nebo iOS. Rozlišují se tři typy iCloud uložiště:
  - **Document storage –** pro uživatele viditelný obsah formou souborů jako jsou například dokumenty či prezentace
  - **Key-value storage –** slouží ke sdílení menšího množství dat jako jsou třeba preference nastavené v aplikaci.

    - **Core Data storage –** podporuje databázové řešení strukturovaných obsahů běžící na serveru a na více zařízení.
- **Social Media Integration –** dva frameworky z této skupiny (Accounts a Social) zjednodušují uživatelům sdílení obsahu na sociálních sítích z aplikací. Při používání Accounts frameworku není potřeba, aby aplikace ukládala přihlašovací údaje, protože uživatelé mohou povolit aplikaci přístup k jejich přihlašovacím údajům na sociální sít. Pokud v databázi Accounts není k dispozici žádný účet pro přihlášení, je následně možné jej vytvořit a uložit přímo z aplikace. Social framework nabízí jednoduché rozhraní pro sdílení statusů nebo fotografií na uživatelský profil na sociální síti.
- **Internationalization and Localization –** volně přeloženo zmezinárodnění a lokalizování je v aplikacích využíváno z důvodu nastavení jazyka v aplikaci na konkrétní oblast, ve které se uživatel nachází (respektive nastavení jazyka na jazyk nastavený na mobilním zařízení). Před samotnou lokalizací je nutno provést zmezinárodnění, což následně umožní správně načíst a zobrazit lokalizovaný obsah.
- **SQLite Database –** databázové řešení, které je možné vložit do aplikací. Je možné vytvoření lokálních databázových souborů a spravovat tabulky a záznamy v těchto souborech. Knihovna je navržena pro obecné potřeby, ale stále je optimalizována k rychlému přístupu k databázovým záznamům.
- **In-App Purchase –** tato služba v rámci frameworku StoreKit umožňuje uživatelům uskutečňovat nákupy v aplikacích nebo na App Store. Existují dva hlavní druhy nákupů v aplikacích: prvním je jednorázový nákup, kdy si uživatel může například v nějaké hře koupit výhodný balíček, který ho posune ve hře dále, a druhým typem je takzvaný subscription, tedy předplatné, kdy uživatel si koupí měsíční nebo roční předplatné pro přístup do aplikace či do placených obsahů.

3.1.2.4 Vrstva Core OS

Nejnižší vrstva operačního systému iOS obsahuje nízko úrovňové funkce a podporuje všechny na ní postavené vrstvy. Služby z této vrstvy nejsou pro uživatele v rámci aplikací natolik viditelné, ale jsou naopak využívány vysoko úrovňovými vrstvami, které jsou

potřeba například v případě komunikace s externími hardwarovými zařízeními, poskytování nízko úrovňového síťového sdílení informací (networking) či fundamentální služby operačního systému jako třeba správa operační paměti nebo zpracovávání souborového systému. [13][14]

- **Accelerate Framework** – obsahuje optimalizované API napsané v jazyce C pro výpočty komplexních a vysokých čísel a zpracování vektorových, obrázkových a DSP (zpracování diskrétního signálu) výpočtů.
- **Security Framework –** tento framework poskytuje veškeré potřebné bezpečnostní opatření, které jsou v chytrém telefonu zapotřebí při komunikaci s externími sítěmi. Zahrnuje do své správy certifikáty, veřejné a privátní klíče, zásad důvěryhodnosti, šifrování či ověřování datové integrity a autentizaci zprávy.
- **Core Foundation –** nabízí základní funkcionality jako jsou datové typy, manipulace s textovými řetězci, manipulaci s URL, základní XML manipulaci nebo data a časy.
- **System –** iOS je postavený na UNIXovém základu a z toho důvodu systémové komponenty z Core OS vrstvy nabízí podobné funkcionality jako u ostatních operačních systémů běžících na UNIXu. V této sekci je jádro operačního systému, na kterém je postavena celá iOS platforma a které nabízí nízko úrovňové rozhraní k hardwaru. Jádro je zodpovědné za virtuální paměť, souborový systém, nízko úrovňový networking či za správu životního cyklu procesů.

3.1.3 Architektura aplikace pro iOS

Samotnému vývoji aplikaci předchází neméně důležitý krok, který je spjat s vhodnou volbou aplikační architektury. Při tvorbě rozsáhlejších projektů, kde je v aplikaci zakomponováno desítky obrazovek je nutné provést analýzu, který z dostupných návrhů architektury bude pro aplikaci ten nejvhodnější. [15]

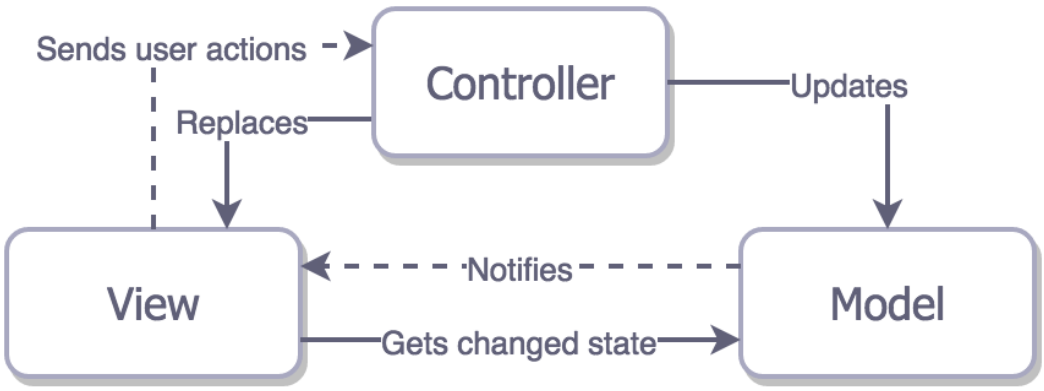
Nebude-li aplikace dodržovat základní principy některé ze zvolených návrhů, stane se zdrojový kód aplikace velmi nečitelný a špatně testovatelný. Při správném zvolení architektury mobilní aplikace je možné od sebe separovat data, které říkají, co se má v aplikaci ukazovat a pohledy, které říkají, jak se mají data zobrazovat. Každý objekt by měl

tedy plnit jen a pouze svoji roli. Stejně tak čitelnost zdrojového kódu by v ideálním případě měla být intuitivní, když by se přidal nový člověk do projektu, byl by schopen se v kódu co nejsnáze zorientovat. [15]

V této kapitole jsou detailněji popsány nejčastější typy aplikačních architektur využívající se ve vývoji mobilních aplikací pro iOS.

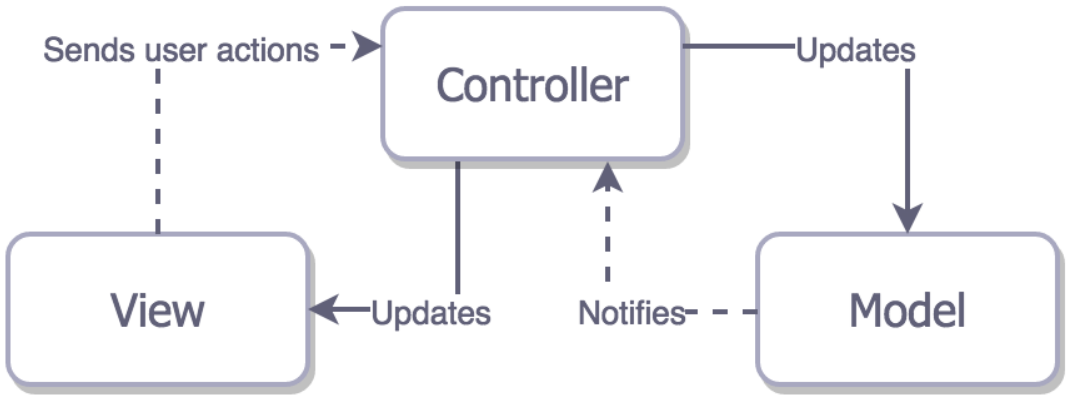
3.1.3.1 MVC

Nejčastěji používaným a nejpopulárnějším návrhem architektury je MVC – Model-View-Controller. Je možné rozlišovat dva typy aplikační architektury MVC – tradiční a Apple. U tradičního je View překládán Controllerem, jakmile je Model změněn. Při vývoji iOS aplikací se tento typ nevyužívá, jelikož všechny tři entity jsou úzce spjaty a každá z nich ví informace o zbývajících dvou. [17]



Obrázek 2 - Tradiční MVC[16]

V případě Apple MVC je Controller prostředníkem mezi View a Model a díky tomu nemají mezi sebou znalost všech dat. Tento typ návrhu je doporučován samotným Applem a je používán ve většině příkladů na oficiálních stránkách. Je to standardní aplikační architektura využívána v projektech pro iOS, macOS a watchOS aplikace. [17]

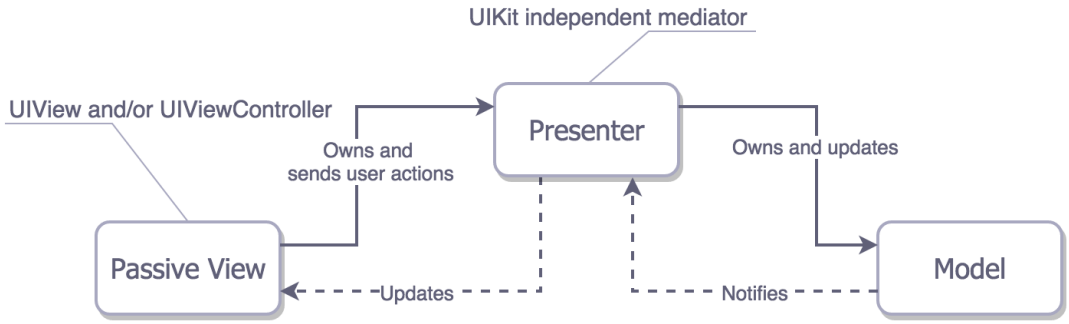


Obrázek 3 - Apple MVC[18]

Model obsahuje údaje a data (případně aplikační kód), který se stará o jejich generování, připravování či zpracovávání. View se rozumí prezentační vrstva, tedy grafické uživatelské rozhraní aplikace, které zobrazuje data a přijímá interakce uživatele jako jsou například dotyk na obrazovce či delší podržení objektu v aplikaci. Controller je prostředníkem mezi Model a View, tedy mezi prezentační a modelovou vrstvou. Na základě změn na prezentační vrstvě provádí změny na modelové vrstvě a zároveň aktualizuje modelovou vrstvu na základě informací z prezentační vrstvy. [19]

3.1.3.2 MVP

Model View Presenter je z velké části odvozen z Apple MVC, o kterém je psáno o kapitolu výše, avšak s jedním markantním rozdílem. U MVP Presenter zastoupil roli prostředníka Controller, který v tomto případě není vůbec spjat s životním cyklem prezentační vrstvy View. Tento krok umožňuje daleko lepší testování, ale za cenu rychlosti z důvodu manuálního vytvoření vazeb mezi daty a akcemi. [21]

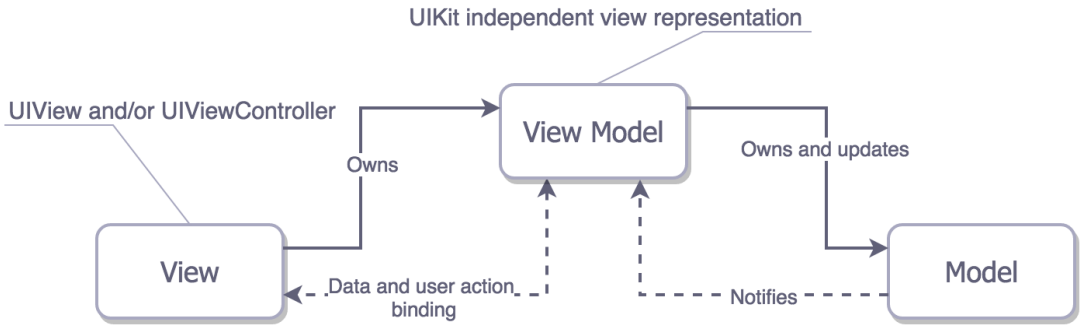


Obrázek 4 – MVP[20]

Nejvíce zodpovědnosti je rozděleno mezi Presenter a Model, zatímco View je jen prosté zobrazení uživatelského rozhraní, který zobrazuje pouze předaná data. [21]

3.1.3.3 MVVM

Model-View-ViewModel vychází z předešlých návrhů aplikační architektury, kde View a Model zůstávají stejné, avšak změna je u prostředníka mezi prezentační a modelovou vrstvou, který je v tomto návrhu označován jako ViewModel. Stejně tak mezi View a Model není žádný vztah, tedy navzájem o sobě nic neví a zároveň View je využíváno pouze jako prezentační vrstva, která jen aktualizuje objekty na obrazovkách aplikace. [21]

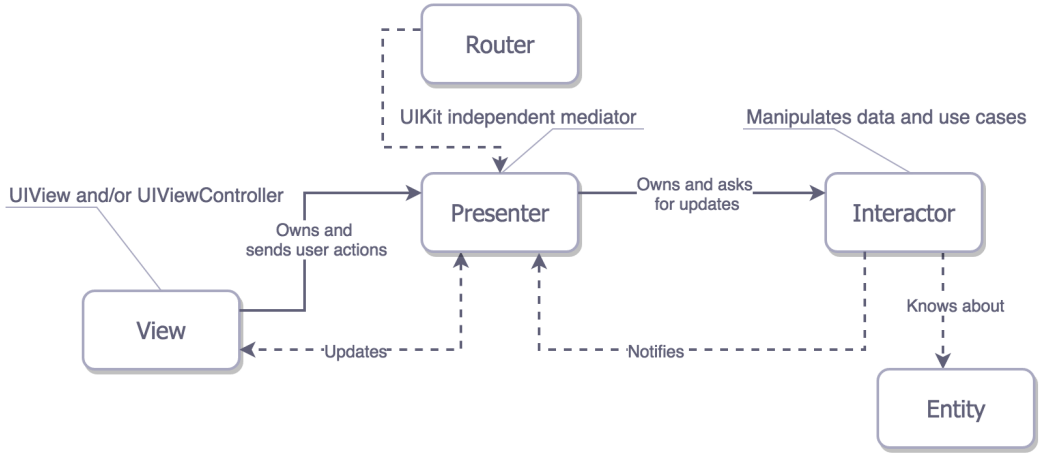


Obrázek 5 – MVVM[22]

V případě MVVM View naslouchá aktualizacím či změnám z uživatelské interakce nebo delegovaným voláním a následně volá funkce ViewModelu, aby provedl požadované aktualizace nebo změny. ViewModel následně buďto mění svojí vnitřní prezentaci nebo provádí změny v Modelu. View je napojeno na ViewModel a aktualizuje své zobrazení na základě detekce jakýchkoli změn. Tento proces je spjat s pojmem Bindings, který je viditelný na schématu výše. Bindings neboli vazby jsou realizované pomocí knihoven ReactiveCocoa nebo RxSwift, tedy reaktivního programování. [21][23]

3.1.3.4 VIPER

VIPER je adaptací takzvané Clean Architecture, tedy v překladu čisté architektury, pro aplikace pro operační systém iOS. Rozpadá se do více vrstev než výše zmíněné návrhy, a to na View, Interactor, Presenter, Entity a Routing. [21]



Obrázek 6 – VIPER[24]

View je vizualizací dat, které přijdou od entity Presenter a zároveň posílá zprávu zpět na základě uživatelské interakce.

Interactor obsahuje logiku související s daty nebo sdílením síťových dat, například získávání dat uložených na serverech.

Presenter obsahuje logiku uživatelského rozhraní. Komunikuje s View na základě zprávy získané od entity Interactor.

Entity se rozumí pouze o objekty reprezentující data.

Routing řídí přechody mezi jednotlivými VIPER moduly.

VIPER modul může být jedna obrazovka nebo celá funkcionalita aplikace. Distribuci jednotlivých zodpovědností má tento návrh aplikační architektury nejlépe pokryt, už jen z důvodů více implementovaných vrstev. Testování spolu souvisí s distribucí zodpovědností, čím lepší distribuce, tím lepší testování. Negativní stránkou použití tohoto návrhu je obrovské množství potřebného kódu. Naneštěstí existují nástroje, které pomáhají vytvářet šablony tříd a protokolů pro implementaci VIPER. [21][23]

3.2 Programovací jazyk Swift

V červnu roku 2014 na každoročně pořádané konferenci WWDC (Worldwide Developers Conference) představil Apple nový programovací jazyk Swift. Do tohoto data se využíval k vývoji aplikací pro iOS či Mac OS X platformy programovací jazyk Objective-C. Záměrem pro vytvoření nového programovacího jazyku bylo snazší, jednodušší, flexibilnější, rychlejší a uživatelsky přívětivější programování než s dosavadním Objective-C. [25]

Swift je objektově orientovaný, ve většině případů imperativní (v případě SwiftUI je deklarativní způsob programování) a kompilovaný programovací jazyk se syntaxí založenou na jazycích jako jsou například C#, F#, Java, Python nebo Ruby. Jeho syntaxe je více zjednodušená, jelikož nevyužívá žádné ukazatele (pointers) a obsahuje vylepšení v rámci datové struktury. Díky těmto odlišnostem je pro vývojáře méně pravděpodobné dopuštění se chybám při psaní kódu. [25]

Mezi jedny z charakteristik Swiftu patří deklarování proměnných, kdy za použití „var“ deklarujeme proměnnou, která se během kódu může měnit a za použití „let“ deklarujeme konstantu, která po čas celého kódu bude uchovávat pouze jednu hodnotu. Deklarace proměnných využívá statický typový systém, který striktně vyžaduje definování typů proměnných, kdy po deklaraci proměnné není možné změnit její datový typ. Dalším znakem je možnost volání funkcí s parametrem jiné funkce, který je typický pro funkcionální programování, které Swift také obsahuje. [25]

Vydání programovacího jazyku Swift doprovázelo také přidání nové technologie do integrovaném vývojovém prostředí (IDE) Xcode verze 6 nazvané Playground. Tato technologie je takzvaný „kompilátor v reálném čase“, tedy ihned po napsání kódu vývojářem je daný kód vykonán. Výhodou tohoto přístupu je velmi rychlý, jednoduchý a efektivní testování algoritmů, které nemusí být navázány na finální aplikaci. [25]

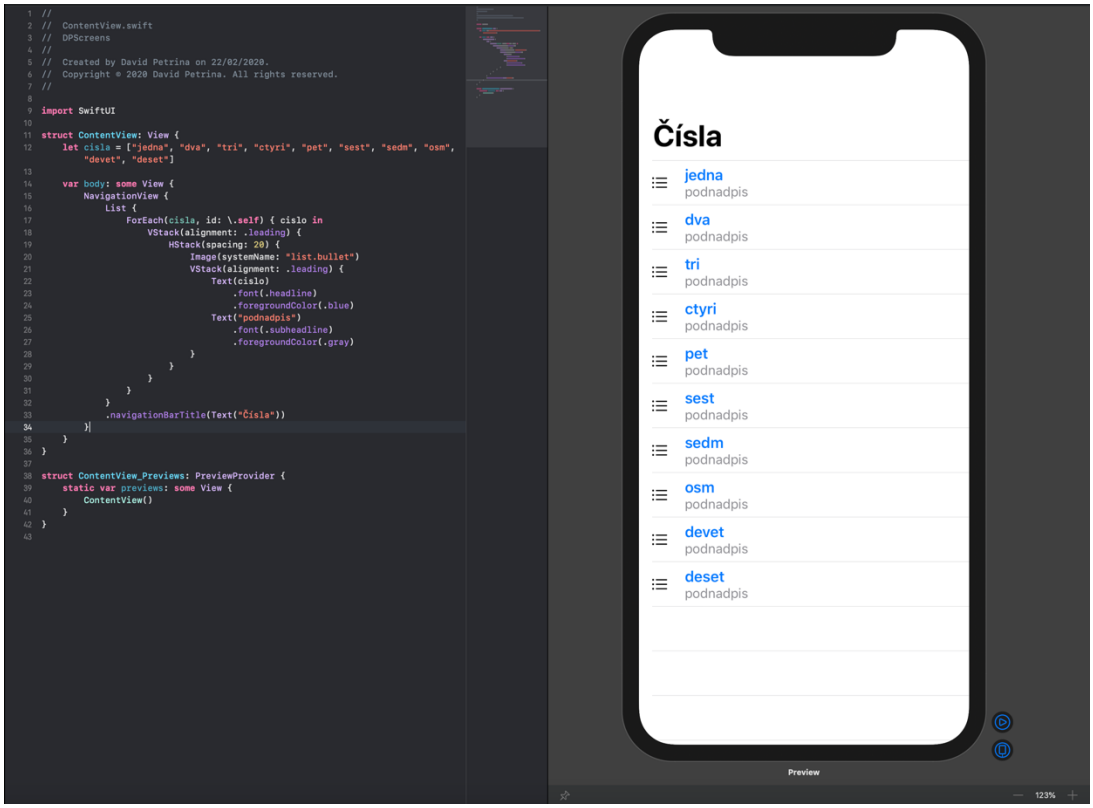
Zdrojový kód, hlášení chyb, fóra a vývojové buildy (kompilace zdrojového kódu) jsou ve Swiftu přístupny komukoli, což z tohoto programovacího jazyka dělá open source projekt. Díky tomu mohou jak zaměstnanci Apple, tak ale i nezávislí vývojáři a přispěvovatelé obohacovat a zdokonalovat tento programovací jazyk. [26]

3.2.1 Framework SwiftUI

Jeden z nejnovějších frameworků představený Applem v červnu 2019 a zpřístupněn pro vývojáře v beta verzi. O tři měsíce později byl vydán spolu s aktuální verzí iOS 13 pro širokou veřejnost. SwiftUI nabízí nový způsob programování aplikací pro operační systémy běžící na produktech od společnosti Apple, tedy přesněji pro iOS, macOS, tvOS a watchOS, který může být i pro začínající vývojáře snazší a uživatelsky přívětivější. [27]

Od frameworku UIKit, který byl a stále je využíván pro tvorbu iOS aplikací, se tento framework liší mimo jiné i způsobem programování. Z dosavadního imperativního způsobu, kdy vývojář musí popisovat výpočet pomocí posloupnosti příkazů a určovat přesný

algoritmus, který bude optimální pro vyřešení daného problému, se ve SwiftUI využívá deklarativní způsob programování. Jehož hlavní myšlenkou je, co se má udělat či zobrazit, ale ne jak se má daná věc udělat či zobrazit. Na obrázku níže je možné vidět způsob zápisu kódu, ve kterém je definován list (neboli seznam či tabulka), který je naplněn prvky z předem nadefinovaného pole spolu s obrázkem ze systémových ikon a podnadpisem. K těmto jednotlivým elementům je možné definovat vlastnosti, které říkají, jakým způsobem se mají zobrazovat. Výsledný kód je tímto z vysoké míry jednodušší, snazší k přečtení a zorientování se. [27]



Obrázek 7 - Ukázka kódu ve SwiftUI [Zdroj: autor]

Novinkou integrovanou v nejnovější verzi Xcode 11 v rámci SwiftUI projektů je náhled obrazovky aplikace v náhledu v reálném čase. Provedené změny v kódu se ihned propíší do obrazovky v náhledu a nabízí tak vývojáři okamžitý přehled o zobrazování nadefinovaných prvků ve výsledné aplikaci. Další možností je úprava jednotlivých elementů v náhledu, kdy tyto změny se ihned propíší do zdrojového kódu. Vývojář si také může

u náhledu přepnout na Live Preview, tedy živý náhled, který simuluje chování aplikace stejně jako kdyby běžela na reálném mobilním zařízení. [27]

Z důvodu využívání nejnovějších sad vývojových nástrojů je tento framework určený minimální verzi operačního systému iOS 13. Většina aplikací uvedených na App Store v dnešní době potřebuje ještě stále zpětnou kompatibilitu s dřívějšími verzemi iOS, a proto prozatím nebude moc aplikací vyvinutých ve SwiftUI. Apple zajistil integraci UIKit a SwiftUI, kdy je možné tyto dva frameworky kombinovat a aplikaci tak například pomalými kroky adaptovat do nového prostředí. [28]

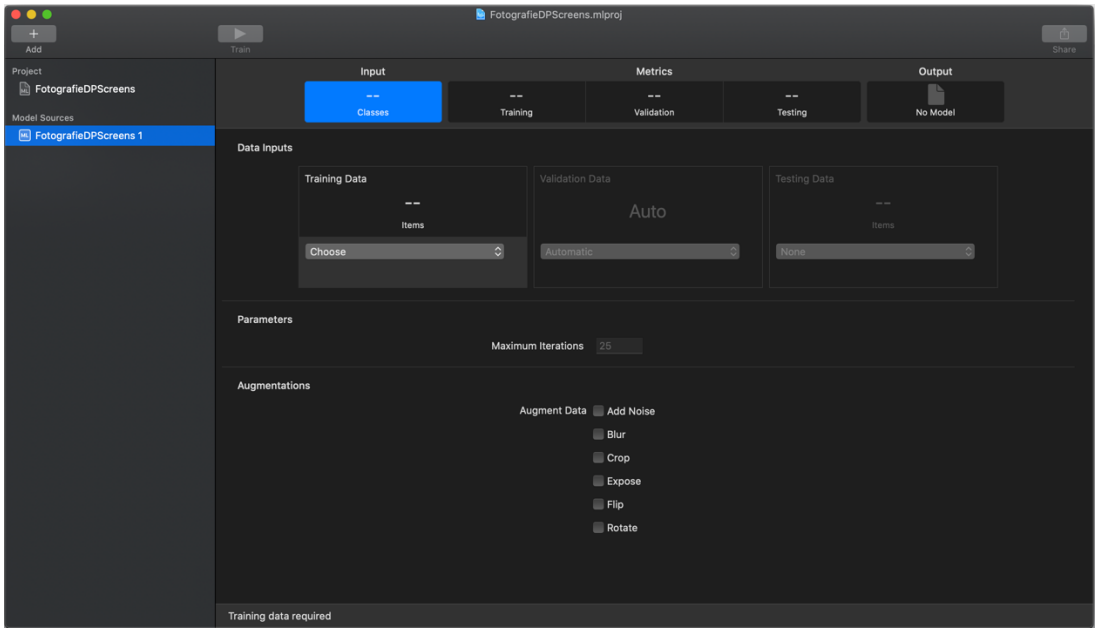
3.2.2 Framework Create ML

Základním principem tohoto frameworku je možnost vytvoření vlastního Machine Learning modelu. Dříve se proces tvorby modelu dělal v Xcode Playground, ale nyní je pro tyto účely dostupná nová aplikace Create ML. Tato aplikace je zahrnuta do vývojářských nástrojů v rámci nativního IDE Xcode. Její předností je nové uživatelsky přívětivé prostředí, ve kterém není potřeba ani jeden řádek napsaného kódu, ale trénování modelu je docílení právě přes grafické prvky aplikace. Spolu s tím také není žádná nutnost dovedností z oblasti strojového učení. [29]

Při spuštění Create ML aplikace se zobrazí jednotlivé typy modelů, které jsou možné trénovat. Mezi ně se řadí následující modely pro práci s: [30]

- fotografiemi, které buďto klasifikují fotografii na základě jejího obsahu nebo nachází různorodé objekty v rámci fotografie
- zvukem, které klasifikují nejvíce dominantní zvuky
- pohybem, které klasifikují obsah pohybových dat získaných z různých senzorů
- textem, které buďto klasifikují text založený na jeho obsahu a generují téma či kategorii nebo zvýrazňují slova v požadovaném textu
- tabulkami, které jsou nejvíce obecné a identifikují nejlepší z více modelů pro následné použití a klasifikování dat z tabulky

Po zvolení typu modelu, který vystihuje jeho výsledné funkce, je možné ihned přejít k procesu tvorby. Pro vytvoření modelu strojového učení pro rozpoznávání a vyhodnocování fotografií je grafické uživatelské rozhraní následující:



Obrázek 8 - Grafické uživatelské rozhraní Create ML aplikace [Zdroj: autor]

Pro spuštění testování a trénování modelu je nutné jako první krok deklarovat podkladová data. Tato data by měla být rozdělena do dvou oddělených složek, jedna složka s fotografiemi pro trénování modelu a druhá pro testování již vytvořeného modelu, a to ještě ideálně v poměru 80:20, tedy 80 % z celkových dat jako tréninková a 20 % z celkových dat jako testovací. Dále je také důležité, aby ani jedna z těchto rozdělení neměla identická data, což by mohlo vést nepřesnému vyhodnocení výsledného modelu. Jednoduchost programu spočívá v přetáhnutí složek s daty do polí Training Data a Testing Data a tím jsou podkladová data připravena ke spuštění trénování modelu pro strojové učení na vstupních datech. [31]

Nepovinnými volbami jsou možnosti Parameters a Augmentations. Parameters udává počet iterací, které model při trénování projede. V případě výsledné nízké tréninková přesnosti je nutné zvýšit počet iterací modelu z důvodu, že při současném počtu nebyla konfigurace schopna zachytit celkovou komplexnost dat. Platí tedy čím vyšší počet iterací, tím je tréninková přesnost modelu vyšší. Avšak zároveň spolu s tím souvisí úměra čím vyšší počet iterací, tím je validační přesnost nižší, jelikož model se při velkém množství iterací učí většímu počtu detailu a nezobecňuje. V rámci sekce Augmentations je možné rozšířit podkladová data o upravené obrázky. Například při zvolení oříznutí se pro každý obrázek z dat vytvoří nová, která bude oříznuta a zahrnuta do procesu trénování modelu. Tato metoda

se využívá v případě, kdy v tréninková datech není dostatečný počet obrázků a validační přesnost modelu nedosahuje vysokých hodnot. [31]

Pokud se po procesu tvorby modelu pro strojové učení zdají být výsledky uspokojivé, je možné výstupný soubor ve formátu .mlmodel uložit do počítače nebo rovnou vložit do projektu v Xcode, který je následně velmi jednoduché inicializovat a implementovat do zdrojového kódu pomocí frameworku CoreML. [31]

3.2.3 Framework Core ML

S příchodem iOS 11 se z hodně těžkého strojového učení stalo dostatečně možné a překvapivě mocné, a to vše díky frameworku Core ML. Apple se drží svého sloganu „Everyone Can Code“, což by se volně dalo přeložit jako „každý dokáže programovat“, a poskytuje mnoho podpůrných materiálů a aplikací pro každého člověka, který má zájem naučit se ovládat programovací jazyk Swift, kdy se hlavně zaměřují na studenty. [32] K tomuto sloganu se řadí i současná verze Core ML 3, díky které je možné implementovat model pro strojové učení do projektu, potažmo zdrojového kódu aplikace, pouze pomocí jednoho řádku kódu. Jak již bylo zmíněno v úvodní větě, strojové učení se tím stává dostupné jakémukoli vývojáři, i těm, kteří s tím nemají žádné zkušenosti. [33]

V momentě, kdy je v projektu v Xcode vložen ML model, je možné s ním následně pracovat. Při otevření modelu v Xcode jsou zobrazeny jeho metadata spolu s dalšími informacemi jako je například třída modelu, ve které jsou automaticky vygenerované její základní funkce pro analyzování a vyhodnocování dat. [31] Tento model je možné inicializovat v kódu následovně: [33]

```
let model = try VNCoreMLModel(for: NazevModelu().model)
```

Po této inicializaci je model strojového učení napojený na aplikaci a je možné s ním začít pracovat a provádět požadované úlohy. Framework Core ML je úzce spjat s frameworky Vision, díky kterému jsou analyzovány obrázky, Natural Language, se kterým je prováděno analyzování textů, Speech, který poskytuje konvertování audia na text a Sound Analysis, pomocí kterého je možné docílit identifikace zvuků v audiu. [33]

Hlavními výhodami Core ML 3 je integrace a analyzování modelů na samotném zařízení, kdy není potřeba žádný server, na kterém by se model vyhodnocoval a tím si získal první místo mezi frameworky strojového učení pro trénování modelu na zařízení. Ve výsledné aplikaci se to projevuje možností jejího využívání v režimu bez aktivního připojení

k internetu, tedy je možné si to představit na příkladu, kdy by bylo vyhodnocováno detekce předem nadefinovaných objektů v reálném čase pomocí fotoaparátu mobilního zařízení. Tato aplikace by díky implementaci Core ML 3 nemusela být připojena k internetu, aby při výskytu hledaného objektu na obrazovce zobrazila text s popisem tohoto objektu. Spolu s tím také souvisí bezpečnost, na které si Apple ve svých zařízeních velmi zakládá. Pořízená fotografie nemá, jakým způsobem se dostat ze zařízení, jelikož samotné analyzování a kvalifikování obsahu probíhá na fyzickém zařízení uživatele. [34]

Na druhé straně do jisté míry větší negativem je fakt, že pokud by bylo třeba změnit tréninkový model, který je zahrnutý v rámci aplikace, bylo by nutné provést jeho trénování pomocí nástroje Create ML s novými daty. Aktualizovaný model by opět bylo třeba nahrát do projektu aplikace, aktualizovat v kódu a v neposlední řadě provést aktualizaci aplikace v rámci App Store, ze kterého si uživatel stáhne aktuální verzi aplikace s novým modelem strojového učení. [34]

3.2.4 Framework Vision

V roce 2017 byl na konferenci WWDC představen nový framework pro počítačové vidění, který je jednoduché integrovat do vyvíjených aplikací a o dva roky později bylo na WWDC19 představeno rozšíření tohoto frameworku o nové funkcionality. Pomoví Vision je možné provádět detekci orientačních bodů na obličeji, detekci textu, rozpoznávání čárových kódů, registraci obrázků a sledování obecných funkcí. [35]

V nejnovější verzi Vision jsou k dispozici již připravené modely a metody, pomocí kterých je jednodušší a rychlejší implementovat chování aplikace. Mezi ně se řadí rozpoznávání textu, skenování dokumentů nebo vyhodnocování, zda se na fotografii nachází kočka nebo pes. Výhodou těchto již vytvořených modelů strojového učení je to, že vývojář nemusí ztrácet čas sbíráním podkladových dat a následného trénování modelu., pokud ovšem najde v aplikaci využití již implementovanými metodami. [35]

Při využívání vlastního modelu strojového učení pro rozpoznávání objektů na fotografiích pomocí Vision frameworku je zapotřebí zapouzdřit request (požadavek) do objektu `VNCoreMLRequest` a následně použít objekt `VNImageRequestHandler` k provedení požadavku. Model strojového učení procesuje vstupní fotografie ve fixním poměru stran, ale nahrané fotografie pro vyhodnocení mohou mít libovolné poměry stran a z toho důvodu musí Vision změnit velikost nebo oříznout fotografii. Pro dosažení

nejlepších výsledků je vhodné nastavit vlastnost `imageCropAndScaleOption` pro deklarovaný požadavek takovým způsobem, aby co nejvíce odpovídala rozložení fotografií, se kterými byl model trénován. [35][36]

Pro provedení požadavku je zapotřebí vytvoření objektu `VNImageRequestHandler` s parametrem zvolené fotografie ke klasifikaci a následného zavolání funkce `perform` s parametrem vytvořené proměnné požadavku. Tato celá metoda běží synchronně a z toho důvodu se musí obalit do fronty běžící na pozadí, aby hlavní fronta nebyla během provádění požadavků blokována: `DispatchQueue.global(qos: .userInitiated).async { }`. [36]

Finálním krokem pro umožnění aplikace strojovému učení je zobrazení výsledných vyhodnocení. Požadavek obsahuje vlastnost `results`, která při správném vyhodnocení bez chyby v procesu obsahuje objekty popisující možné klasifikace identifikované modelem strojového učení. [35][36]

3.3 UI specifikace

Mezi prvními kroky při vývoji aplikací je nutné vytvořit si specifikaci celého problému, která následně bude nápomocná při samotné implementaci softwaru. Tento postup je využíván nejen při vývoji mobilních aplikací, ale také při vývoji aplikací pro jakoukoli platformu či jejich typ. [37]

V prvotním stádiu se vyhotovují papírové prototypy, na kterých jsou zobrazeny jednotlivé obrazovky aplikace a spolu s tím i rozmístění prvků. Jedná se o hrubý náčrt výsledného stavu, avšak je důležitý z důvodu komunikace se zákazníkem. Po vyhotovení papírových prototypů se zákazníkovi předloží a vysvětlí se mu, co který prvek v aplikaci má na starost. Zákazník je tak schopen lépe si představit funkcionalitu a orientaci v aplikaci. V této fázi je tento krok kritický, jelikož je snadné zjistit případně nedorozumění v zadání a papírové prototypy předělat. V pozdějších fázích přípravy či vývoje by takovýto krok znamenal větší komplikace jak po časové stránce, tak i po finanční. [37]

Po schválených papírových prototypech zákazníkem je možné pokračovat v logických návrzích (nebo také označováno drátěné modely), které jsou již sofistikovanější a zobrazují již požadované rozmístění prvků po obrazovkách. Tyto návrhy jsou spojeny s use cases, tedy v českém označení případy užití. Ty popisují chování aplikace z uživatelského pohledu a definují, co uživatel očekává či požaduje od aplikace. S tím úzce souvisí scénáře, které

jsou naopak brány z pohledu systému na aplikaci. Ty udávají, co systém provede po příslušné uživatelské interakci. Wireframes spolu s use cases a scénáři jsou v rukou vývojáře stavebním kamenem pro implementaci funkcionalit aplikace. [37]

4 Vlastní práce – praktická část

4.1 Analýza požadavků

Před vývojem aplikace je nutné, aby byla sepsána analýza požadavků, která by se dala shrnout jako dokument obsahující veškerou funkcionalitu, chování prvků a obrazovek či hardwarovou náročnost plánované aplikace. Na základě tohoto dokumentu je možné provádět další kroky v procesu návrhu mobilní aplikace jako je například user interface specifikace.

Analýza požadavků je rozdělena na dvě skupiny, a to funkční a nefunkční požadavky. Funkční požadavky určují, jak se má aplikace chovat a jaké úkony má provádět. Nefunkční požadavky definují obecné omezení kladené na systém. Ke každému požadavku je dále uvedena jeho priorita, která může nabývat hodnot Critical (kritická), High (vysoká), Medium (střední) a Low (nízká). Při implementaci aplikace je vhodné dodržovat nepsanou zásadu, která hovoří, že před vydáním aplikace by měly být pokryty a implementovány veškeré požadavky s výjimkou priority Low. Tyto požadavky s nízkou prioritou by nijak neměly zasahovat do funkčnosti aplikace a mělo by se jednat spíše o grafické vady či menší nepřesnosti, které nejsou pro vydání aplikace stěžejní a mohou se dodělat při ostrém provozu aplikace. Posledním atributem ke každému požadavku je složitost, která se pohybuje na škále od 1 do 5, kdy hodnota 5 je nejvyšší možná složitost. Alternativou k této škále by mohlo být uvedení složitosti formou nacenění požadavku v hodinách, případně v manday (v pracovních dnech, které mají typicky každý po 8 hodinách), což se využívá ve většině reálných projektů.

Funkční požadavky jsou zachyceny v následující tabulce:

Tabulka 1 - Funkční požadavky [Zdroj: autor]

Název	Popis	Priorita	Složitost
Zobrazení seznamu psích plemen	Je nutné zobrazit seznam či tabulku s řádky reprezentující jednotlivá psí plemena.	Low	2
Zobrazení detailu psího plemene	Pro více informací o psím plemenu je nutné zobrazit obrazovku se základními informacemi.	Medium	2
Možnost vybrání fotografie z knihovny ze zařízení	Pro vyhodnocení plemene psa bude možné vybrat fotografii z knihovny fotografiích uložených na zařízení, potažmo uložených na iCloud.	Critical	3
Možnost pořízení fotografie	Pro vyhodnocení plemene psa bude možné vyfotit fotografii psa.	Critical	3
Zobrazení vybrané fotografie	Vybraná či vyfocená fotografie bude zobrazena na obrazovce pro ověření uživatelem k následnému vyhodnocení.	High	1
Vyhodnocení fotografie	Vstupní fotografie bude pomocí strojového učení analyzována a vyhodnotí plemeno psa vyskytující se na fotografii.	Critical	5
Zobrazení základních informací užívání aplikace	V rámci úvodní obrazovky budou zobrazeny základní informace o aplikaci a jejího používání .	Low	1

V následující tabulce jsou zobrazeny nadefinované nefunkční požadavky:

Tabulka 2 - Nefunkční požadavky [Zdroj: autor]

Název	Popis	Priorita	Složitost
Programovací jazyk a frameworky	Aplikace bude napsána v programovacím jazyce Swift za pomoci frameworků SwiftUI, CoreML a Vision.	High	2
Verze operačního systému	Z důvodu využití frameworku SwiftUI bude nutné minimální verze koncového zařízení iOS 13.	High	1
Hardware koncového zařízení	Koncové zařízení musí podporovat verzi operačního systému iOS 13.	Low	1
Připojení k internetu	Z důvodu lokálního načítání dat z JSON souboru a lokálně uložených fotografií není potřeba aktivní připojení k internetu při používání aplikace.	Low	1
Přístup k fotoaparátu zařízení	Pro pořizování fotografií pro následné vyhodnocení musí být uděleno povolení k přístupu k fotoaparátu.	Critical	2
Aplikační architektura	Aplikační architektura aplikace bude MVVM.	Medium	1
Načítání dat	Data budou načítána do datové vrstvy z lokálního JSON souboru.	High	3
Zajištění ML modelu	Model pro strojové učení bude vytvořen pomocí aplikace Create ML.	Critical	5

4.2 Sběr podkladových dat

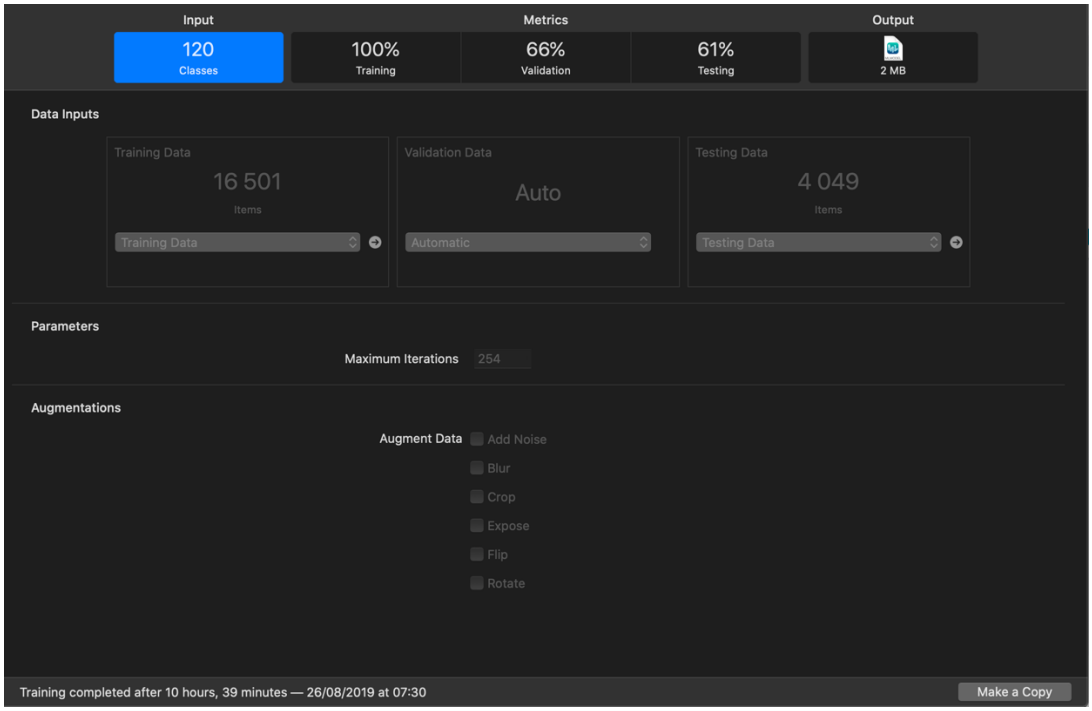
Pro vytvoření modelu pro strojové učení je nutné mít podkladová data, díky kterým můžeme stroj naučit požadovaným klasifikováním obrázků. Podkladová data pro vytvářenou aplikaci byla získána z veřejně dostupné databáze psích plemen z webových stránek Stanfordské univerzity [38]. Tento data set obsahuje celkem 20 580 fotografií 120 nejčastějších psích plemen, kdy každé plemeno má v průměru 150 různých fotografií.

Pro každé plemeno je nutné rozdělit data set na dvě skupiny, a to na tréninková a testovací data. Toto rozdělení bylo provedeno v poměru 80 % tréninková data a 20 % testovací data.

4.2.1 Tvorba Machine Learning modelu

S rozdělenými podkladovými daty je možné začít proces tvorby modelu pro strojové učení. Po vložení podkladových dat do aplikace Create ML je vhodné nastavit nepovinné atributy pro trénování modelu. Počet iterací byl zvolen na 256, což by mělo pokrýt obsáhlost dat a docílit tak vysoké přesnosti tréninkové fáze. Rozšíření modelu o upravené fotografie nebylo zvoleno žádné z parametrů.

Na obrázku níže lze vidět výsledné vyhodnocení modelu, kdy přesnost tréninkových dat je 100 %, validační přesnost modelu 66 % a přesnost vyhodnocení tréninkových dat je 61 %. Výstupný model je připravený k implementaci do projektu aplikace a vyhodnocování zadaných fotografií uživatelem.



Obrázek 9 - Vytvořený model pro strojové učení [Zdroj: autor]

4.3 Tvorba UI specifikace

V této kapitole je vytvořena User Interface specifikace, která poskytuje základní popis veškerých obrazovek aplikace, funkcionalitu a požadované zobrazení interakčních prvků z pohledu uživatele. Na základě těchto jednotlivých činnosti jsou vytvořeny scénáře, které vykonává systém. Poslední částí je ztvárnění těchto požadavků do wireframes. Díky těmto krokům je následná implementace aplikace pevně stanovená a ve výsledku by se tedy neměla lišit od zadání.

4.3.1 Úvodní obrazovka

Po zapnutí aplikace se uživateli zobrazí úvodní obrazovka, která obsahuje logo aplikace a krátký a stručný popis pro možnosti používání aplikace. Ve spodní části obrazovky je zobrazen Tab Bar, který slouží k navigaci mezi obrazovkami.

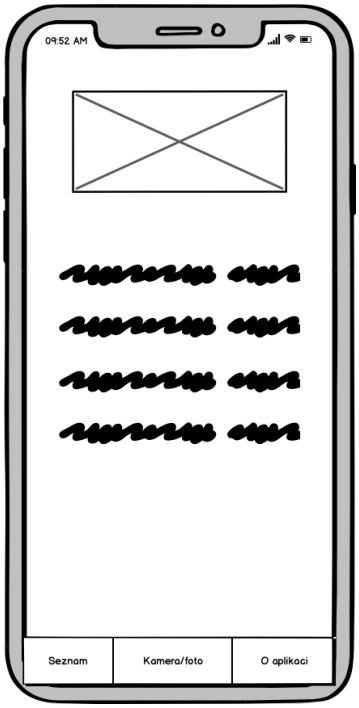
Use cases pro úvodní obrazovku:

- Uživatel očekává zobrazení loga aplikace spolu s informacemi o ovládání aplikace.

- Uživatel požaduje Tab Bar, kterým bude moci přepínat mezi obrazovkami se seznamem plemen, obrazovkou s možností vyfotit fotografii nebo vybrat fotografii z galerie a obrazovkou o aplikaci.

Scénáře pro úvodní obrazovku:

- Systém zobrazí logo ve formátu Image.
- Systém zobrazí informace o ovládání aplikace ve formátu String.
- Systém zobrazí TabView obsahující tři záložky:
    - seznam plemen, po jejímž kliknutí systém zobrazí obrazovku se seznamem plemen,
    - kameru / foto, po jejímž kliknutí systém zobrazí obrazovku s volbou vyfocení fotografie či její vybrání z galerie,
    - o aplikaci, po jejímž kliknutí systém zobrazí obrazovku se základními informacemi o aplikaci.



Obrázek 10 - Wireframe úvodní obrazovky [Zdroj: autor]

4.3.2 Seznam plemen

Na záložce se seznamem plenem je zobrazena tabulka všech plemen, se kterými aplikace pracuje a v jednotlivých řádcích zobrazuje ikonu plemene, jeho název a na pravé

straně textem s ikonkou pro zobrazení detailní obrazovky o vybraném plemenu. Dále je na obrazovce zobrazen titul stránky a pod ním pole pro vyhledávání plemena. Ve spodní části obrazovky je zobrazen Tab Bar, který slouží pro navigaci mezi jednotlivými obrazovkami aplikace.

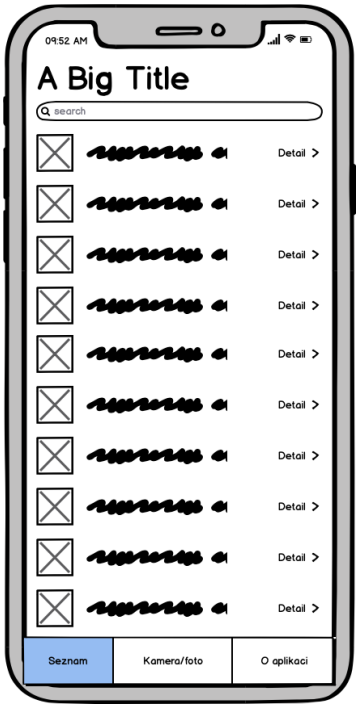
Use cases pro obrazovku se seznamem plemen:

- Uživatel očekává nadpis obrazovky.
- Uživatel očekává pole pro vyhledávání, díky kterému může filtrovat plemena dle zadaného textu.
- Uživatel očekává zobrazení seznamu plemen, ve kterém požaduje:
    - zobrazení obrázku plemena,
    - název plemena,
    - text a ikonku pro intuitivní přechod na detail plemena.
- Uživatel požaduje po kliknutí na řádek s plemenem přechod na obrazovku s detailem plemena.
- Uživatel požaduje Tab Bar, kterým bude moci přepínat mezi obrazovkami se seznamem plemen, obrazovkou s možností vyfotit fotografii nebo vybrat fotografii z galerie a obrazovkou o aplikaci.

Scénáře pro obrazovku se seznamem plemen:

- Systém zobrazí titul obrazovky v rámci NavigationView.
- Systém zobrazí pole pro vyhledávání (SearchBar), do kterého bude uživatel schopen napsat textový řetězec.
- Systém zobrazí seznam plemen z datového zdroje ve formátu JSON v rámci List.
    - Systém zobrazí obrázek plemena ve formátu Image.
    - Systém zobrazí název plemena ve formátu String.
    - Systém zobrazí text spolu s ikonou pro přechod na detail zvoleného plemena ve formě String a Image.
    - Po kliknutí na prvek ze seznamu systém zobrazí obrazovku s detailem plemena.
- Systém zobrazí TabView obsahující tři záložky:

- seznam plemen, která je aktivní a lehce barevně odlišena,
- kameru / foto, po jejímž kliknutí systém zobrazí obrazovku s volbou vyfocení fotografie či její vybrání z galerie,
- o aplikaci, po jejímž kliknutí systém zobrazí obrazovku se základními informacemi o aplikaci.



Obrázek 11 - Wireframe obrazovky se seznamem plemen [Zdroj: autor]

4.3.3 Seznam plemen – hledání

Tato obrazovka je stejná jako je tomu u předchozí podkapitoly jen s tím rozdílem, že uživatel klikl do pole pro vyhledávání z důvodu filtrace požadovaného plemena na základě zadaného textu.

Use cases pro obrazovku se seznamem plemen s aktivním vyhledáváním:

- Uživatel očekává po kliknutí do pole pro vyhledávání možnost zadání textu pro vyhledání plemena a požaduje:
  - zobrazení klávesnice po kliknutí do pole pro vyhledávání,
  - zobrazení vyfiltrovaných plemen na základě zadaného textu.
- Uživatel očekává zobrazení tlačítka pro zrušení vyhledávacího módu a přechodu zpět na celý seznam plemen.

- Uživatel požaduje po kliknutí na řádek s plemenem přechod na obrazovku s detailem plemena.

Scénáře pro obrazovku se seznamem plemen s aktivním vyhledáváním:

- Systém po kliknutí na vyhledávací pole SearchBar zaktivní prvek, do kterého je možné psát text ve formě String.
- Systém zobrazí klávesnici pro psaní do vyhledávacího pole.
- Systém zobrazí vyfiltrované položky ze seznamu plemen z datového zdroje ve formátu JSON v rámci List na základě zadaného textu v SearchBar ve formě String.
    - Systém zobrazí obrázek vyfiltrovaného plemena ve formátu Image.
    - Systém zobrazí název vyfiltrovaného plemena ve formátu String.
    - Systém zobrazí text spolu s ikonou pro přechod na detail zvoleného plemena ve formě String a Image.
    - Po kliknutí na prvek z vyfiltrovaného seznamu systém zobrazí obrazovku s detailem plemena.
- Systém zobrazí tlačítko pro zrušení vyhledávacího módu.
    - Po kliknutí na tlačítko systém zobrazí původní seznam všech plemen.

Obrázek 12 - Wireframe obrazovky s vyhledáváním plemen [Zdroj: autor]

4.3.4 Detail plemene

Po kliknutí na řádek s plemenem v předešlé obrazovce je zobrazena obrazovka s detailem daného plemena, na které je zobrazeno tlačítko pro návrat zpět na seznam plemen, obrázek a základní informace o plemenu. Ve spodní části obrazovky je zobrazen Tab Bar, který slouží pro navigaci mezi jednotlivými obrazovkami aplikace.

Use cases pro obrazovku s detailem plemene:

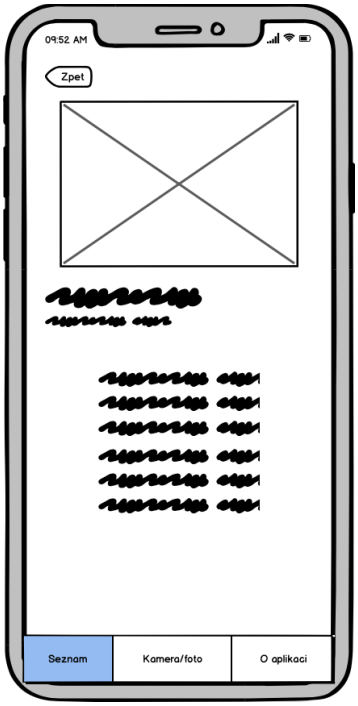
- Uživatel očekává zobrazení:
    - tlačítka zpět na návrat na seznam plemen.
    - obrázku vybraného plemena.
    - základních informací o zvoleném plemenu:
        - název,
        - velikost,
        - výšku,
        - váhu,
        - barvu,
        - průměrný věk,
        - zemi původu.
- Uživatel požaduje Tab Bar, kterým bude moci přepínat mezi obrazovkami se seznamem plemen, obrazovkou s možností vyfotit fotografii nebo vybrat fotografii z galerie a obrazovkou o aplikaci.

Scénáře pro obrazovku s detailem plemene:

- Systém zobrazí tlačítko zpět, po jehož kliknutí se zobrazí seznam plemen s aktivním vyhledáváním.
- Systém zobrazí fotografii psa ve formátu Image.
- Systém zobrazí základní informace o plemenu ve formátu String:
    - název,
    - velikost,
    - výšku,
    - váhu,
    - barvu,

- průměrný věk,
- zemi původu.

- Systém zobrazí TabView obsahující tři záložky:
  - seznam plemen, která je aktivní a lehce barevně odlišená. Po jejímž kliknutí systém zobrazí obrazovku se seznamem všech plemen,
  - kameru / foto, po jejímž kliknutí systém zobrazí obrazovku s volbou vyfocení fotografie či její vybrání z galerie,
  - o aplikaci, po jejímž kliknutí systém zobrazí obrazovku se základními informacemi o aplikaci.



Obrázek 13 - Wirerame obrazovky detailu plemene [Zdroj: autor]

4.3.5 O aplikaci

Na obrazovce O aplikaci je zobrazen text, který popisuje aplikaci a dává možnost uživateli zaslat připomínky či nahlásit chyby na emailovou adresu. Ve spodní části obrazovky je zobrazen Tab Bar, který slouží pro navigaci mezi jednotlivými obrazovkami aplikace.

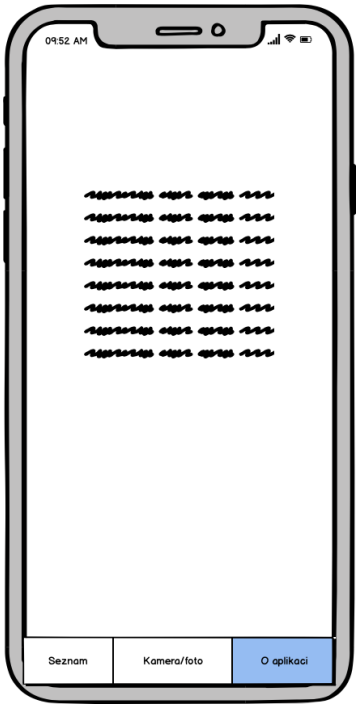
Use cases pro obrazovku o aplikaci:

- Uživatel očekává zobrazení základních informací.

- Uživatel požaduje Tab Bar, kterým bude moci přepínat mezi obrazovkami se seznamem plemen, obrazovkou s možností vyfotit fotografii nebo vybrat fotografii z galerie a obrazovkou o aplikaci.

Scénáře pro obrazovku o aplikaci:

- Systém zobrazí informace o aplikaci ve formátu String.
- Systém zobrazí TabView obsahující tři záložky:
    - seznam plemen, po jejímž kliknutí systém zobrazí obrazovku se seznamem plemen,
    - kameru / foto, po jejímž kliknutí systém zobrazí obrazovku s volbou vyfocení fotografie či její vybrání z galerie,
    - o aplikaci, která je aktivní a lehce barevně odlišena od ostatních.



Obrázek 14 - Wireframe obrazovky o aplikaci [Zdroj: autor]

4.3.6 Volba kamery nebo knihovny fotografií

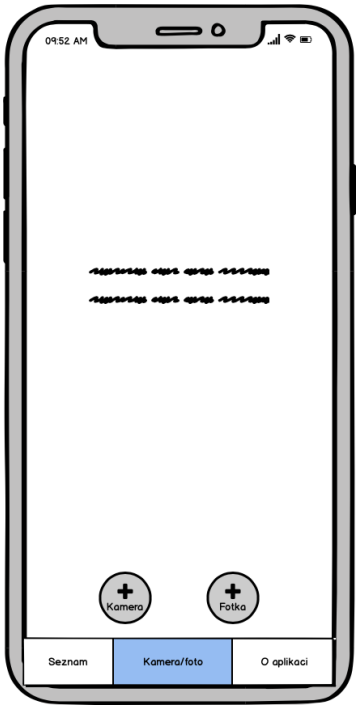
Na této obrazovce si uživatel vybírá, zda bude chtít pořídit fotografii pomocí kamery nebo vybrat fotografii z galerie fotografií.

Use cases pro obrazovku s volbou kamery nebo knihovny fotografií:

- Uživatel očekává zobrazení textu, který bude objasňovat kroky, které může provést.
- Uživatel očekává zobrazení tlačítka pro vyfocení fotografie.
- Uživatel očekává zobrazení tlačítka pro vybrání fotografie z galerie fotografií.
- Uživatel požaduje Tab Bar, kterým bude moci přepínat mezi obrazovkami se seznamem plemen, obrazovkou s možností vyfotit fotografii nebo vybrat fotografii z galerie a obrazovkou o aplikaci.

Scénáře pro obrazovku s volbou kamery nebo knihovny fotografií:

- Systém zobrazí text objasňující kroky, které může uživatel provést, ve formátu String.
- Systém zobrazí tlačítko pro pořízení fotografie.
    - Po kliknutí na tlačítko systém zobrazí kameru zařízení, díky které bude schopen uživatel pořídit fotografii.
    - Při prvním otevření kamery systém zobrazí hlášku o povolení přístupu ke kameře.
- Systém zobrazí tlačítko pro vybrání fotografie z galerie fotografií.
    - Po kliknutí na tlačítko systém zobrazí galerii fotografií uložených na zařízení, ve které uživatel může vybrat fotografii.
- Systém zobrazí TabView obsahující tři záložky:
    - seznam plemen, po jejímž kliknutí systém zobrazí obrazovku se seznamem plemen.
    - kameru / foto, po jejímž kliknutí systém zobrazí obrazovku s volbou vyfocení fotografie či její vybrání z galerie.
    - o aplikaci, po jejímž kliknutí systém zobrazí obrazovku se základními informacemi o aplikaci.



Obrázek 15 - Wireframe obrazovky pro volbu fotografie [Zdroj: autor]

4.3.7 Zvolená fotografie pro vyhodnocení plemene

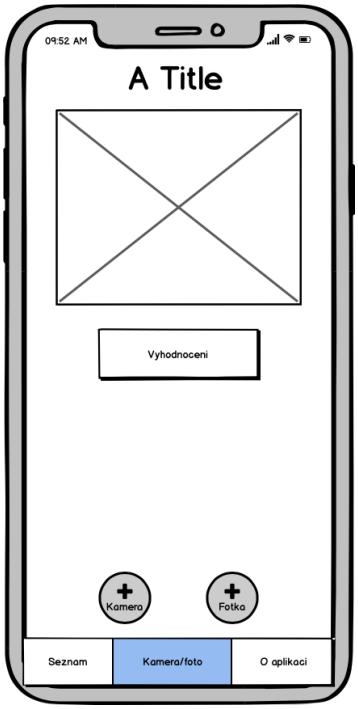
Po vyfocení plemena psa nebo vybrání z galerie fotografií se uživateli zobrazí obrazovka, která bude obsahovat danou fotografii, tlačítko pro spuštění vyhodnocení plemene a zůstanou zobrazeny tlačítka pro pořízení fotografie nebo výběr z galerie.

Use cases pro obrazovku se zvolenou fotografií pro vyhodnocení plemena:

- Uživatel očekává zobrazení:
  - nadpisu stránky,
  - vyfocené či vybrané fotografie,
  - tlačítka pro vyhodnocení plemene,
    - po kliknutí na tlačítko požaduje zobrazení obrazovky s vyhodnocením plemena psa.
  - tlačítka pro vyfocení fotografie,
  - tlačítka pro vybrání fotografie z galerie fotografií.
- Uživatel požaduje Tab Bar, kterým bude moci přepínat mezi obrazovkami se seznamem plemen, obrazovkou s možností vyfotit fotografii nebo vybrat fotografii z galerie a obrazovkou o aplikaci.

Scénáře pro obrazovku se zvolenou fotografií pro vyhodnocení plemena:

- Systém zobrazí nadpis obrazovky ve formátu String.
- Systém zobrazí vyfocenou/vybranou fotografii ve formátu Image.
- Systém zobrazí tlačítko pro vyhodnocení plemena z fotografie.
    - Po kliknutí na tlačítko systém provede klasifikaci fotografie na základě Machine Learning modelu a zobrazí obrazovku s výsledným vyhodnocením plemena psa z fotografie.
- Systém zobrazí tlačítko pro pořízení fotografie.
    - Po kliknutí na tlačítko systém zobrazí kameru zařízení, díky které bude schopen uživatel pořídit fotografii.
- Systém zobrazí tlačítko pro vybrání fotografie z galerie fotografií.
    - Po kliknutí na tlačítko systém zobrazí galerii fotografií uložených na zařízení, ve které uživatel může vybrat fotografii.
- Systém zobrazí TabView obsahující tři záložky:
    - seznam plemen, po jejímž kliknutí systém zobrazí obrazovku se seznamem plemen,
    - kameru / foto, po jejímž kliknutí systém zobrazí obrazovku s volbou vyfocení fotografie či její vybrání z galerie,
    - o aplikaci, po jejímž kliknutí systém zobrazí obrazovku se základními informacemi o aplikaci.



Obrázek 16 - Wireframe obrazovky se zvolenou fotografií [Zdroj: autor]

4.3.8 Vyhodnocení plemene

Prvky na obrazovce jsou totožné těm, které jsou uvedené na obrazovce se zvolenou fotografií. Rozdílem je, že po kliknutí na tlačítko pro vyhodnocení plemene psa je zobrazen text identifikující plemeno a řádek s výsledkem, po jehož kliknutí se uživatel dostane na obrazovku s detailem vyhodnoceného psa.

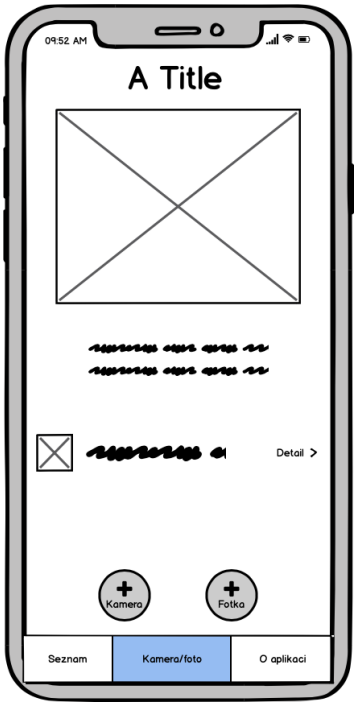
Use cases pro obrazovku s vyhodnocením plemene:

- Uživatel očekává zobrazení:
    - nadpisu stránky,
    - vyfocené či vybrané fotografie,
    - textu s výsledným vyhodnocením plemena psa z fotografie,
    - řádku s vyhodnoceným plemenem ze seznamu plemen,
        - po kliknutí na řádek uživatel požaduje přechod na obrazovku s detailem plemena.
    - tlačítka pro vyfocení fotografie,
    - tlačítka pro vybrání fotografie z galerie fotografií.

- Uživatel požaduje Tab Bar, kterým bude moci přepínat mezi obrazovkami se seznamem plemen, obrazovkou s možností vyfotit fotografii nebo vybrat fotografii z galerie a obrazovkou o aplikaci.

Scénáře pro obrazovku s vyhodnocením plemene:

- Systém zobrazí nadpis stránky ve formátu String.
- Systém zobrazí vyfocenou/vybranou fotografii ve formátu Image.
- Systém zobrazí textový řetězec s klasifikovaným plemenem na základě fotografie ve formátu String.
- Systém zobrazí řádek s vyhodnoceným plemenem ze seznamu plemen z datového zdroje ve formátu JSON.
    - Systém zobrazí obrázek vyhodnoceného plemena ve formátu Image.
    - Systém zobrazí název vyhodnoceného plemena ve formátu String.
    - Systém zobrazí text spolu s ikonou pro přechod na detail zvoleného plemena ve formě String a Image.
    - Po kliknutí na vyhodnocený řádek plemena systém zobrazí obrazovku s detailem daného plemena.



Obrázek 17 - Wireframe obrazovky s vyhodnocením plemene [Zdroj: autor]

4.4 Implementace aplikace

Po provedení veškerých analytických částí zahrnujících analýzu požadavků a s ní související specifikaci uživatelského rozhraní je možné zahájit implementování aplikace v integrovaném vývojovém prostředí Xcode. V ideálním případě by se programování aplikace mělo držet co největší měrou specifikaci, ve které jsou definována veškerá chování a funkcionality.

Mobilní aplikace je implementována deklarativním programováním za použití frameworku SwiftUI. Výslednou aplikační architekturou je zvoleno MVVM, kdy jsou separovány datové, prezentační a řídící vrstvy.

4.4.1 Datová vrstva

Jedním z prvních kroků při implementaci aplikace je vytvoření jejího datového modelu, respektive datové vrstvy, která bude uchovávat a předávat informace a data na jednotlivé obrazovky. Datový model byl vytvořený následujícím způsobem:

```
struct Breed : Codable, Identifiable, Comparable {

    static func < (lhs: Breed, rhs: Breed) -> Bool {
        lhs.breedCZ < rhs.breedCZ
    }

    var id: Int
    var breed: String
    var breedCZ: String
    var fur: String
    var size: String
    var weight: String
    var height: String
    var color: String
    var avgAge: String
    var originCountry: String
    fileprivate var imageName: String
}
extension Breed {
    var image: Image {
        ImageStore.shared.image(name: imageName)
    }
}
```

Datové modely se vytváří buďto jako struktura nebo třída. Pro tento model byla vybrána struktura z důvodu využívání jednoduchých datových typů. Zároveň bylo nutné rozšířit strukturu o tři protokoly a to `Codable`, `Identifiable` a `Comparable`.

Díky protokolu `Codable` je možné provádět kódování a dekódování dat serializovaného formátu jako je například JSON. Datová základna byla vytvořena v souboru formátu JSON, který je v současné době jedním z nejvyužívanějších datových formátů pro výměnu dat. Při každém otevření aplikace se data načtou do modelu a jsou ihned k dispozici pro prezentační vrstvy k zobrazení požadovaných informací.

Protokol `Identifiable` zajišťuje identifikaci a zobrazování unikátních dat v rámci prezentačních vrstev. Pro představu na reálném příkladu z implementované aplikace je možné poukázat na zobrazení tabulky se seznamem psích plemen, ve které jsou zobrazovány jednotlivé řádky tabulky pomocí ForEach cyklu. Tento cyklus prochází celý datový model, respektive načtená data ze souboru JSON. K tomu, aby byly správně zobrazeny jednotlivé řádky tabulky, je třeba identifikovat dílčí záznamy, což se provádí pomocí porovnávání unikátního ID čísla jednotlivých záznamů.

Posledním použitým protokolem pro rozšíření struktury modelu byl protokol `Comparable`, který umožnuje využívání operandů >, <, >= a <= pro porovnání jednotlivých datových typů. Porovnávání bylo nutné provést z důvodu seřazení prvků v poli obsahující veškerá načtená data z JSON souboru abecedně dle proměnné `breedCZ`, tedy dle jména plemene v českém jazyce. Díky tomuto bylo možné zobrazovat seřazená plemena v tabulce seznamu všech plemen v abecedním řazení.

4.4.2 Seznam psích plemen

V momentě, kdy je datová vrstva v aplikace implementována a data jsou k dispozici k zobrazování, je možné přejít k tvorbě prezentačních vrstev pro zobrazování seznamu plemen a jejich obrazovek s detailem. V této problematice se řešení implementace pomocí SwiftUI osvědčilo jako jednoduché, snadné a velmi čitelné.

Nejdříve byla vytvořena struktura `BreedRow` pro zobrazení samotného řádků seznamu, ve které se přiřadila data z modelu do proměnné `breed` a posléze bylo možné jejich zobrazení na obrazovce.

```
struct BreedRow: View {
    var breed : Breed

    var body: some View {
        HStack {
            breed.image
                .resizable()
```

```
                .aspectRatio(contentMode: .fill)
                .frame(width: 60, height: 60)
                .clipShape(Circle())
            Text(breed.breedCZ)
            Spacer()
            Text("Detail")
        }
        .padding()
    }
}
```

Pomocí `HStack` je zajištěno, že prvky v něm obsažené se budou zobrazovat na obrazovce horizontálně vedle sebe. Tímto způsobem jsou zobrazovány prvky malé fotografie psího plemene, které jsou zmenšeny a zobrazeny v kruhové podobě, dále text zobrazující název plemene v českém jazyce. Text „Detail“ je použit pro snazší uživatelské pochopení možnosti přechodu na obrazovku s vybraným plemenem a jeho základními informacemi. Mezi prvními dvěma a třetím prvkem je deklarován `Spacer()`, který elementům říká, aby mezi sebou udělali nezbytně velkou mezeru a tím se posunuli ke kraji obrazovky.

S hotovou implementací zobrazování jednotlivých řádků je možné se přesunout k vytvoření nové struktury pro DatabaseView, tedy prezentační vrstva pro zobrazování seznamu psích plemen.

```
struct DatabaseView: View {
    @State private var searchQuery: String = ""

    var body: some View {
        NavigationView {
            VStack {
                List {
                    SearchBar(text: $searchQuery, placeholder: "Vyhledat
plemeno...")

                    ForEach(breedData.filter { searchQuery.isEmpty ? true
: "\($0)".localizedCaseInsensitiveContains(searchQuery)}.sorted()) {
breed in
                        NavigationLink(destination: BreedDetail(breed:
breed)) {

                            BreedRow(breed: breed)
                        }
                    }
                }
                .onAppear(perform: {
                    UITableView.appearance().separatorStyle = .singleLine
                    UITableView.appearance().separatorColor = .orange
                })
            }
```

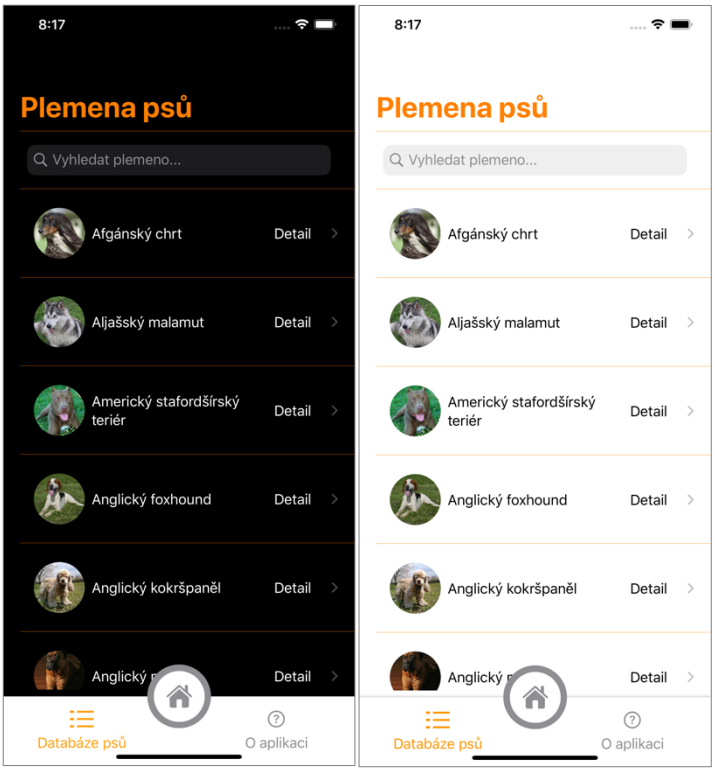
```
            .navigationBarTitle(Text("Plemena
psů").foregroundColor(.orange))
            .resignKeyboardOnDragGesture()
        }
    }
}
```

Na počátku struktury je deklarovaná proměnná `searchQuery`, která uchovává text napsaný uživatel při vyhledávání plemene pomocí vyhledávacího pole. Před deklaraci této proměnné je přidána vlastnost `@State`, která zajišťuje, že SwiftUI bude automaticky sledovat změny a aktualizovat části kódu, kde je tato proměnná použitá.

V rámci `List`, což je zobrazení seznamu či lépe řečeno tabulky, je volána struktura `SearchBar` implementující logiku při vyhledávání, která je definována separátně z důvodu zachování přehlednosti a principů aplikační architektury MVVM. Hlavní komponentou je v tomto zobrazení `ForEach` cyklus, který pro každý záznam v datové vrstvě zobrazuje `BreedRow` pro příslušné psí plemeno.

Řádky seznamu jsou vnořeny do `NavigationLink`. Po klinutí na řádek zobrazení implementované struktury je pomocí parametru `destination: BreedDetail()` zobrazena obrazovka `BreedDetail`. V této struktuře jsou zobrazeny základní informace o plemenu načtené z datové vrstvy (příloha č. 1). Je zde krásný příklad deklarativního způsobu programování, kdy vývojář přesně definuje, jak se mají prvky zobrazovat a jak mají být rozvrženy po obrazovce.

Další z předností SwiftUI je automatické přepínání barev implementovaných prvků pro světlé a tmavé režimy. Není potřeba nikde v kódu ani ve vlastnostech aplikace nastavovat, jak se má aplikace zobrazovat pro jeden z režimů, ale na základě napsaného zdrojového kódu se sama adaptuje na uživatelem zvolený mód jeho mobilního zařízení. Pokud by aplikace měla nadefinovanou grafickou stránku, která by byla neměnná, pak by bylo třeba ji implementovat do kódu aplikace, aby se pro oba módy aplikace zobrazovala stejně. Hlavním prvkem, u kterého by toto bylo třeba ošetřit, by byl `Text`, který se bez vývojářských změn automaticky adaptuje na světlý nebo tmavý režim.



Obrázek 18 – Seznam plemen v Dark mode a v Light mode [Zdroj: autor]

Hlavní barvou prvků v aplikaci byla vybrána oranžová barva v hexadecimálním zápisu ff9500, která koresponduje se systémovou oranžovou barvou pro operační systém iOS.

4.4.3 Implementace strojového učení

Pro účely implementace modelu strojového učení a klasifikování fotografií bylo potřeba vytvořit třídu, která bude zajišťovat celý proces. Jelikož není v současné chvíli možné implementovat strojové učení pomocí frameworků CoreML a Vision v rámci kódu SwiftUI, je třeba jeho implementace s UIKit. Pro účely aplikace byla vytvořena třída `ImageClassificationModel`:

```
final class ImageClassificationModel : ObservableObject { }
```

K této třídě je nutné přidat protokol `ObservableObject` z frameworku `Combine`, který pomocí vlastností `@Published` ze SwiftUI automaticky oznamuje změny vlastností veškerým prezentačním obrazovkám, které používají tento objekt a díky tomu jsou synchronizovány se svými daty. V této třídě jsou implementovány tři proměnné, které slouží

pro uložení vybrané fotografie ke klasifikaci, vyhodnocení klasifikace a názvu nejpodobnějšího plemene:

```
    @Published var image : UIImage? = nil
    @Published var classificationText : String = ""
    @Published var topClassBreed : String = ""
```

Prvním krokem pro vytvoření logiky strojového učení bylo vytvoření požadavku pro klasifikování. V rámci této proměnné byl inicializován model strojového učení a deklarování samotného požadavku s parametrem implementovaného modelu. Celá proměnná byla deklarována jako `private lazy`. Pomocí `private` je možné s proměnnou pracovat pouze v rámci třídy, ve které byla deklarována a pomocí `lazy` je zajištěno, že proměnná a její kód nebude provedena ihned po načtení třídy, ale až v okamžiku, kdy se proměnná v kódu volá.

```
    private lazy var classificationRequest : VNCoreMLRequest = {
        do {
            let model = try VNCoreMLModel(for: Dedog().model)
            let request = VNCoreMLRequest(model: model,
completionHandler: { [weak self] request, error in
                self?.processClassification(for: request, error: error)
            })
            request.imageCropAndScaleOption = .scaleFit
            return request
        } catch {
            fatalError("Nepodařilo se načíst Core ML model: \(error)")
        }
    }()
```

Tato proměnná je takzvaná „computed property“, což znamená, že se proměnná chová jako metoda a vrací hodnotu na základě jejích vlastností. V rámci deklarace požadavku je volána metoda `processClassification()`, která provádí klasifikaci fotografií a přiřazuje výsledné hodnoty do deklarovaných proměnných.

```
    private func processClassification(for request: VNRequest, error:
Error?) {
        DispatchQueue.main.async {
            guard let results = request.results else {
self.classificationText = "Nepodařilo se klasifikovat
obrázek.\n\(error!.localizedDescription)"; return }
            let classifications = results as!
[VNClassificationObservation]
            if classifications.isEmpty {
                self.classificationText = "Nic nebylo rozpoznáno."
```

```
            } else {
                let topClassifications = classifications.prefix(2)
                let descriptions = topClassifications.map {
classification in
                    return String(format: "%.0f%% %@",
classification.confidence * 100, classification.identifier)
                }
                self.classificationText = descriptions.joined(separator:
"\n")
                let topOneClass = classifications.prefix(1)
                let topOneDesc = topOneClass.map { classification in
                    return String(classification.identifier)
                }
                self.topClassBreed = topOneDesc.joined()
            }
        }
    }
```

Metoda klasifikování fotografie vrací výsledky vyhodnocení v podobě pole objektu `VNClassificationObservation`. Tyto výsledky mají dvě dílčí proměnné: identifier, tedy identifikátor, který reprezentuje třídu detekovaného objektu, a confidence, tedy spolehlivost, která udává pravděpodobnost této třídy.

Po implementaci požadavku a vyhodnocování fotografie je možné se přesunout k napsání kódu pro provedení požadavku po uživatelem zvolené fotografii. Pomocí metody `updateClassification()` je zajištěn převod vstupní fotografie z formátu UIImage do formátu CIImage, který je nezbytný pro prováděný algoritmů v rámci frameworků pro strojové učení. S výslednou fotografií ve formátu CIImage a zvolenou orientací je implementován `handler` korespondující objektu `VNImageRequestHandler`, na kterém je možno volat funkci `perform`, která spouští kód implementovaný v rámci proměnné `classificationRequest`.

```
    func updateClassification() {
        self.classificationText = "Vyhodnocování..."
        let orientation =
CGImagePropertyOrientation(self.image!.imageOrientation)
        guard let ciImage = CIImage(image: self.image!) else {
fatalError("Nepodařilo se vytvořit \(CIImage.self) z \(String(describing:
image))!")}
        DispatchQueue.global(qos: .userInitiated).async {
            let handler = VNImageRequestHandler(ciImage: ciImage,
orientation: orientation)
            do {
                try handler.perform([self.classificationRequest])
            } catch {
                print("Nepodařilo se provést
vyhodnocení.\n\(error.localizedDescription)")
            }
```

```
        }
    }
```

Výslednou metodu provádění klasifikace je nyní možné volat z jakékoli třídy v rámci aplikace.

4.4.4 Vyhodnocování fotografií

Integrování implementovaných funkcionalit strojového učení v rámci prezentačních vrstev vyvíjených pomocí SwiftUI je velmi jednoduché. Prezentační vrstva zobrazující uživateli možnost vyfocení či vybrání fotografie a její následné klasifikování byla vytvořena pomocí struktury `CameraPhoto`. Pomocí deklarování proměnné s vlastností `@EnvironmentObject` je možné jej přiřadit třídu `ImageClassificationModel` a tím získat možnost přístupu k jejím funkcím.

```
@EnvironmentObject var imageCModel : ImageClassificationModel
```

V souvislosti tím je nutné provést změnu v souboru `SceneDelegate.swift`, ve kterém je nutné definovat veškeré vytvořené takzvané environment objekty, aby je SwiftUI mohl zpřístupnit pro všechny využívané prezentační vrstvy. Implementace tohoto povolení je následující:

```
var imageClassificationModel = ImageClassificationModel()

window.rootViewController = UIHostingController(rootView:
contentView.environmentObject(imageClassificationModel))
```

Po tomto kroku je vše připraveno pro umožnění využívání logiky strojového učení ve vytvořené struktuře `CameraPhoto`. V rámci struktury byla vytvořena podmínka pro situaci, kdy není vyfocena ani vybrána fotografie uživatelem, je zobrazena struktura `PlaceholderView`, ve které jsou obsaženy základní pokyny pro zvolení fotografie. Pokud uživatel pořídil či vybral fotografii, je zobrazeno `CameraPhoto`, ve kterém je náhled dané fotografie spolu s tlačítkem s textem pro vyhodnocení.

```
self.imageCModel.image == nil ? PlaceholderView().toAnyView() : VStack {
    Image(uiImage: self.imageCModel.image!)
        .resizable()
        .aspectRatio(self.imageCModel.image!.size, contentMode: .fit)
```

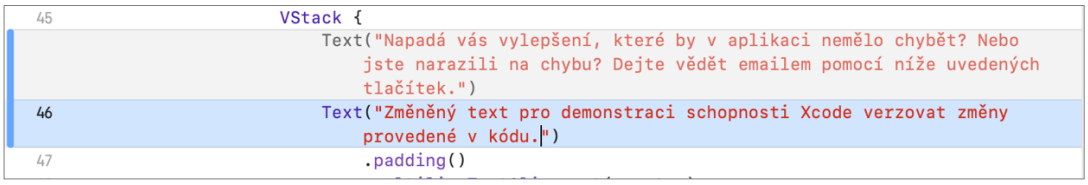
```
            .edgesIgnoringSafeArea(.bottom)
            .navigationBarTitle("Vyhodnocení fotografie psa", displayMode:
.inline)
            .padding()
        Text(self.imageCModel.classificationText.localizedCapitalized)
            .foregroundColor(.white)
            .font(.system(size: 22))
            .shadow(color: .black, radius: 1, x: 2, y: 2)
            .padding()
            .background(Rectangle()
                .foregroundColor(Color.init(.gray))
                .cornerRadius(20))
            .onTapGesture {
                 self.imageCModel.updateClassification()
}
```

Díky modifikátoru u textového prvku `.onTapGesture` je po kliknutí na tento prvek volána funkce `updateClassification()` ze třídy s logikou strojového učení. Po vyhodnocení fotografie je zobrazen výsledek dvou nejvíce podobajících se psích plemen a zároveň je zobrazen řádek s možností zobrazení základních informací o nejvíc podobném plemenu. Tento prvek byl implementován a zobrazen pomocí struktury `BreedRow`, která je využita také v zobrazení seznamu plemen.

4.4.5 Verzování zdrojového kódu

Psaní zdrojového kódu pro jakýkoli software je v dnešní době ve většině případů spojeno s jeho verzováním. Vytváření verzí kódu usnadňuje práce na projektech ve více osobách, kdy každá může pracovat na své části a po dokončení všech dílčích částí je možné je spojit do jedné finální.

Vývojové prostředí Xcode nabízí možnost zobrazování provedených změn od posledního commitu, tedy od poslední potvrzené editace a její uložení do repozitáře. Při zobrazení detailního zobrazení změn v kódu je vidět původní kód, označený šedou barvou, a napsaný nový kód, označený modrou barvou.



Obrázek 19 - Verzování kódu v Xcode [Zdroj: autor]

Celý zdrojový kód vyvinuté aplikace je nahrán na webové službě Bitbucket na URL https://bitbucket.org/davepet/diplomova-prace.

4.5 Komparace aplikace s alternativami na App Store

Implementovaná aplikace zobrazuje uživateli seznam plemen a pro každé z nich obrazovku se základními informacemi. Po zvolení možnosti vyfotografování či vybrání z uložených fotografií, aplikace algoritmus strojového učení klasifikuje psí plemeno obsažené na fotografii. V obchodě App Store je možné nalézt větší množství aplikací s tématikou rozpoznávání psích plemen jako jsou například Dogedex, Dog ID, Dog Scanner nebo Dog Book: Breed Identifier.

Aplikace Dog Scanner zobrazuje vyhodnocování a výsledky klasifikace psích plemen doplněné pěknými grafickými prvky. Zároveň její předností je velmi rozsáhlý data set pro trénování modelu. Obsahuje možnost rozpoznání 371 plemen a model strojového učení byl trénován na 415 773 fotografií, což je 20krát více podkladových dat, než bylo k dispozici pro účely této práce. Díky tomuto velkému počtu fotografií je možné model vytrénovat na vysokou validační přesnost a získat tím o mnoho lepší výsledky při klasifikování psích plemen. Dalšími funkcemi této aplikace je historie klasifikací, uživatelský profil s možností odemykání odznáčků například za klasifikované plemeno a je implementována nástěnka sloužící ke sdílení fotografií svých psů, kdy ostatní uživatelé mohou jednotlivé příspěvky komentovat.

V aplikaci Dog Breed ID je nutno vyzdvihnout rozšíření aplikace o funkcionalitu rozpoznávání psích plemen v reálném čase, kdy není nutné vybrat či pořídit fotografii, ale stačí pouze fotoaparát mobilního zařízení nasměrovat na psa a aplikace vyhodnotí 3 nejpodobnější plemena. Tato funkcionalita je založeno na frameworku ARKit, který zajišťuje integraci aplikace o rozšířenou realitu.

5 Výsledky a diskuse

5.1 Implementace

Proces vývoje mobilní aplikace s tématikou rozpoznávání psích plemen pomocí strojového učení se úzce držel nadefinovanou analýzou požadavků a následnými případy užití a scénáři z UI specifikace. Aplikační architektura aplikace byla zvolena MVVM z důvodu velmi dobré kompatibility s frameworkem SwiftUI, pomocí kterého byla celá aplikace implementována. Zdrojový kód aplikace byl z důvodu využití této architektury rozdělen na tři vrstvy, a to modelovou, prezentační a řídící vrstvu.

Díky SwiftUI byla implementace mobilní aplikace rychlá, přehledná a bez větší ztráty času při hledání chyb. Rozvržení částí aplikace do vrstev model, view a viewmodel napomohlo rychlejší opravě vyskytujících se chyb v kódu. Tento způsob implementace je novým způsobem programování aplikací (nejen) pro iOS, kdy jsou znovu používané již naprogramované třídy a tím je možné je vkládat do nových tříd bez nutnosti jejich opětovné implementace.

Jelikož je SwiftUI vydaný pro vývoj aplikací velmi krátkou dobu, stále se ještě potýká s nedostatky, které limitují jeho funkce a plnohodnotné nezávislé použití. I když stojí naproti frameworku UIKit co se například způsobu programování týče, stále je nutná jejich vzájemná integrace a kompatibilita. Ne všechny objekty z UIKit mají ekvivalent ve SwiftUI a je proto nutná jejich implementace a následné deklarování do struktur s kódem ze SwiftUI. Příkladem využití integrace těchto dvou frameworků je implementace funckionality pro strojové učení, respektive rozpoznávání objektů z fotografií, kde jsou v rámci třídy `ImageClassificationModel` deklarované proměnné, které jsou volány a zobrazovány ve struktuře napsané pomocí frameworku SwiftUI.

Aplikace byla vyvinuta dle zadání s funkcionalitami, chováním i vzhledem splňuje stanovené požadavky.

5.2 Uživatelské testování

Uživatelské testování bylo provedeno na dvou osobách, kterým byla aplikaci stáhnuta do mobilního telefonu v testovací/beta verzi. Cílem uživatelského testování bylo zjištění poznatků od reálných uživatelů aplikace, které by mohly podchytit problémy a nedostatky skrývající se v aplikaci. Hodnocena byla nejen funkčnost prvků zobrazených na

obrazovkách, ale také celkový takzvaný „user experience“, tedy uživatelský požitek, do něhož se řadí jak plynulé procházení aplikace, tak i její grafické zobrazení.

Chyby, které by zásadním způsobem ovlivňovaly funkcionalitu aplikace, nalezeny nebyly. V obou případech byla nalezena chyba při občasné špatné klasifikaci psího plemene, což bylo způsobeno validační přesností nasazeného modelu strojového učení. Tato chyba nebyla opraveno, jelikož je nejdříve nutné získat dostatečné množství podkladových dat, ze kterých bude možné vytvořit přesnější model.

Co se týče vizuální stránky obrazovek, na toto téma bylo vzneseno vícero doporučení. Na obrazovkách zobrazující jeden či více odstavců s textem by bylo vhodnější udělat text více čitelnější. Toto bylo opraveno pomocí přidání většího odsazení textu od stran a přidání více prostoru mezi jednotlivé řádky. Dalším podnětem ke zlepšení bylo přidání ikon a obrázků pro lepší uživatelskou přehlednost, kdy například u obrazovky s volbou kamery nebo knihovny pro zvolení fotografie ikony dostatečně nereprezentovaly druh vybrání fotografie.

Výstupy z uživatelského testování byly přínosné a bylo by na místě provést toto testování na skupině o více lidech. Už jen z důvodu poskytnutí více názorů na aplikaci a možnosti podchycení více příležitostí ke zlepšení.

5.3 Wireframes vs implementace

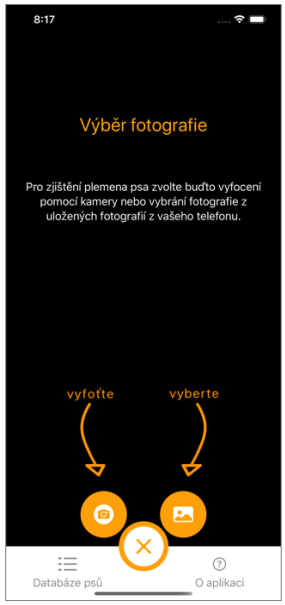
Díky dodržení zadání vyvinutá aplikace obsahovalo veškeré nadefinované funkce potřebné k jejímu bezchybnému chodu. Vývoj se držel dle zadaných případů užití a scénářů, ale po základní implementaci byly do jisté míry upraveny obrazovky k přehlednějšímu zobrazení, než bylo nadefinováno v drátěných modelech.

Odchylka od drátěných modelů je také při implementaci spodního navigačního panelu, kdy nebylo dodrženo zásady z Human Interface Guidelines, kde je stanoveno, že by tento prvek na obrazovce měl být formou systémového prvku Tab Bars, který umožňuje uživateli rychlý přesun mezi různými sekcemi v rámci aplikace. Implementovaným řešením bylo vložení vlastních tlačítek, které mají vlastnosti a chování jako to doporučované a dají se uzpůsobit. Tento přístup byl zvolen z důvodu zlepšení grafického zobrazení, při kterém je uživateli hlavní prvek, respektive funkcionalita, nejvíce na očích.

Do větší míry odlišné implementované obrazovky od drátěných modelů byla úvodní obrazovka, na které byl přidán i doporučený návod pomocí obrázků, kde na fotografii mít umístěného psa.

Obrázek 20 - Implementovaná úvodní obrazovka aplikace [Zdroj: autor]

Druhou obrazovkou byla volba kamery nebo výběr z knihovny fotografií, kde na základě uživatelského testování byly přidány grafické prvky usnadňující rychlejší orientaci uživatele pro následnou volbu klasifikované fotografie.



Obrázek 21 - Implementovaná obrazovka pro volbu fotografie [Zdroj: autor]

Poslední obrazovkou, u které došlo k mírnému odchýlení od drátěného modelu, byla obrazovka o aplikaci, kde k základním informacím o aplikaci přibyly také tlačítka pro zpětnou vazbu a nahlášení chyb. Je to prvek, který nebyl zahrnutý ani do analýzy požadavků, ale v aplikaci hraje důležitou roli, kdy dává uživatelům možnost vyjádřit své pocity z užívání aplikace či nahlásit chyby a nedostatky, které mohou být následně odstraněny a zlepší tak celkový uživatelský požitek.

Obrázek 22 - Implementovaná obrazovka O aplikaci [Zdroj: autor]

5.4 Přesnost ML modelu

Validační přesnost modelu strojového učení implementovaného v aplikaci je 66 %. Jinými slovy ze dvou třetin je aplikace schopna správně rozeznat psí plemeno na zvolené fotografii uživatelem. Tento jev je způsoben velmi podobnými plemeny, které odlišují jen určité specifické znaky, které nejsou na fotografiích tak snadno rozeznatelné a platí jen pro ně. Pro psí plemena s jasnými a unikátními rysy je model velmi přesný.

Pro zvýšení validační přesnosti modelu je třeba rozšířit podkladová data o více fotografií pro každé plemeno v řádu tisíců. Trénování modelu by následně ve svém procesu mělo dostatek fotografií, na kterých by se model naučil všem hlavním rysům plemene.

Z veřejně dostupných zdrojů je nejvíce zastoupený data set ze Stanford University, který je také využit pro aplikaci v rámci práce a z toho důvodu je obtížnější nalézt rozšiřující data. Ideálním řešením by byla implementace logiky ukládání vyfocených či vybraných fotografií uživatelem, kdy by potvrdil správný výsledek aktuálního vyhodnocení plemene

modelem. Tyto fotografie by bylo možné přidat k tréninkovým datům modelu a zvýšit tak jeho přesnost. Avšak toto řešení by mělo svá úskalí. Prvním z nich by byla povinnost informování uživatele o ukládání či přepoužívání fotografií, kterou by měl možnost přijmout či odmítnout. Druhou samotná svědomitost uživatelů při zaškrtávání správnosti výsledku, kdy by se mohlo stát, že uživatelem nepravdivě zaškrtnuté fotografie by zkreslovaly model a přinášely tak ještě nepřesnější výsledky.

5.5 Zhodnocení

Vývoj mobilní aplikace byl rychlý a efektivní s pomocí nejnovějších nástrojů a knihoven. Zdrojový kód aplikace je čitelný a rozdělený do jednotlivých vrstev dle zvolené aplikační architektury. Díky tomuto rozdělení se podařilo podchytit a opravit nalezené chyby. Výstupy z uživatelského testování pomohly zajistit zlepšení grafické stránky aplikace a pro koncového uživatele se tak stala přehlednější a snazší pro pochopení. Obrazovky a chování aplikace bylo dodrženo v souladu s UI specifikací a bylo tím dosaženo splnění stanovených podmínek.

Aplikace je schopna vyhodnocovat psí plemena ve většině případů korektně, ale stále by bylo vhodné zajistit přesnější model pro klasifikaci, pomocí kterého by byly uživateli zobrazovány věrohodnější výsledky hlavně u podobných plemenech.

6 Závěr

Diplomová práce se zabývala tématem vývoje mobilní aplikace pro mobilní zařízení s operačním systémem iOS. Proces vývoje mobilní aplikace bylo možné realizovat v praktické části, při které byla vytvořena funkční aplikace pro mobilní zařízení iPhone.

Prvním krokem praktické části bylo vytvořit analýzu požadavků zachycující nutné chování a vlastnosti aplikace. Na jejímž základě bylo možné sestavit UI specifikaci, ve které jsou definovány prvky a jejich chování na obrazovkách aplikace jak z pohledu uživatele, tak z pohledu systému. Toto výsledné chování obrazovky je následně zakomponováno do drátěných modelů, které vizuálně umožňují lepší představivost aplikace. Před zahájením implementace aplikace byl pomocí podkladových dat vytvořen model strojového učení, který byl implementován do kódu aplikace. Aplikační architekturou byla zvolena MVVM, která od sebe odděluje prezentační, datovou a řídící vrstvu a tím poskytuje mimo jiné snadnou přepoužitelnost již vytvořených kódů. Pomocí frameworku SwiftUI byl zajištěn vývoj deklarativním programováním a pomocí frameworků CoreML a Vision bylo dosaženo implementace strojového učení, v konkrétním případě aplikace rozpoznávání psího plemena na základě uživatelem zvolené fotografii.

Chování a vzhled aplikace se drží předem stanovených požadavků z analýzy požadavků a UI specifikace, kdy u malého počtu prvků bylo na základě výstupů z uživatelského testování provedeno několik dílčích změn pro lepší přehlednost. Úvodní obrazovka aplikace byla přizpůsobena snazšímu uživatelskému porozumění a byl přiložen návod pro vhodné vybírání fotografií k vyhodnocování. Obrazovky se seznamem plemen a základními informace a s volbou fotografie byly vyvinuty dle zadaných návrhů s přidáním drobných grafických prvků. Pro poslední obrazovku „O aplikaci“ byly kromě uvedení základních informací o obrazovce přidány také funkce pro zpětnou vazbu či nahlášení chyby.

I přes bezchybné grafické a funkční hledisko má aplikace prostor pro implementaci nových funckionalit, které by tak zvýšily její potenciál a zlepšily její konkurenční výhodu mezi aplikacemi se stejnou tématikou. Následující funkcionalitou, která by se v aplikaci mohla objevit, by bylo rozpoznávání psího plemena v reálném čase za pomoci rozšířené reality, kdy by uživatel nemusel pořizovat fotografii či vybírat z knihovny v telefonu.

Aplikace prošla schvalovacím procesem nutným pro publikaci v obchodě App Store a je možné ji vyhledat pod názvem DeDog.

7 Seznam použitých zdrojů

1. AKSHAY, Digital Tech. What Is The iOS Operating System? *Medium* [online]. 2020, 31. 12. 2019 [cit. 2020-02-14]. Dostupné z: https://medium.com/@digitaltechakshay/what-is-the-ios-operating-system-b19c5d19f5bc

2. Themes - iOS - Human Interface Guidelines - Apple Developer. *Apple Developer* [online]. Apple, ©2020 [cit. 2020-02-14]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

3. NAVEEN. What is the architecture of iOS. *Intellipaat.com* [online]. ©2011-2019, 10.9.2019 [cit. 2019-10-28]. Dostupné z: https://intellipaat.com/blog/tutorial/ios-tutorial/ios-architecture/

4. NAVEEN. Ios architecture. In: *IntelliPaat* [online]. ©2011-2019, 10.9.2019 [cit. 2019-10-28]. Dostupné z: https://intellipaat.com/wp-content/uploads/2015/12/ios-architecture.png

5. BOHON, Cory. Understanding multitasking on iOS: Should you quit apps? *TechRepublic* [online]. CBS Interactive, ©2019, 4.2.2014 [cit. 2019-10-28]. Dostupné z: https://www.techrepublic.com/article/understanding-multitasking-on-ios-should-you-quit-apps/

6. APPLE. UIKit. *Apple Developer Documentation* [online]. Apple, ©2019 [cit. 2019-10-30]. Dostupné z: https://developer.apple.com/documentation/uikit

7. APPLE. MapKit. *Apple Developer Documentation* [online]. Apple, ©2019 [cit. 2019-10-30]. Dostupné z: https://developer.apple.com/documentation/mapkit

8. APPLE. GameKit. *Apple Developer Documentation* [online]. Apple, ©2019 [cit. 2019-10-30]. Dostupné z: https://developer.apple.com/documentation/gamekit

9. ZAVŘEL, Roman. Apple končí s Game Center, v iOS 10 jej již nenajdete. *Letem světem Applem* [online]. Text Factory, ©2011-2020, 14. 6. 2016 [cit. 2019-10-30]. Dostupné z: https://www.letemsvetemapplem.eu/2016/06/14/apple-konci-s-game-center-v-ios-10-jej-jiz-nenajdete/

10. RAMNATH, Rajiv. Basics of the Media Layer for iOS App Development. *Dummies* [online]. ©2020 [cit. 2020-01-16]. Dostupné z: https://www.dummies.com/web-design-development/mobile-apps/basics-of-the-media-layer-for-ios-app-development/

11. Media Layer. *Developer Apple Documentation* [online]. 2015 [cit. 2020-01-16]. Dostupné z:

https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/MediaLayer/MediaLayer.html

12. Core Services Layer. *Apple Developer* [online]. Apple, 2015 [cit. 2020-02-16]. Dostupné z: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/CoreServicesLayer/CoreServicesLayer.html

13. BHARDWAJ, Sachin. Core OS Layer in iPhone. *C# Corner* [online]. ©2020, 14.3.2013 [cit. 2020-02-13]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/d49768/core-os-layer-in-iphone/

14. Core OS Layer. *Apple Developer* [online]. Apple, 2015 [cit. 2020-02-16]. Dostupné z: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/CoreOSLayer/CoreOSLayer.html

15. ZABŁOCKI, Krzysztof. Good iOS Application Architecture: MVVM vs. MVC vs. VIPER. *Realm* [online]. 2020, 8.5.2017 [cit. 2020-02-16]. Dostupné z: https://academy.realm.io/posts/krzysztof-zablocki-mDevCamp-ios-architecture-mvvm-mvc-viper/

16. ORLOV, Bohdan. Traditional MVC. In: *Medium* [online]. ©2020, 28.11.2015 [cit. 2020-02-16]. Dostupné z: https://miro.medium.com/max/868/1*E9A5fOrSr0yVmc7Kly5C6A.png

17. Understanding The Most Popular iOS Design Patterns in Swift. *IOS App Templates* [online]. iosapptemplates.com, 2016, 1.5.2019 [cit. 2020-02-16]. Dostupné z: https://www.iosapptemplates.com/blog/mobile-app-development/ios-design-patterns-swift

18. ORLOV, Bohdan. Cocoa MVC. In: *Medium* [online]. ©2020, 28.11.2015 [cit. 2020-02-16]. Dostupné z: https://miro.medium.com/max/870/1*c0aGaDNX41qu6e8E4OEgwQ.png

19. LACKO, Ľuboslav. *Vývoj aplikací pro iOS*. Brno: Computer Press, 2018. ISBN 978-80-251-4942-3.

20. ORLOV, Bohdan. Passive View variant of MVP. In: *Medium* [online]. ©2020, 28.11.2015 [cit. 2020-02-16]. Dostupné z: https://miro.medium.com/max/1400/1*hKUCPEHg6TDz6gtOlnFYwQ.png

21. ORLOV, Bohdan. IOS Architecture Patterns: Demystifying MVC, MVP, MVVM and VIPER. *Medium* [online]. ©2020, 28.11.2015 [cit. 2020-02-16]. Dostupné z: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52

22. ORLOV, Bohdan. MVVM. In: *Medium* [online]. ©2020, 28.11.2015 [cit. 2020-02-16]. Dostupné z: https://miro.medium.com/max/1398/1*uhPpTHYzTmHGrAZy8hiM7w.png

23. CHUNG, Eddy. The 7 Most Popular iOS Architecture Patterns Explained. *Zero To App Store* [online]. [cit. 2020-02-16]. Dostupné z: https://www.zerotoappstore.com/the-most-popular-ios-architecture-patterns-explained.html

24. ORLOV, Bohdan. VIPER. In: *Medium* [online]. ©2020, 28.11.2015 [cit. 2020-02-16]. Dostupné z: https://miro.medium.com/max/3196/1*0pN3BNTXfwKbf08lhwutag.png

25. GONZÁLEZ GARCÍA, Cristian, Jordán PASCUAL-ESPADA, Cristina PELAYO G-BUSTELO a Juan Manuel CUEVA-LOVELLE. Swift vs. Objective-C: A New Programming Language. *International Journal of Interactive Multimedia and Artificial Intelligence* [online]. 2015, 3(3), 74-81 [cit. 2020-02-18]. DOI: 10.9781/ijimai.2015.3310. ISSN 1989-1660. Dostupné z: http://www.ijimai.org/journal/node/776

26. Swift. *Apple Developer* [online]. Apple, ©2020 [cit. 2020-02-18]. Dostupné z: https://developer.apple.com/swift/

27. SwiftUI. *Apple Developer* [online]. Apple, ©2020 [cit. 2020-02-18]. Dostupné z: https://developer.apple.com/xcode/swiftui/

28. DE VRIES, Reinder. Get Started With SwiftUI: Apple's New UI Framework. *Adjust* [online]. 2020, 5. 2. 2020 [cit. 2020-02-18]. Dostupné z: https://www.adjust.com/blog/get-started-with-swiftui/

29. HUDSON, Paul. Training a model with Create ML. *Hacking with Swift* [online]. 19. 10. 2019 [cit. 2020-02-21]. Dostupné z: https://www.hackingwithswift.com/books/ios-swiftui/training-a-model-with-create-ml

30. GRAY, Andrew. Hands-On with the All-New Create ML App: Machine Learning for the Masses - Part 1. *CapTech* [online]. ©2020, 19. 6. 2019 [cit. 2020-02-21]. Dostupné z: https://captechconsulting.com/blogs/hands-on-with-the-all-new-create-ml-app-machine-learning-for-the-masses

31. Creating an Image Classifier Model. *Apple Developer* [online]. ©2020 [cit. 2020-02-21]. Dostupné z: https://developer.apple.com/documentation/createml/creating_an_image_classifier_model

32. Education - Teaching Code. *Apple* [online]. Apple, ©2020 [cit. 2020-02-21]. Dostupné z: https://www.apple.com/education/teaching-code/

33. Core ML. *Apple Developer* [online]. Apple, ©2020 [cit. 2020-02-21]. Dostupné z: https://developer.apple.com/documentation/coreml

34. HILLS, Dennis. Understand Core ML on iOS in 5 Minutes. *Medium* [online]. 23. 4. 2018 [cit. 2020-02-21]. Dostupné z: https://medium.com/@dmennis/understand-core-ml-on-ios-in-5-minutes-bc8ba5411a2d

35. Vision. *Apple Developer* [online]. Apple, ©2020 [cit. 2020-02-24]. Dostupné z: https://developer.apple.com/documentation/vision

36. MISHRA, Abhishek. *Machine Learning for iOS Developers*. Indianapolis: John Wiley, 2020. ISBN 978-1119602873.

37. PAVLÍČEK, Josef. *Interakce člověk a počítač* [online]. In: Moodle [online]. 2012 [cit. 2018-03-08]. Dostupné z: https://moodle.czu.cz/enrol/index.php?id=8916

38. Stanford Dogs Dataset. *Stanford Computer Vision Lab* [online]. ©2020 [cit. 2020-03-12]. Dostupné z: http://vision.stanford.edu/aditya86/ImageNetDogs/

8 Přílohy

Příloha č. 1: Zdrojový kód pro strukturu BreedDetail

```
struct BreedDetail: View {
    var breed : Breed

    var body: some View {
        GeometryReader { geometry in
            ScrollView {
                VStack {
                    self.breed.image
                        .resizable()
                        .aspectRatio(contentMode: .fit)
                        .frame(width: geometry.size.width, height: geometry.size.height/3)
                        .padding(1.0)
                    VStack(alignment: .leading) {
                        HStack(alignment: .top) {
                            Text(self.breed.breed)
                                .font(.subheadline)
                            Spacer()
                            Text(self.breed.originCountry)
                        }
                    }
                    .fixedSize(horizontal: false, vertical: true)
                    .padding()
                    VStack(alignment: .leading, spacing: 20) {
                        HStack(spacing: 29) {
                            Text("Velikost plemene:")
                            Text(self.breed.size)
                                .fontWeight(.semibold)
                        }
                        HStack(spacing: 129) {
                            Text("Srst:")
                            Text(self.breed.fur)
                                .fontWeight(.semibold)
                        }
                        HStack(spacing: 115) {
                            Text("Barva:")
                            Text(self.breed.color)
                                .fontWeight(.semibold)
                        }
                        HStack(spacing: 113) {
                            Text("Výška:")
                            Text(self.breed.height)
                                .fontWeight(.semibold)
                        }
                        HStack(spacing: 121) {
                            Text("Váha:")
                            Text(self.breed.weight)
                                .fontWeight(.semibold)
                        }
                        HStack {
                            Text("Průměrný věk dožití:")
                            Text(self.breed.avgAge)
                                .fontWeight(.semibold)
                        }
                    }
                    .padding()
                    .fixedSize(horizontal: false, vertical: true)
                    .font(.body)
                    Spacer()
                }
                Spacer()
            }
            .navigationBarTitle(Text(self.breed.breedCZ), displayMode: .inline)
        }
    }
}
```