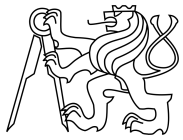
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Mobilní lexikon zvířat pro ZOO Praha
Student:	Minh Chu Anh
Vedoucí:	Ing. Josef Gattermayer
Studijní program:	Informatika
Studijní obor:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2017/18

Pokyny pro vypracování

Pražská ZOO uveřejnila v rámci portálu opendata.praha.eu množství zajímavých informací o zde žijících zvířatech. Cílem práce je vytvořit mobilní aplikaci pro telefony a tablety, která tato data atraktivní formou přiblíží návštěvníkům. Aplikace bude určena pro mobilní telefony s iOS.

1. Navrhněte vhodnou funkcionalitu pro mobilní aplikaci na základě dostupných dat. Návrh vytvořte ve formě wireframů a s vedoucím tento návrh dostatečně iterujte.
2. Proveďte průzkum trhu - jaké existují podobné aplikace, jaký mají business model, jaké mají počty stažení, odhadněte ziskovost.
3. Konzultujte s vedoucím práce grafické pojetí aplikace (grafiku dodá vedoucí).
4. Navrhněte, implementujte a otestujte mobilní aplikaci pro iOS.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 24. února 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Mobilní lexikon zvířat pro ZOO Praha

Chu Anh Minh

Vedoucí práce: Josef Gattermayer

8. ledna 2017

Poděkování

Děkuji svému vedoucímu bakalářské práce, že mi toto téma bakalářské práce nabídnul a byl v celou dobu vypracování k dispozici s pomocí. Také děkuji panu Ondřejovi Profantovi, správci portálu opendata.praha [1] za vyřešení vyskytlých se problémů na portále a za spolupráci s technickým vedením pražské zoo panem Jiřím Malinou, který dokázal opravit prvotní chybné datasety, vyskytující se na portále.

[1] opendata.praha – `www.opendata.praha.eu`

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. ledna 2017 .....

Odkaz na tuto práci

Chu Anh, Minh. *Mobilní lexikon zvířat pro ZOO Praha.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Práce se zabývá vývojem mobilní aplikace, běžícím na operačním systému iOS za pomocí portálu opendata.praha, platformy poskytované bezplatně od třetí strany. Ma za cíl atraktivní formou prezentovat zvířata, která se vyskytují v pražské zoo tamním návštěvníkům a následně jim pomoct se orientovat ve zdejší oblasti.

Klíčová slova iOS, Swift, zoo, Lexikon, otevřená data, interaktivní mapa, mobilní aplikace

Abstract

This works aims at the development of a mobile application running on the iOS operating system with the help of the portal opendata.praha, a free platform which was provided by a third party. The goal is to present living animals in the Prague zoo for its visitors in an attractive form and help them to orient themselves in the local area.

Keywords iOS, Swift, zoo, Lexicon, opendata, interactive map, mobile application

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešním moderním světě se bez mobilních zařízení jednoduše neobejdeme. Využíváme jej na denním pořádku a usnadňuje nám to život. K tomu abychom mohli dosáhnout plného využití těchto zařízení nepochybně patří i používání nejrůznějších aplikací dostupném na trhu. Naštěstí existují prostředky, které nám vývoj těchto aplikací práci velice zjednodušují. Firmám, dodávajícím nástroje na vývoj je jasné, jak jsou trhy s mobilními aplikacdemi důležité. Ostatně důkazem toho je, že za první kvartál roku 2016 firmě Apple[2] tvořili aplikace a služby s tímto spojené 20% z celkového obratu[1]. Na trhu v současné době dominují dvě platformy a to iOS a Android, na kterých běží většina mobilních zařízení.

Vzhledem k osobním zkušenostem, jak uživatelském tak i vývojářském, se v této práci zaměřím na iOS a popíšu, jak dosáhnout výsledku za pomocí současných trendů a principů ve vývoji.

Na téma bakalářské práce jsem narazil při procházením webových stránek Ackee[3], firmou založenou absolventy ČVUT, která se primárně zabývá vývojem mobilními aplikacemi. Pan Josef Gattermayer zde zveřejnil článek, kde nabízel několik témat závěrečných prací, většinou točících se kolem mobilního vývoje. Vybral jsem si téma „Mobilní lexikon zvířat“, která mě svojí zajímavostí nejvíce zaujala.

[2] Apple – `www.apple.com`
[3] Ackee – `www.ackee.cz`

Kapitola 1

Cíl práce

Pražská zoo v rámci webového portálu opendata.praha, na kterém se uchovávají data ve strojově čitelném formátu přístupných bez licenčních omezení, zveřejnila velké množství informací o tamních zvířatech.

Cílem této práce je s pomocí technickým vedením pražské zoo a s portálem opendata.praha vytvořit mobilní aplikaci Lexikon zvířat, která využívá tyto data a atraktivní formou jej přiblíží návštěvníkům všech věkových kategorií. Aplikace poběží na operačním systému iOS a bude využívat současné moderní trendy a prvky k vývoji.

K dosažení cíle se nepochybně vážou tyto úkoly:

- Průzkum trhu s podobným tématem
- Prostudování a otestování funkčnosti API portálu opendata
- Otestování datové sady, zveřejněné pražskou zoo
- Navrhnutí wireframes pro aplikaci a následná konzultace s vedoucím práce
- Vytvoření iOS aplikace, včetně kompletního designu

Kapitola 2

Analýza

V této části se pokusím rozebrat danou problematiku, zanalyzovat vstupy, které mám k dispozici a najít nejlepší řešení s využitím těch nejvhodnější technologií. Než se pustím do analýzy, pokusím se vysvětlit, na jakém principu fungují otevřená data.

2.1 Opendata

Koncept opendata jsou podle definice informace a data zveřejněná na interneta, která jsou úplná, strojově čitelná, používající standardy s volně dostupnou specifikací, zpřístupněná za jasně definovaných podmínek užití dat s minimem omezení a snadno dostupná uživatelům při vynaložení minima možných nákladů.[2]

Jako první se systematickým zveřejňováním dat v otevřené podobě začala zabývat vláda USA. Katalog založila v květnu roku 2009. Ten záhy získal podporu prezidenta Obamy. v prosinci téhož roku vydal prezident direktivu, která nařizovala státním institucím a agenturám zveřejňování datových sad prostřednictvím portálů.[3]

V České republice otevřená data můžeme nalézt pro větší města nebo kraje nebo i od ministerstva vnitra a ministerstva financí. Právě jeden z těchto portálů, několikrát výše zmiňovaném, budeme v aplikaci využívat.

2.2 Analýza portálu a dostupných datasetů

Dříve, než se začnu zabývat funkčností aplikace, zanalyzuju a otestuju funkčnost serveru a korektnost dat, které budu využívat napříč aplikace. Data stahována z portálu jsou totiž základním stavebním prvkem aplikace a aplikace bez nich nebude správně fungovat.

Opendata.praha

Nejdříve jsem udělal důkladný průzkum portálu opendata.praha. Zjistil jsem, že se na portálu se vyskytují data ve formátu XLSX[4] a CSV[5]. Po dalším zkoumání jsem zjistil, že na portálu je implementované webové REST API[6], které umí zpracovávat restové požadavky. Pomocí této technologie není totiž potřeba stahovat datasety manuálně, ale místo toho se z aplikace pošle požadavek na server, který požadavek umí zpracovat a následně poslat data zpátky v JSON[7] formátu.

Datasety

Při bližším zkoumáním těchto datasetů od pražské zoo jsem bohužel narazil na velké nedostatky a bez jejich opravení jsem nemohl nadále pokračovat ve své práci. v datasetech se totiž vyskytovaly duplicitní data, která byly mezi sebou špatně provázaná. Data byly i nekompletní, což dokazovalo to, že by se v celém zoo vyskytovalo pět druhů zvířat.

Po konzultaci s vedoucím práce jsme došli k závěru, že data v ten moment byla nepoužitelná a bylo nutné je opravit. S vedoucím jsme tedy naplánovali schůzku s vedením portálu opendata.praha Ondřejem Profantem a s technickým vedením pražské zoo Jiřím malinou. Zde jsme probrali vyskytlé chyby, ktere vedení zoo s pomocí Ondřeje opraví a navíc poskytnou další data, které jsem navrhl. Navrhl jsem totiž, aby se v datech nacházel i odkaz na obrázky zvířat, které byly v té době k dispozici na webu pražské zoo.

Po kratší pauze se data skutečně dala Do pořádku a já jsem měl všechny prostředky k tomu abych začal vývoj.

2.3 Požadavky na aplikaci

Výsledkem analýzy jsou tyto funkční a nefunkční požadavky aplikace:

Funkční požadavky

- Synchronizace dat z portálu opendata.praha
- Ukládání dat pro offlinové využití
- Zobrazení seznamu všech zvířat
- Řazení seznamu zvířat podle abecedy
- Hledání zvířete podle jména

[4]XLSX – souborový formát, kde se ukládají textové, tabulkové dokumenty
[5]CSV – souborový formát, ve kterých jsou jednotlivé položky oddelené čárkou
[6]Representational State Transfer – architektura pro distribuované prostředí[4]
[7]JSON – reprezentace dat, který je člověkem i strojem čitelná a snadno generovatelná

- Zobrazení detailních informací u jednotlivých zvířat (potrava, latinský název, chov, rozmnožování a další zajímavosti)
- Stahování obrázků z webových stránek zoopraha.cz pomocí odkazů, získané z dat
- Rozdělení jednotlivých zvířat podle potravy, třídy a lokality
- Přehled míst na mapě se zvířaty v okolí Pražské zoo

Nefunkční požadavky

- Aplikace poběží na iOS 9.0 a novější
- Aplikace bude napsána v jazyce Swift
- Aplikace bude mít přehlednou navigaci s plynulým uživatelským rozhraním
- Aplikace poběží v offline režimu

Kapitola 3

Průzkum trhu

Z výše uvedených požadavků nejdříve prozkoumám trh a zjisím, jak se s tím vypořádali předchozí vývojáři, jaký mají business model, zda už takové řešení existuje, případně jak se od nich budu lišit a zda-li vymyslím lepší řešení.

3.1 Monetizace aplikace

Předtím než se pustím Do odhadu ziskovostí aplikací, nejdříve popíšu existující modely monetizace. v současné době existuje několik metod, jak můžeme aplikaci monetizovat.

Placené aplikace

Za každé unikátní stažení aplikace můžeme obdržet určitou částku peněz, kterou si určíme ještě před vydáním aplikace do appstoru. Průměrné částky za jednotlivé stažení se pohybují od jednoho až po pěti dolarů.

In-App purchases

Princip tohoto způsobu je vcelku jednoduchý. Aplikace jsou většinou volně ke stažení, uživatelům nabízejí omezenou funkčnost a za příplatek se jim další funkce rozšíří. Zde se může jednat buď o jednorázový příplatek, nebo o měsíční, až ročné předplatné.

Reklamy v aplikaci

Tvůrcům aplikací se v případě velkého počtu jejího stažení a užívání vyplatí vydat aplikaci jako bezplatnou a těžit za každé zobrazení reklamy, obsažené v aplikaci. Případně za poplatek tvůrcům se z aplikace reklamy smažou.

Propagace nebo rozšíření businessu

Samozřejmě nesmíme zapomenout na ty aplikace, které mají za účel propagovat už rozjetý business. Tyto aplikace sami o sobě nevydělavájí, ale svým způsobem podporují, propagují nebo rozšiřují už nějakou fungující službu. Zde bych jako příklad uvedl aplikace od telefonních operátorů, nebo od různých bank.

3.2 Odhad ziskovosti

Níže provedu průzkum několik aplikací, prozkoumám jejich business model a odhadnu ziskovost těchto řešení. Těchto výsledků jsem dosáhnul tím, že jsem si nejprve procházel AppStore[8] a snažil jsem se vyhledat aplikace na základě několika různých klíčových slov. Pokud jen použiju informace z appstoru, nezjistil bych žádné statistiky o těchto aplikacích. Narozdíl od appstore, Google Apps[9] poskytuje informace o rozmezí počtu stažení. Pokud aplikace má verze pro obě platformy a známe přibližný poměr uživatelů iOS a Android v České republice, který je 5:1[5], můžeme se od této důležité informace odrazit. Můj odhad životnosti iOS projektů je čtyři roky, zde jsem nedokázal najít žádné výzkumy, které se touto problematikou zabývaly, proto nedokážu garantovat svoji přesnost odhadu.

3.3 Údolí slonů

Na klíčové slovo *zoo praha*, mě kupodivu nenavedlo appstore na žádnou aplikaci tohoto jména, ale na aplikaci jménem Údolí slonů. Už z názvu je jasné, že se celá aplikace zaměřuje pouze na jeden typ zvířete, což je velké omezení.

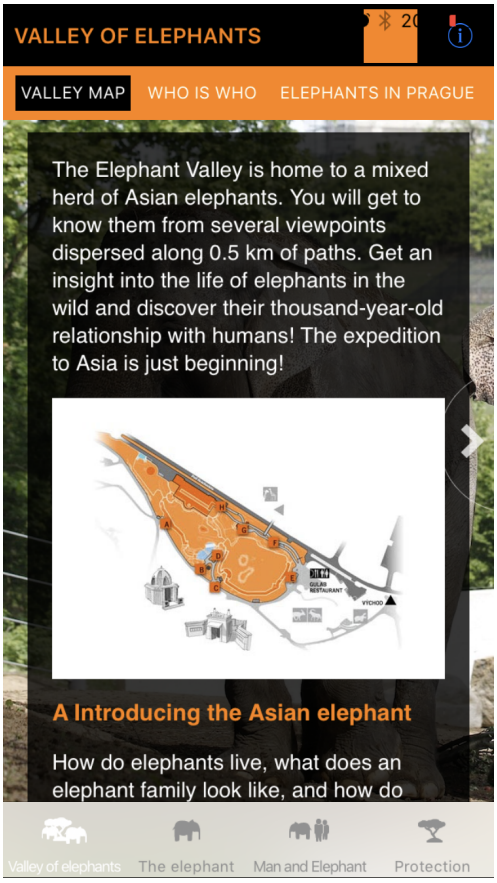
Uživatelské rozhraní

Při prvním spuštěním aplikace mě neminulo chybné zobrazování některých uživatelských prvků. Status bar je překryt plovoucím tlačítkem v aplikaci a některé tlačítka žádnou akci nevyvolávala . Jako pozitivum tu vidím četnost různých obrázků a zajímavé informace o ochraně zvířat.

Business model

Po delším zkoumání a procházením si internetových stránek pražské zoo jsem zjistil, že právě tato aplikace byla vydána samotnou zoo. Můžem se zde tedy zabývat, jak si tato aplikace vede, jako propagační materiál. Zjistil jsem, že aplikace má i svou verzi na platformě Android, kde má 1000 až 5000 stažení. Z výše uvedeného poměru můžem tak odhadnout, že iOS verze má přibližně

[8] Trh s mobilními aplikacemi pro platformu iOS
[9] Trh s mobilními aplikacemi pro platformu Android



Obrázek 3.1: Vzorový obrázek z AppStore

200 až 1000 počet stažení. Jak dále bychom mohli odhadnout ziskovost? Mým dalším hrubým odhadem, by zde byl, že aplikace by mohla oslovit 25% uživatelů, co si aplikaci mohli stáhnout a ročně by navštěvovali zoologickou zahradu. S tímto odhadem jsem se inspiroval aplikací Spotify, který podle všeho má až 25% platících uživatelů.

Pokud s touto informací budu nadále počítat, tak roční zisk by činil maximálně 50 000Kč, pokud beru v potaz cenu vstupného 200Kč. Po důkladném procházením aplikace a na základě mých zkušeností jako vývojář, jsem odhadl projekt na 80 člověkohodin, z čehož jsem vyvodil, že náklady za aplikaci by se mohly vyšplhat na 80 000Kč. V tabulce 3.3 započítávám i roční diskont 6%. Z tabulky můžeme vidět, že zhruba třetím rokem se dostáváme do zisku.

	0 rok	1 rok	2 rok	3 rok	4 rok
Vývoj aplikace	-80 000	0	0	0	0
Poplatky AppStore	0	-2 000	-2 000	-2 000	-2 000
Technická podpora	0	-1 0000	-1 0000	-10 000	-10 000
Zisk za lístky	0	50 000	50 000	50 000	50 000
Cash flow	-80 000	38 000	38 000	38 000	38 000
Discounted cash flow	-80 000	35 720	33 440	31 160	28 880

Tabulka 3.1: Odhad cash flow Údolí slonů za čtyřleté období

Celkový čistý zisk	49 200 Kč
Návratnost investice	1.62

Tabulka 3.2: Odhad celkového zisku Údolí slonů

3.4 Pavilon želv

Na stejném principu funguje i aplikace Pavilon želv. Dá se říct, že aplikace je téměř totožná až na mediální obsah. Je tu stejná struktura aplikace a vyskytují se zde i stejné elementy. Také jsem zde nalezl stejné chyby jako v předchozí aplikaci. Avšak počet stažení je zde znatelně méně. Na android app store se pohybuje počet stažení mezi 100 - 500. Pokud se zde použil stejný zdrojový kód jako v předchozí aplikaci, což předpokladám že udělali, tak bych tu odhadl náklad za výměnu obsahu na 10 000 Kč. Pokud použiju stejný princip odhadu, vyšlo by, že hned první rokem se dostává aplikace do zisku.

3.5 Zoo Liberec

Další aplikace, na kterou jsem narazil, je Zoo Liberec. Aplikace byla vydána samotnou libereckou zoologickou zahradou. Tato aplikace mě zaujala mnohem více než ty předchozí. Zde jsem měl přehledný navigační systém, který je vyřešen pomocí tabBaru[10]. Aplikace určitě splnila svůj účel, a to informovat návštěvníky o aktualním dění v zoo pomocí přehledné tabulky událostí a programů. Obsahuje interaktivní mapu zoo, která pomáhá navštěvníkovi v orientaci a vypisuje důležité body, jako jsou parkovací místa a veřejné WC. Podle mého názoru aplikace má poněkud nešťastnou kombinaci barevných prvků. Kombinace černé, oranžové a bílé barvy se mi zdá dost chudé a aplikace působí velmi zastarale. Pro aplikaci, které má téma o přírodě bych vybral kombinaci veselejších barev. Také se zde využívá minimum obrázků a grafických prvků, což nepřidává na aktraktivitě aplikace.

[10]UITabBarController - komponenta využívaná pro navigaci v aplikaci, navigační lišta je zpravidla ve spodní části obrazovky a obsahuje klikatelná tlačítka, pomocí kterých se naviguje mezi obrazovky[6]

Obrázek 3.2: Pavilon želv

Překvapilo mě, že jen 500 - 1000 uživatelů si aplikaci stáhlo na platformě Android. Aktuální vstupné zde činí 90Kč. Pro vyšší komplexnost aplikace je můj odhad 200 člověkohodin. Pokud je můj odhad životnosti iOS aplikací čtyři roky, investice by zde byla ztrátová.

3.6 Shrnutí

Z výše uvedéného průzkumu jsem zjistil několik věcí. Určitě se vyvarovat chyb, aby aplikace fungovala i na budoucích zařízení. Budu klást důraz na optimalizaci uživatelského zařízení pro různé velikosti displayů na zařízeních. Toho dosáhnu využitím nativních komponent, které nám poskytuje Apple. Blíže se tomu budu věnovat v kapitole o uživatelském rozhraní aplikace. Důležité je také mít přehlednou navigaci, aby se v ní mohli orientovat uživatelé všech věkových kategorií.

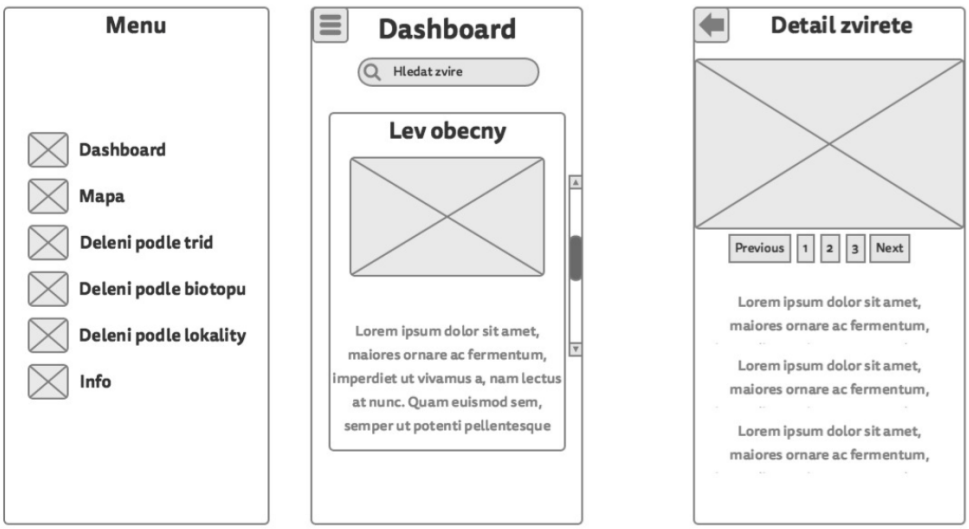
Obrázek 3.3: Aplikace Zoo Liberec

Kapitola 4

Návrh aplikace

4.1 Wireframe aplikace

Po detailním průzkumu uvedených aplikací jsem navrhl wireframy[11] aplikace a po sléze jsem je předal vedoucímu bakalářské práce pro zkontrolování. Pro náčrt wireframů jsem si vybral nástroj mockingbird, který je bezplatný k užití.



Obrázek 4.1: Wireframe aplikace pomocí nástroje Mockingbird

[11] wireframe – drátěný náčrt aplikace, který se využívá pro definování funkcionalit aplikace

4.2 Výběr programovacího jazyka

Na výběr máme jazyky Objective-C a Swift, které jsou poskytovány a spravovány přímo společností Apple. Objective-C je již starším jazykem, užívaným k vývoji aplikací na iOS a OS X platformě. Vychází ze staršího jazyka Smalltalk a už tu je s námi přes třicet let[7]. Je pořád hojně využíván, protože řada firem má aplikaci napsanou v Objective-C a přechod na novější jazyk Swift by byl velice nákladný.

Swift je nástupcem Objective-C, říká se o něm, že je to „Objective-C bez C“. Poprvé byl představen roku 2014 na konferenci Applu. Z hlediska syntaxe je Swift jazyku Objective-C velmi podobný, ale na rozdíl od Objective-C nevyužívá ukazatele na proměnné[8]. Swift je velice mladý jazyk, který se neustále vyvíjí, důkazem toho je, že každým rokem se vydává novější verze jazyka. Současná verze je 3.0, která už je velice stabilní.

Pro tuto práci bude použit jazyk Swift. Jedním z důvodů výběru tohoto jazyka je i moje pracovní zkušenost. Swift je v současné době vývojáři preferovaným jazykem a proto existuje mnoho nezávislých knihoven, které tak usnadňují vývoj aplikace.

4.3 Výběr vývojového prostředí

Pro prostředí vývoje jsem si vybral Xcode. Stejně jako většina vývojářů iOS je Xcode preferované prostředí. Je to software vydaným, spravovaným Applem a obsahuje veškeré nástroje potřebné od napsání prvního řádku kódu, až po vydání Do AppStore.

Existuje však i jiná alternativa k Xcode a tou je AppCode[12]. Výhody AppCode oproti Xcode jsou četnosti vlastních přizpůsobení. AppCode například umí i generovat jednotlivé řádky kódu, které si nadefinujeme[9].

Nevýhodou AppCode je závislost na Xcodu, pro použití AppCode pořád potřebujeme mít nainstalovaný Xcode. Další nevýhodou je také stabilita programu. Mnohým vývojářům se vyskytly problémy při vývoji, kde jim program padal, nebo nefungovaly vstupy od klávesnice[10]. AppCode také nemá podporu pro Storyboarding[13] a GUI editoru. Storyboarding je jeden z několika způsobů jak vytvářet uživatelské rozhraní. Dále o nich budu popisovat v kapitole o uživatelském rozhraní a proč jsem se rozhodl je používat.

4.4 Výběr architektury aplikace

V současné době se pro vývoj mobilních aplikací využívají typů architektur. Měli bychom vybírat danou architekturu na základě zkušeností, struktuře a

[12] AppCode - vývojové prostředí iOS aplikací od firmy JetBrains

[13] Storyboarding - vytváření uživatelského rozhraní pomocí grafického editoru ve vývojovém prostředí

funkčnosti aplikace.

MVC - Model View Controller, nebo Massive ViewController?

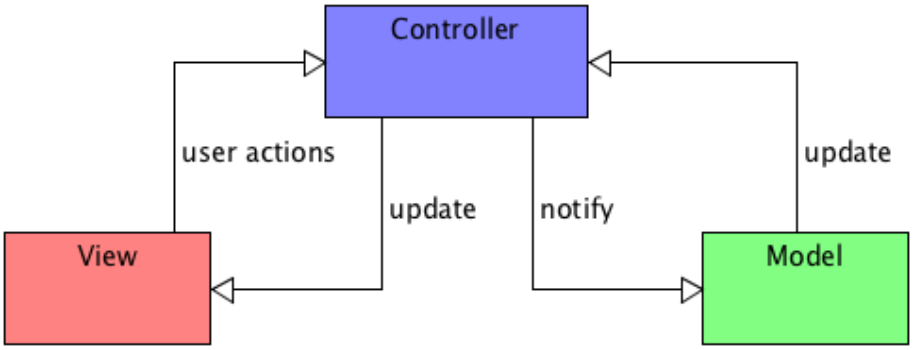
Je všeobecné známá a je to vůbec nejvíce užívaná architektura. MVC architektura se všeobecně rozděluje Do tří vrstev[11].

Model Reprezentuje data a business logiku aplikace.

View Zobrazuje uživatelské rozhraní .

Controller Má na starosti tok událostí v aplikaci a obecně aplikační logiku.

V iOS vývoji se controller nazývá ViewController. Protože se také stará o uživatelské rozhraní. Nejen, že se stará o logiku, ale třeba i přeposílává data z modelové vrstvy do svých views. Vývojáři controlleru přivlastňují jméno Massive ViewController, protože často se zde objevují funkce týkajících se síťových požadavků, nebo i zpracovávání dat. Objevují se zde funkce, která by neměly být součástí controlleru, ale podle definice by neměly patřit ani do jiných vrstev. ViewController se tak stává postupně hůře a hůře čitelným.[12]

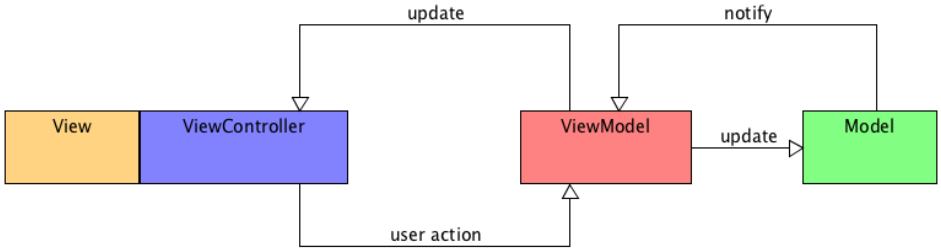


Obrázek 4.2: Model View Controller

MVVM - Model view view model

Kvůli výše uvedenému problému Massive View Controlleru vývojáři vymysleli další řešení. MVVM narozdíl od MVC používá místo ViewControlleru ViewModel. Tato vrstva přímo komunikuje s modelovou vrstvou, stará se například o transformaci dat a uchovává si jejich stav. Následně pak tyto data předá svému view. Protože se každý ViewModel váže jen ke jednomu konkrétnímu View, je její funkce omezená a je ma jen jeden konkrétní úkol. Aplikace

je tak rozdělena do menších logických modulů a díky tomu můžeme moduly opětovně používat podle pravidla DIY[14]



Obrázek 4.3: Model View ViewModel

VIPER

VIPER vznikla jako vylepšená verze MVVM. Přibyli nám tu navíc tyto vrstvy:

View Zobrazuje to, co vidí uživatel.

Interactor Stará se o business logiku.

Presenter Stará o přípravu obsahu, který obdržel od Interactoru. Také se stará o vstupy od uživatele.

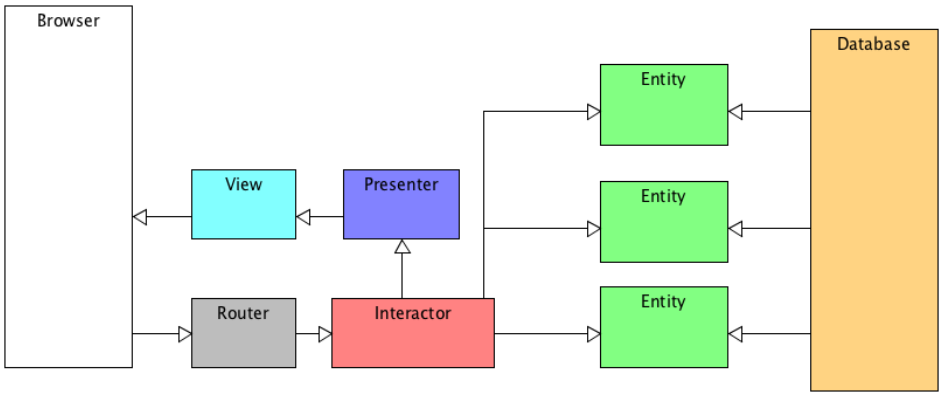
Entity Obsahuje datový model

Routing Má na starosti navigační logiku aplikace. Například se stará, která obrazovka se má objevit uživateli.Svým členěním umožňuje tak pohodlnější paralelní programovaní, kde není problém pracovat na projektu i ve vícečlenném týmu. Pro jednotlivé moduly můžeme také psát nezavislé testy. Právě i kvůli tomuto důvodu je tato architektura mnohem vhodnější pro unit testing[13].

Použitá architektura

Pro menší aplikace jsou MVVM a VIPER zbytečně složité. I pro ty nejjednodušší aplikace je vyžadováno vytvoření mnoho tříd, modulů a boilerplate[15] kódů. I ta nejmenší změna funkcionality v aplikaci vzžaduje přpisování mnoho kódu s tím spojených, a proto je tato architektura spíše vhodná pro aplikace, které mají přesně definovanou strkturu a funkcionalitu.

[14]Don't repeat yoursself
[15]Opakující se kód s cílem získání nějakého výsledku



Obrázek 4.4: VIPER

Narozdíl od přímočarého MVC se zde kód dělí i Do několika abstraktních úrovní a svojí komplexností pro nové vývojáře tak bývá těžce pochopitelná. Ostatně nejsou tyto dvě architektury tak často používány. v praxi jsou využívány jen těmi zkušenějšími vývojáři, co rádi experimentují s novými technologiemi. Nutno podotknout, že MVVM používají vývojáři v Ackee jako standartní archtitekturu.

Vzhledem k osobním zkušenostem a k vlastnosti aplikace jsem si vybral MVC architekturu, která se podle mě jeví jako nejvhodnější. Jak už jsem výše zmínil, MVC je základní architektura přímo doporučovaná Applem. Stejně tak je i díky svojí přímočarostí čtenářům blíže k pochopení. Také dokážu nevýhody architektury MVC, jako je Massive View Controller vhodné vykompenzovat širokou škálou vlastnosti moderního jazyka Swift, jakými jsou například tzv. protokoly a extensiony. Více se tím budu zabývat přímo v realizaci.

Kapitola 5

Realizace

Dříve než se můžeme ponořit Ddo vývoje aplikace musíme se napřed vypořádat s konfigurací projektu.

5.1 Počáteční konfigurace projektu

V dnešní době, není žádnou hanbou využívat knihovny třetích stran. Podle mého názoru, není důvod proč bychom měli vymýšlet nové řešení komponent, které už bylo vyřešené. Proto budu v aplikaci používat knihovny, které mi vývoj mnohonásobně zrychlí. Seznam knihovny můžeme nalézt na konci této kapitoly. Pokud bychom ale šli tradiční cestou a chtěli si nainstalovat frameworky ručně, nebylo by ta zcela jednoduché. Museli bychom si stáhnout knihovnu, rozbalit archív, celý adresář zkopírovat do projektu a následně v Xcode projektu nalinkovat binární soubor knihovny v sekci Link With Libraries. Tento proces bychom museli zopakovat pro všechny knihovny. Navíc neexistuje cesta pro aktualizaci knihoven na budoucí verze. K tomu byl vyvinut nástroj Cocoapods.

Cocoapods

Cocoapods je nástroj, který se stará o management externích knihoven třetích stran. Pomocí něho můžeme přidávat, mazat a aktualizovat knihovny bez větší namáhy. Nainstalujeme si nástroj pomocí příkazu gems cocoapods v terminálu a následně v projektu provedeme příkaz pod init. Tímto příkazem se nám vytvořil v adresáři soubor Podfile, do něhož si nadefinujeme potřebné knihovny. Po té stačí spustit příkaz pod install a Cocoapods se o všechny záležitosti postará. Po té už máme celý projekt nakonfigurovaný a můžeme se pustit do vývoje.

5.2 Extensiony v aplikaci

Extension ve Swiftu znamená přídavek ke třídě. Swift nám totiž dovoluje jakékoliv třídě definovat rozšiřující funkci, můžeme ji definovat kdekoliv v aplikaci a kompilátor nám už vyřeší jak tyto funkce propojit. Je to velmi pohodlné a efektivní. v aplikaci jsem například napsal rožšířující funkci defaultnímu controlleru, aby uměl zobrazovat chybovou hlášku. Můžu pak tuto metodu vyvolat v jakémkoliv controlleru, jako kdyby byla funkce defaultně implementována.

```
extension UIViewController {
    func showRequestTimeOut() {
        self.showAlertMessage("Error",
            message: "Request timed out")
    }

    func showAlertMessage(_ title: String, message: String) {
        let alertCtrl = UIAlertController(title: title,
            message: message,
           preferredStyle: .alert)

        let okAction  = UIAlertAction(title: "OK",
           style: .default,
           handler: nil)

        alertCtrl.addAction(okAction)
        self.present(alertCtrl, animated: true,
         completion: { _ in })
    }
}
```

5.3 Modelová vrstva v aplikaci

Persistence dat

Z předešlé analýzi víme, že aplikace bude fungovat v offline režimu. Abych toho docílil, potřeboval jsem zajistit perzistenci dat. Pro persistenci dat v iOS aplikaci existuje několik řešení.

SQLite Prvním řešením je použití vlastního souboru sqlite. iOS systém totiž umožňuje tento soubor uchovávat v aplikaci a následně z ní číst pomocí sqlite Cčkových funkcí[16]. Zásadní nevýhodou tohoto řešení je pracnost. Museli bychom si ručně napsat kód, který by se staral o čtení jednotlivých rádků, sloupců v souboru a následně je zpracovat a napasovat

[16] funkce napsané v jazyce C

do patřičných objektů. Také by bylo nutné se zde starat o vícevláknové čtení ze souboru.

Coredata Stará se o business logiku. Druhým řešením je použití CoreData přímo od Applu. CoreData je knihovna starající se o modelovou vrstvu objektů v aplikaci. Poskytuje generalizované a automatizované řešení úkolů spojených s životním cyklem objektů, včetně jejich perzistence[14]. Nevýhodou CoreData, je složitá konfigurace a psaní boilerplate kódů.

Realm Třetím a také použítým řešením v aplikaci je Realm. Realm byl navržen jako rychlejší a efektivnější alternivou SQLite a Coredata[15]. Výhodou je, že toto řešení je také cross-platformní a je dostupné jak pro web, tak i pro iOS a Android.

Definovaní struktury objektů v modelové vrstvě

Zde jsem si nadefinoval struktury objektů, které v databázi budou uložené. Každý objekt, který se bude ukládat do databáze, dědí od třídy Object. Třída Object je zde dodána od knihovny Realm a tímto děděním dávám Realmu vědět, že se o objekt bude starat.

```
class Animal: Object {
    dynamic var id: Int = 0
    dynamic var name: String = ""
    dynamic var animalClass: Int = 0
    dynamic var latinName: String = ""
    dynamic var order : Int = 0
    dynamic var spread: String = ""
    dynamic var biotopNote: String = ""
    dynamic var foodNote: String  = ""
    dynamic var attractions: String = ""
    dynamic var animalDescription: String = ""
    dynamic var breeding: String = ""
    dynamic var reproduction: String = ""

    override class func primaryKey() -> String? {
        return "id"
    }
}
```

Příkladem je tu třída Animal. Animal nese informace konkrétního zvířete, jako je název, odkaz obrázku, informace o potravě, výskytu, rozmnožování a atd. Narozdíl od normalní proměnné deklarovane ve projektu se proměnné v těchto objektech deklarují pomocí předložky dynamic var. Zde je duležitá obezřetnost při deklarování typu, pokud se totiž deklaruje typ, který není Realmem podporovaný, aplikace v momentě práce s databazí spadne.

Také je důležité zmínit, že pokud plánuju objekty v databázi také aktualizovat, musím objektu určit primární klíč. V Realmu se primarní klíč definuje přepsáním statické funkce primaryKey, jako návratovou hodnotu se vrátí jméno proměnné ve formě řetezce. Každý objekt implementuje funkci new, která zajišťuje serializaci JSON Do objektu.

```
static func new (json: JSON) -> FoodRelation {
        let relation = FoodRelation()
        relation.foodID = json["id_f"].intValue
        relation.animalId = json["id"].intValue

        return relation
}
```

Komunikace se serverem opendata.praha

Jak už jsem předtím uvedl, vývojáři mají tendenci veškerou práci se serverem dávat do Controlleru a počet řádek kódu se tak zvyšuje a zvyšuje. Toho jsem se chtěl vyvarovat a proto jsem pro práci se serverem vytvořil tyto třídy, které jsou s tímto spojené.

APIRequest Třída obalující jeden požadavek na server.

APIQueue Má v aplikaci funkci fronty, v níž se řadí jednotlivé APIRequesty.

ApiCommunicator Chová se jako samotný singleton[17] a má za úkol posílat a zpracovávat požadavky ze serveru.

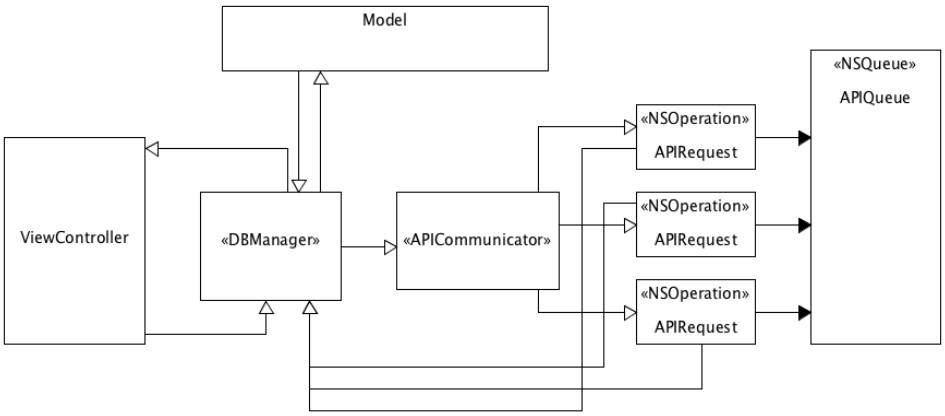
DBManager Má za úkol management objektů v databázi. Každý controller, kter pracuje s daty má právě jednu instanci DBManageru.

Životní cyklus jednoho požadavku

V momentě, kdy chci zaktualizovat data v daném controlleru, pošlu příkaz instanci DBManager o aktualizaci databáze. DBManager po té zažádá třídu APICommunicator o vytvoření APIRequestu a posléze jej zařadí do fronty APIQueue. Jakmile dostává DBManager zpětnou vazbu, aktualizuje data ve své databázi o dá svému controlleru vědět o své změně.

Důvodů, proč jsem navrhnul tuto možná na první pohled složitou strukturu, je několik. Celá operace probíhá asynchronně a tak neblokuje uživatelské rozhraní. Dále abych zabránil vytížení serveru mnohonásobnými požadavky, zde ty požadavky serializuji a postupně je posílám až v momentě, kdy server zpracuje ty předchozí.

[17] singleton - Statická instance



Obrázek 5.1: Životní cycklus webového požadavku

5.4 View vrstva v aplikaci

Xcode GUI nebo kód?

Na toto téma se vždy vytvoří dva tábory iOS vývojářů a každá strana si hájí svůj názor. Vytváření užívatelského rozhraní aplikace v kódu je výhodný tím, že se lépe využívá k opětovnému využití. Také víme, kde přesně jsme definovali nějaký element. Avšak s nástupem nových mobilních zařízení a výskyt nových různých velikostí displejů se musí kód rozětvit pro různé příklady displejů. Apple proto přišel s autolayoutem a její podpory v Xcode GUI.

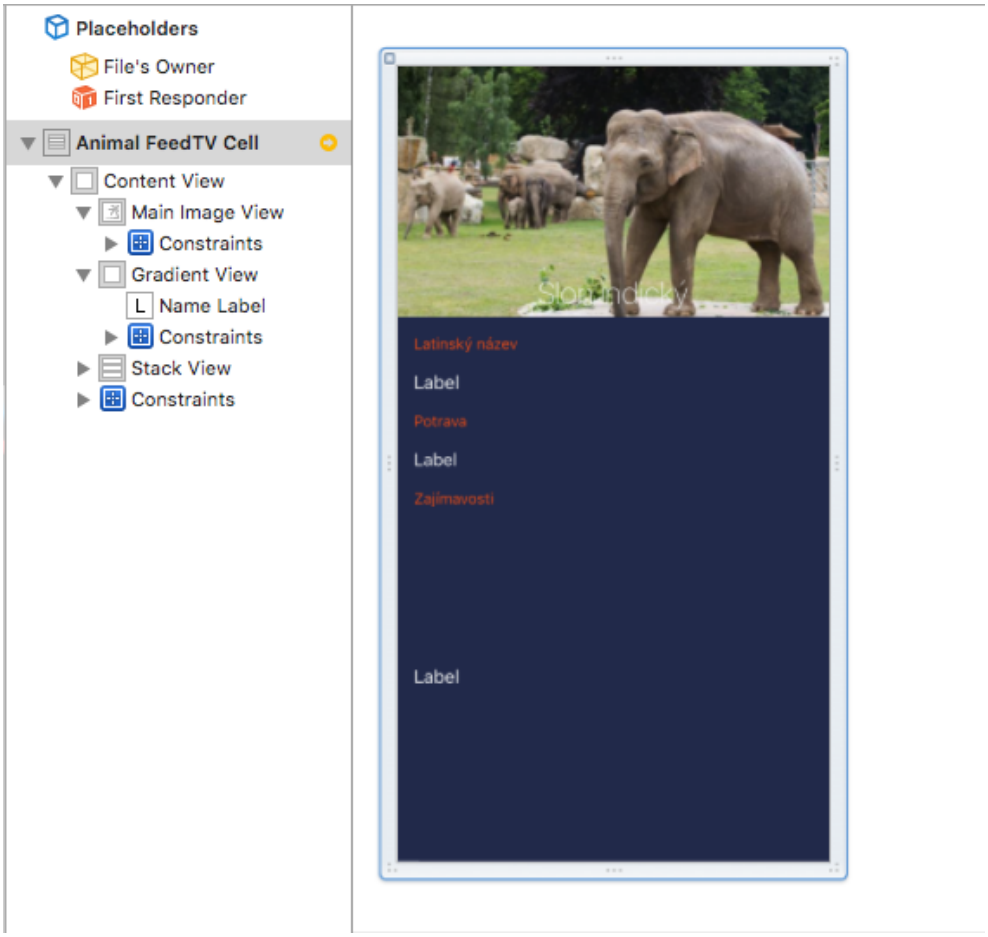
Autolayout

Autolayout je přizpůsbení vzhledu aplikace k různým displejům. Má podporu i v kódu, ale její využitelnost bez nějakých knihoven je velmi obtížné. Na to abychom nadefinovali pozici, nebo rozměry nějakého elementu potřebujeme napsat dlouhé řádky kódu. S využitím např. knihovny SnapKit, toho lze dosáhnout snáze.

Xib

V projektu využívám Xiby pro definování view vrstvy. Xib je zkratka pro Xcode Interface Builder. K vytvoření fungujícího uživatelského rozhraní můžu pomocí drag and drop funkce vytvořit různá tlačítka, textové pole a jiné objekty na plátně designu. Tímto způsobem se tak většinou oddělí uživatelské rozhraní od implementace a zabrání se tím michání více věcí[16].

Jsou tu také nějaké omezení, kdy je nutné používat kód, například když se používájí různé animace.



Obrázek 5.2: Řádek tabulky se zvířetem

Řádky tabulky

Aplikace ma jasně daný design. Správně navržená aplikace bý měla používat co nejvíce komponent a elementů, které se dají opětovně využít. V aplikaci se vyskytujou většinou Controllery, které obsahujou tabulku tableView, k tomu jsou nepostradatelné řádky tabulky UItableViewCell.

Abych se vyhnul opakováním se psaním stejného kódu, navrhnul jsem několik typů řádků, které budu využívat napříč v celé aplikaci. Tyto řádky dědí od hlavní třídy UITableViewCell a ke každému implementačnímu souboru patří i jeho Xib soubor. V tableView pak musím určit, který typ řádku se bude využívat, typicky funkcí tableView.register: forCellReuseIdentifier. Tato funkce používá řetězcovou identifikaci, která se podle mého názoru nejeví jako dostatečně bezpečná. Stačí abych udělal v nějaké části překlep, aplikace by se na místě použití spadla.

```
public extension UITableViewCell
{
    static public var autoReuseIdentifier : String {
        return NSStringFromClass(self)
              + "AutogeneratedIdentifier"
    }
}
```

Abych se vyhnul řetězcovému identifikaci řádku, napsal jsem zde extension funkci pro defaultní třídu UITableViewCell, která bude umět generovat svůj vlastní identifikátor. Docílil jsem toho pomocí funkce NSStringFromClass a pro větší bezpečnost připojil vlastní přívlastek.

Zobrazování obrázku v aplikaci

Zde jsem narazil na první problém, kdy jsem potřeboval odkaz pro obrázek zvířete ze serveru. Server totiž link obrázku posílal obsažený v html řetězci, který byl uložený v databázi. Bylo proto nutné projít si řetězec a vytěžit z toho daný odkaz. Problém se vyřešil pomocí regulárního výrazu. Naštěstí v iOS máme na to funkce. Nadefinoval jsem proto vzor regulárního výrazu a pomocí funkce NSRegularExpression.matches jsem si vyhledal pozice vyhledáváného řetězce a posléze vytvořil podřetězec, který obsahoval odkaz obrázku.

```
regex = try NSRegularExpression(pattern: pattern,
 options: NSRegularExpression.Options(rawValue: UInt(0)))
let opts = NSRegularExpression.MatchingOptions
(
 rawValue: UInt(0)),
 range: range
)
let matches = regex.matches(in: searchString, options: opts)

if let match = matches.first {
   let matchString = (searchString as NSString)
    .substring(with: match.rangeAt(1))

   return matchString.trimmingCharacters(in: .whitespaces)
}
```

Zobrazování obrázku v jednotlivých UIImageView[18] je vyřešeno metodou setImage, dodávaný knihovnout SDWebImage. Této funkci stačí poskytnout odkaz, zastupující obrázek a o asynchronní stahování a ukládání obrázku do paměti se už postará.

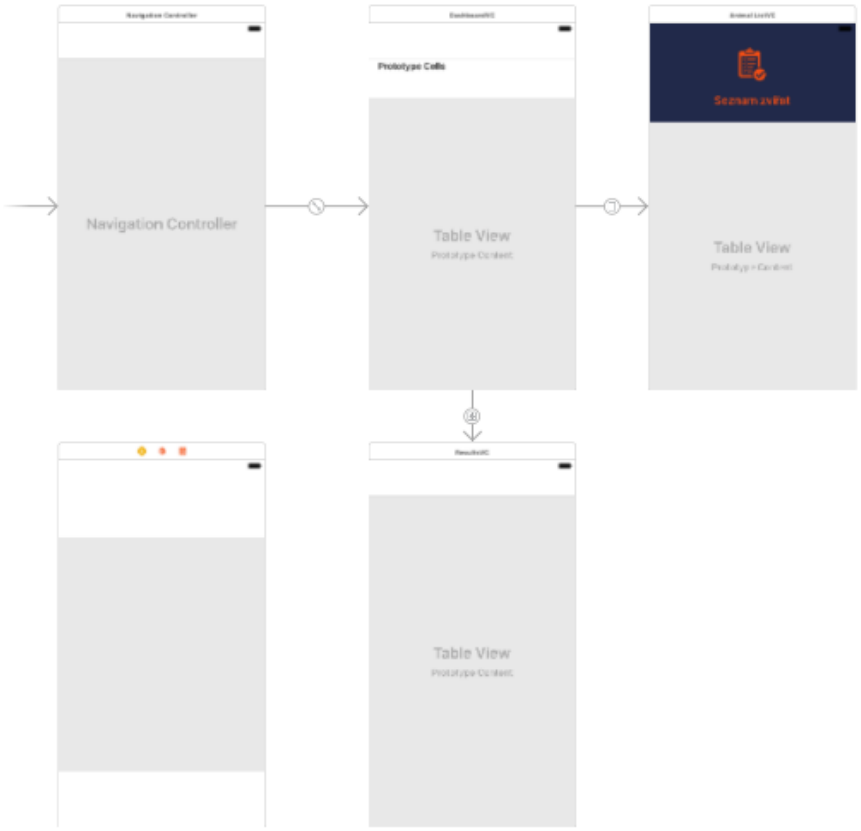
[18]Nativní komponenta v iOS, která umí zobrazit obrazek

5.5 Logická vrstva v aplikaci

Storyboardy

Pro kompletní navigaci mezi jednotlivými obrazovky jsem využil storyboardy. Vývojáři často dělají chybu, že všechny obrazovky dávají do jednoho storyboardu. Ze storyboardu se potom stává nečitelná pavučina obrazovek a celé vývojové prostředí se důsledkem toho zasekává. Také se pak vyskytují slučovací problémy, když se soubor verzuje a více vývojářů pracuje paralelně na jednom souboru storyboardu.

Storyboard je v principu totiž jen generovaný XML soubor, ve kterém jsou definováne dané objekty. Xcode nám pak z toho souboru dokáže zobrazit designové plátno, na kterém už vidíme jednotlivě rozvržené elementy. Proto jsem pro každou logickou sekci aplikace vytvořil jeden storyboard a v ní jsem definoval menší počet obrazovek, které se pro danou sekci budou využívat.



Obrázek 5.3: Ukázka storyboardů v aplikaci

Předávání dat mezi jednotlivými obrazovky

Existuje několik způsobů, jak vyřešit problematiku předávání dat mezi controllery. K tomu se ve vývoji používají Segue přechody. Pomocí drag and drop funkce na designovém plátně spojím dva obrazovky aplikace a přídělím mu přechodový identifikátor. v kódu pak můžu přechod vyvolat funkcí performSegue, kterému poskytnu přechodový identifikátor. Následně musím naimplementovat funkci prepareForSegue, ve kterém předám data následující obrazovce.

Dalším způsobem je vytvoření instance obrazovky pomocí storyboardu a její funkce instantiateViewControllerWithIdentifier v implementačním kódu. Zde je nutné mít přidělený identifikátor obrazovky, který se dá nastavit ve storyboard GUI. Po vytvoření přidělím instanci data a pomocí navigačního zasobníku navigationController provedu tranzici z jedné obrazovky do druhé. Tuto tranzici vyvolám pomocí funkce navigationController.pushViewController.

Fulltextové hledání v aplikaci

Jeden z funkčních požadavků bylo hledání zvířat podle názvu. Vyhledávání jsem implementoval pomocí nativní komponenty UISearchController a jeho UISearchBaru, který jsem umístil v horní části domovké obrazovce aplikace. po kliknutí na panel se provede animace, která přesune tento panel na navigační lištu a zobrazí klávesnici pro zadání textu. Toto vše nám zajístí nativní komponenta UISearchController, kterou jsem vytvořil uvnitř třídy DashBoardVC. Při vytvoření jsem komponentě předal instanci třídy ResultsVC, která zobrazuje výsledky hledání. po vytvoření komponenty je nutno přiřadit její vyhledávací panel do hlavičky tabulky, kterou máme implementovanou v DashBoardVC.

Následně je nutné implementovat delegační funkci, která je vyvolána při každé změně textu ve vyhledávacím panelu. v této funkci jsem naimplementoval logiku vyhledávání daného textu v poli zvířat. Níže je uvedena jednoduchá logika, jak vyfiltrovat pole zvířat pomocí filtračního funkce pro jakékoliv pole. Je to jeden z mnoha funkcionálních programových konceptů jazyka Swift. Je zde pouze nutné napsat porovnávací funkci pro vyhledávaný text. Jak je z obrázku vidět, porovnávám zde mezi názvem zvířete a daným textem bez ohledu na velká a malá písmena.

```
func filterContentForSearchText(searchText: String) {
    searchResultsVC.filteredAnimals =
    datasource.filter { animal in
        return animal.name.lowercased()
          .contains(searchText.lowercased())
    }
    searchResultsVC.tableView?.reloadData()
}
```

5.6 Použité technologie pro vývoj

Externí knihovny

Alamofire je knihovna, používaná na komunikaci se serverem. Třída API-Request s touto knihovnou přimo pracuje.

SwiftyJSON je knihovna starající se o searializaci JSONu a mapování do jednotlivých objektů. v aplikaci jej využívám v modelové vrstvě.

Realm je řešení perzistence v aplikaci.

SDWebImage je knihovna starající se o stahování a cachování obrázků.

ReSideMenu je knihovna umožňuje vytvořit přehledné a pohodlné boční menu aplikace. Je jí potřeba nakonfigurovat v aplikačním delegátu a předat instanci levé obrazovky menu a pravou obrazovku zobrazující hlavní obsah.

Verzování aplikace

Pro vývoj aplikací je vhodné jednotlivé změny v kódu verzovat. Je to velice dobré při hledání chyb v aplikaci, které se budou objevovat. Pomocí verzování můžeme tak vyhledat, které ve kterých verzích aplikace se chyby vyskytují. v mobilním vývoji se nejvíce využívá git jako verzovací nástroj.

Git

Git se může používat jak v příkazovém řádku, nebo i ve vývojovém prostředí Xcode. Já jsem používal Git nástroj i s kombinací Xcode. Xcode totiž má podporu asistovacího editoru, který umí zobrazovat více verzí kódu v souboru. Pomocí tohoto editoru můžu přehledně srovnávat změny mezí různými větvemi v git stromu.

Analytické nástroje

V projektu byl využit Crashlytics, který patří mezi analytické nástroje pro zpětnou vazbu. Je to platforma pro reporting pádu iOS a Android aplikací. S crashlytics získáme informace o pádu aplikace v reélném čase. Pomocí knihovny crashlytics nemusíme psát žadné extra kódy v aplikaci[17] do aplikace stačí naimportovat knihovnu pomocí cocoapods a po té podle návodu v několika krocích nakonfigurovat projekt.

Kapitola 6

Testování

Pro vývoj každého softwaru jsou testy nutnou zálžitostí. v této kapitole se proto budu zabývat několika typy testování. Popíšu zde jak jsem kontroval správnou funkčnost aplikace.

6.1 Unit testing

Pro unit testing jsem využil testovací sadu XCTests, kterou nám nabízí samotný Xcode. Jeho samotná podpora v Xcode je velmi dobrá.

XCTests

V projektu jsem přidal nový testovací scénář LexiconTests, který mi vytvořil testovací soubor LexiconTests. Struktura testovacího souboru je velmi jednoduchá, v souboru jsou funkce setUp() a tearDown() a samotné testovací funkce. Funkce setUp() se chová jako inicializiční funkce, která se vyvolá vždycky před začátkem testovácí funkce. Zde se většinou inicializují a konfigurují objekty, které se budou testovat. Funkce tearDown() se volá automaticky po každém testu. Zde objekty čistíme a uvolňujeme z paměti.

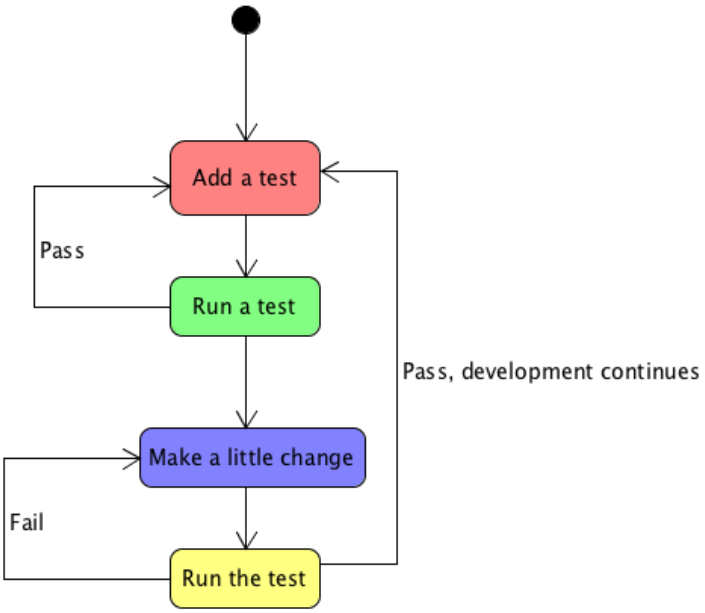
```
var animal : Animal!
override func setUp() {
    super.setUp()

    animal = Animal()
}

override func tearDown() {
    animal = nil
    super.tearDown()
}
```

Test driven development

Při vytváření modelové vrstvy jsem postupoval metodou TDD[19]. Princip je zde napsat chování objektu testy ještě předtím, než se začne psát samotný objekt. Správné chování na začátku je test, který neprojde. Aby test prošel musí objekt Animal obsahovat shodné hodnoty, které jsem na začátku definoval. Pomocí Xcode funkce XCAssert otestujeme shodnost jednotlivých hodnot. po neprošlém testu jsem naimplementoval třídu jen potřebným kódem aby test jednoduše prošel, po té jsem znovu provedl refaktorizaci implementačního kódu a přešel jsem na další test. Těmito kroky jsem provedl celé implementační řešení objektů v modelové vrstvě.



Obrázek 6.1: Programování řízené testy

Mockování objektů

Při psaní unit testů jsem musel potřebné objekty k testování mockovat. Je potřeba mít testy od sebe izolované. Někdy je totiž potřeba vědět jestli jedna funkce vyvolává druhou funkci. Například když potřebuju zjistit jestli se zavolá funkce, která provede parsování html řetězce. Mě konkrétně v tomto testu totiž nezajímá jak se provede parsování, ale jestli se parsovací funkce zavolá. Otestování parsovací funkce totiž patří již do separátního testu.

[19] Programování řízené testy

```
class AnimalMockObject: Animal {
    var getStringFromPatternWasCalled = false
    override func getStringFromPattern(pattern: String,
                                       searchString : String)
                                        -> String {
        getStringFromPatternWasCalled = true
        return ""
    }
}
```

Příkladem je mockování třídy Animal a přepsání parsovací funkce getStringFromPattern. Tato funkce se při správném chování má zavolat při provedení funkce getImageURL. Funkce getImageURL má za úkol vytěžit odkaz obrázku z html řetězce.

```
    func testImageParsing() {
        let animalMock = AnimalMockObject()
        animalMock.animalDescription = "testDescription"
         _ = animalMock.getImageURL()
        XCTAssert(animalMock.getStringFromPatternWasCalled)
    }
```

6.2 Uživatelské testování

Distribuce aplikace pro testovací účely

Hotovou beta verzi aplikaci je potřeba důkladně otestovat. K otestování aplikace jsem dokázal sehnat malý seznam reálných uživatelů, kteří budou aplikaci testovat na uživatelském úrovni. Aby si testeři aplikací mohli otestovat aplikaci, která ještě nebyla výdána na AppStore, musel jsem vytvořit betu verze aplikace pomocí distribučního nástroje. Kvůli bezpečnostním důvodům je celkem obtížné instalovat aplikace, které nejsou uvedené na AppStore. K tomu abychom mohli distribuovat aplikace, musíme mít vyvojářský účet od společnosti Apple a registrovat zařízení do testovacích profilů. Pokud máme firemní vývojářský účet, můžeme tento krok přeskočit a zabalit aplikaci pomocí firemního účtu. Firemní vývojářský účet ma privilegium firemní distribuce, kde není potřeba registrovat žádná zařízení a aplikaci pak může kdokoliv nainstalovat.

Dále jsem zvolil distribuční nástroj HockeyApp, aby testeři byli schopni snáze instalovat aplikace. HockeyApp umožňuje totiž instalovat aplikace pomocí odkazu, který se rozesílá testerům. Existujou další alternativy jako jsou TestFlight a TestFairy, obě tyto alternativy jsou velmi podobné HockeyAppu a nevidím v nich žádný zásadní rozdíl. Zvolil jsem HockeyApp kvůli osobním zkušenostem.

Testovací scénář

Pro testery jsem zjednodušil testování připravením testovacího scénáře, podle kterého měli postupovat. Scénář vedl participanty těmi hlavními případy užití. Testeři byli většinou dlouhodobými uživateli iOS nebo Android zařízení. Níže uvádím jen potestové výsledky testovacích scénářu od jednotlivých testerů.

První tester

Zkušená uživatelka iOS zařízení.

1. *Jaká byla podle tebe celková přehlednost aplikace, bylo lehké se v něm orientovat?* Osm z deseti

2. *Jaká byla responzivnost aplikace, nezasekla nebo nespadla ti aplikace v nějakém bodě?* Aplikace reagovala na vstupy velice rychle. Ne.

3. *Změnil/a bys něco na uživatelském rozhraní? A Proč?* Seznamy zvířat bych seřadila abecedně. Seznam lokací bych také logicky seřadila. Při vybírání zvířete ze seznamu se objevuje bílá čára v momentě přepnutí obrazovky. po kliknutí na jednu třídu zvířete mě aplikace nasměrovala na prázdný seznam.

Z výsledku je vidět, že testerka si aplikaci důkladně prošla a vytkla několik důležitých chyb, kterých jsem si nevšimnul. Všechny seznamy v aplikaci jsem pak seřadil abecedně podle názvů. Zjistil jsem také proč se objevuje divný separátor při změně obrazovky. Způsobovala to defaultní barva řádky v tabulce. Poslední chyba byla složitější. Zde jsem musel pro každou třídu zvířat vylistovat celý seznam zvířat a následně vyfiltrovat seznam tříd podle odpovídajícího seznamu.

Druhý tester

1. *Jaká byla podle tebe celková přehlednost aplikace, bylo lehké se v něm orientovat?* Přehlednost dobrá. Význam dashboardu mi unikal, dokud mi ho tvůrce neobjasnil.

2. *Jaká byla responzivnost aplikace, nezasekla nebo nespadla ti aplikace v nějakém bodě?* Nezasekla se. Responzivnost uspokojivá.

3. *Změnil/a bys něco na uživatelském rozhraní? A Proč?* v mapě zoo chybí uživatelská poloha, chtěl bych vědět kde se momentálně nacházím. Navigace na mapě nefungovala.

Druhý tester měl pár důležitých poznámek. Sekce dashboard, byla možná matující a proto jsem tuto sekci přejmenoval na Seznam Zvířat. Dále jsem umístil uživatelskou polohu a opravil navigaci k zoo.

6.2.1 Třetí tester

1. *Jaká byla podle tebe celková přehlednost aplikace, bylo lehké se v něm orientovat?* Aplikace byla velmi přehledná.

2. *Jaká byla responzivnost aplikace, nezasekla nebo nespadla ti aplikace v nějakém bodě?* Aplikace fungovala bez problémů.

3. *Změnil/a bys něco na uživatelském rozhraní? A Proč?* U vyhledávaného zvířete by mohla být možnost vyhledat ho na mapě zoologické zahrady.

Poslední tester byl s aplikací vcelku hodně spokojený. Chtěl by mít rozšíření aplikace o další možnost lokace zvířete. Tato funkcionalita by v aplikaci měla velkou využitelnost, ale bohužel nám současné datasety od pražské zoo tuto možnost nenabízí. Respěktivě jsou tyto informace obsažené v datasetech, ale bohužel provázanost těchto dat, není dostatečně kompletní. v budoucnu bych to viděl jako dobrou požadavek na doplňení funkčnosti od pražské zoo.

6.3 Akceptační testy

Po dohodě s vedoucím práce jsme se dohodli, že vývoj je dokončen, pokud aplikace splňuje následující funkcionality.

Obrazovky v aplikaci

- Boční menu – Boční menu bude dostupné z levého tlačítka obrazovky. Bude usnadňovat navigaci v aplikaci.

- Seznam zvířat – Tato obrazovka bude zastupovat funkci domovské stránky aplikace. Obsahuje všechna zvířata, která naleznem v zoo, seřazena v přehledné tabulce podle abecedy. Každý řádek tabulky bude obsahovat fotku zvířete se jménem a jednu zajímavost o něm. po kliknutí na řádek se aplikace přesune do detailu zvířete. Zde naleznem informace další detailnější informace o zvířeti jako je potrava, výskyt a rozmnožování.

- Zvířata dle dělení – Jsou to obrazovky, na kterých budou zvířata seskupena do jednotlivých výše uvedených kategorií. Také tu bude možnost zobrazení detailu zvířete.

- Interaktivní mapa zoo – Pro tuto obrazovku bude použita nativní komponenta map v iOS. Zaměříme se zde na okolí zoo a zobrazíme zde body, na kterých se zvířata vyskytují. Bude to možnost navigace do nativních map v mobilu a výpočet trasy od aktuální polohy až k Zoo Praha.

- Informace o aplikaci – Zde budou veškeré zdroje, které budou v aplikaci využity.

Tyto navržené funkčnosti byli úspěšně dokončeny a tudíž akceptační testy proběhly úspěšně.

Závěr

Cílem této práce bylo vytvotření mobilní aplikace, která využívá portálu opendata.praha k vytěžení informací, poskytnutých od pražské zoo a atraktivní formou je prezentuje uživatelům aplikace.

Během analýzy bylo zjištěno chybnost datasetů, ve kterých se vyskytovaly nekompletní a chybně provázaná data. Důsledkem toho jsme s vedoucím práce naplánovali schůzku s technickým vedením zoo a portálu opendata.praha, kde jsme probrali všechny nedostatky a návrhy ke zlepšení. Po kratší pauze se věci daly do pořádku a moje práce mohla dále pokračovat.

Před samotným návrhem aplikace jsem provedl důkladný průzkum podobných aplikací a provedl hrubý odhad ziskovosti těchto řešení.

Při návrhu obrazovek jsem kladl důraz na požadavky, které jsem si nadefinoval při analýze dostupných datasetů. Výsledný drátěný model aplikace jsem předal vedoucímu ke zhodnocení a po několika iteracích s vedoucím jsem začal se samotnou implementací.

V kapitole o implementaci popisuju jednotlivé vrstvy v aplikaci a řešení komplexnějších problémů, které se vyskytly při vývoji.

Během implementace modelové vrstvy jsem zkusil metodu programování řízené testy. Tuto metodu zpětně hodnotím jako velice obtížnou a málo přínosnou pro tento typ aplikace. Tato metoda je velice časově náročná, pokud nemáme představu přesného chování jednotlivých tříd. Při každé menší změně objektu je pak nutné testy znovu přepisovat. Podle mého názoru jsou pro tyto typy aplikací důkladné uživatelské testy mnohem důležitější a přínosnější.

Po dokončení implementace jsem na závěr provedl uživatelské testování, kde jsem obdržel několik objektivních připomínek k aplikaci, podle kterých jsem následně výslednou aplikaci upravil.

Literatura

[1] Quartz: *The second largest part of Apple's revenue now comes from something called Services [online].* [cit. 2016-29-12]. Dostupné z: `http://qz.com/674049/the-second-largest-part-of-apples-revenue-now-comes-from-something-called-services/`

[2] OtevřenáData.cz: *Co jsou otevřená data [online].* [cit. 2016-29-12]. Dostupné z: `http://www.otevrenadata.cz/otevrena-data/co-jsou-otevrena-data/`

[3] opendata.cz: *Proč otevřená data [online].* [cit. 2016-29-12]. Dostupné z: `http://www.opendata.cz/en/node/29`

[4] zdrojak.cz: *REST: architektura pro webové API [online].* [cit. 2016-20-12]. Dostupné z: `https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/`

[5] mobilmania.cz: *Čísla mluví jasně: Jaké mobilní OS používají Češi [online].* [cit. 2016-29-12]. Dostupné z: `http://www.mobilmania.cz/clanky/cisla-mluvi-jasne-jake-mobilni-os-pouzivaji-cesi/sc-3-a-1328855/default.aspx`

[6] Apple: *UITabBarController [online].* [cit. 2016-29-12]. Dostupné z: `https://developer.apple.com/reference/uikit/uitabbarcontroller`

[7] Techotopia: *The History of Objective-C [online].* [cit. 2016-29-12]. Dostupné z: `http://www.techotopia.com/index.php/The_History_of_Objective-C`

[8] Clayton, J.: *The Swift Apprentice.* Razeware LLC, třetí vydání, ISBN 978-1942878131.

[9] Atomic Object: *Why I Prefer AppCode over Xcode [online]*. [cit. 2016-29-12]. Dostupné z: https://spin.atomicobject.com/2014/08/02/appcode-vs-xcode

[10] Prolific interactive: *AppCode vs Xcode [online]*. [cit. 2016-29-12]. Dostupné z: http://blog.prolificinteractive.com/2015/05/15/appcode-vs-xcode/

[11] zdrojak.cz: *Úvod do architektury MVC [online]*. [cit. 2016-29-12]. Dostupné z: https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[12] raywenderlich.com: *Model-View-Controller (MVC) in iOS [online]*. [cit. 2016-22-12]. Dostupné z: https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach

[13] objc.io: *VIPER [online]*. [cit. 2016-22-12]. Dostupné z: https://www.objc.io/issues/13-architecture/viper/

[14] Apple: *Coredata [online]*. [cit. 2016-29-12]. Dostupné z: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html

[15] Rollout: *iOS Databases: SQLLite vs. Core Data vs. Realm [online]*. [cit. 2016-20-12]. Dostupné z: https://blog.rollout.io/ios-databases-sqllite-core-data-realm

[16] Apple: *Interface builder [online]*. [cit. 2016-22-12]. Dostupné z: https://developer.apple.com/xcode/interface-builder/

[17] Fabric: *Crashlytics [online]*. [cit. 2016-29-12]. Dostupné z: https://docs.fabric.io/apple/crashlytics/overview.html/

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

MVVM Model View ViewModel

JSON JavaScript Object Notation

API Application programming interface

MVC Model View Controller

REST Representational State Transfer

Příloha B

Wireframy

Menu

- Dashboard
- Mapa
- Deleni podle trid
- Deleni podle biotopu
- Deleni podle lokality
- Info

Dashboard

Hledat zvire

Lev obecny

Lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti pellentesque

Detail zvirete

Previous | 1 | 2 | 3 | Next

Lorem ipsum dolor sit amet, maiores ornare ac fermentum,

Lorem ipsum dolor sit amet, maiores ornare ac fermentum,

Lorem ipsum dolor sit amet, maiores ornare ac fermentum,

Mapa

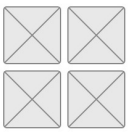
Tridy

- Option 1
- Option 2
- Option 3
- Option 4
- Option 5
- Option 6

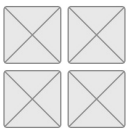
Seznam zvirat

- Some text
- Some text
- Some text
- Some text

Biotopy



Lokality

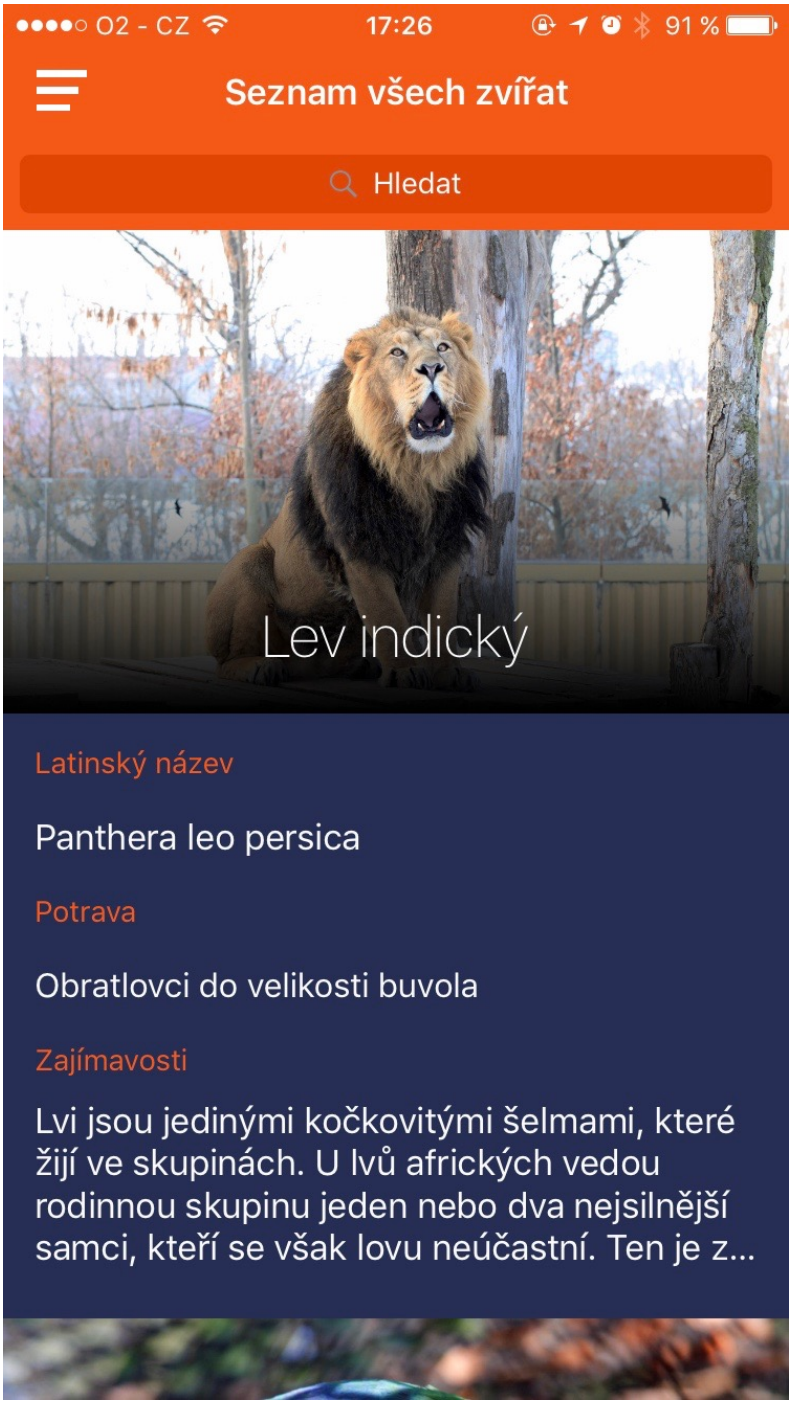


Info

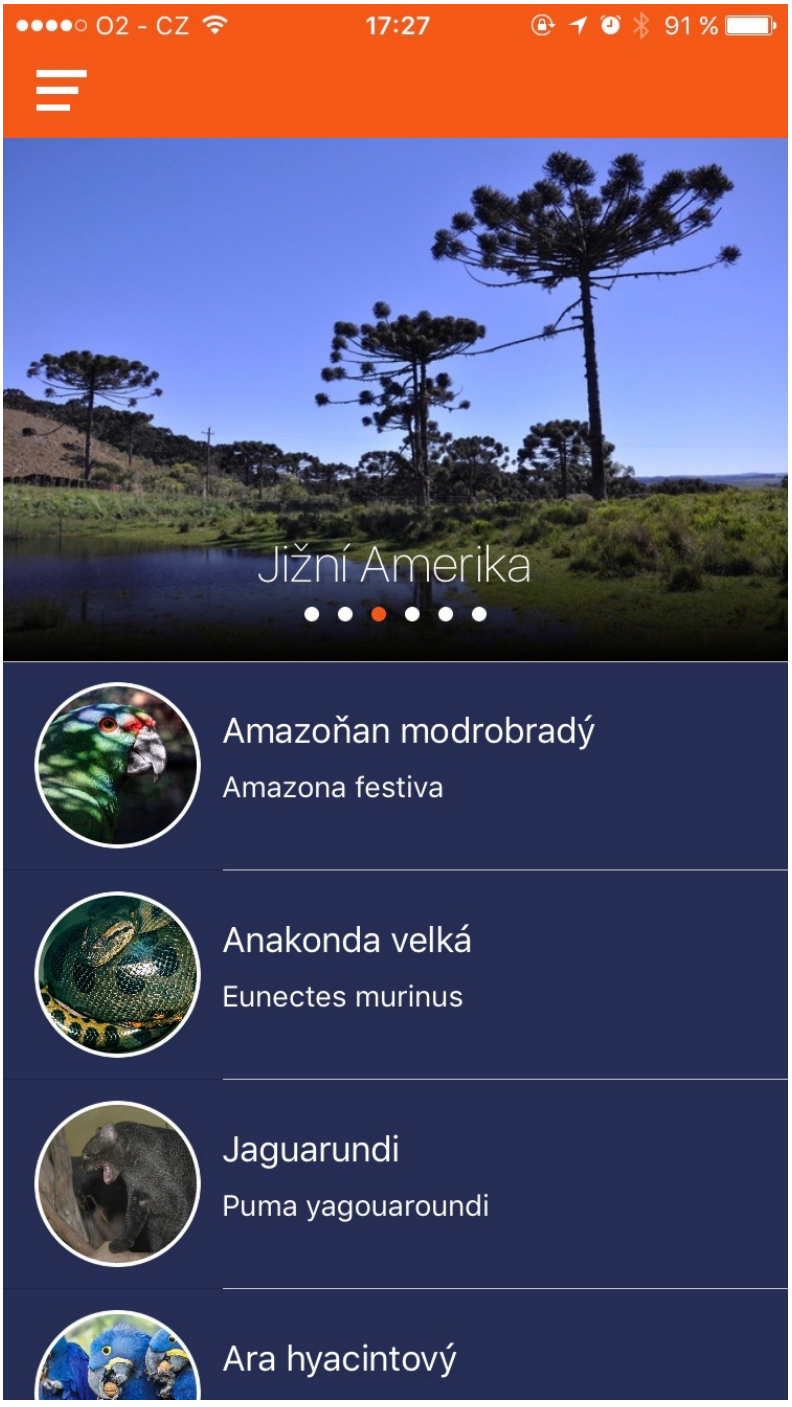
Lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti pellentesque quisque. In eget sapien sed, sit duis vestibulum ultricies, placerat morbi amet vel, nullam in in lorem vel. In molestie elit dui dictum, praesent nascetur pulvinar sed, in

Příloha C

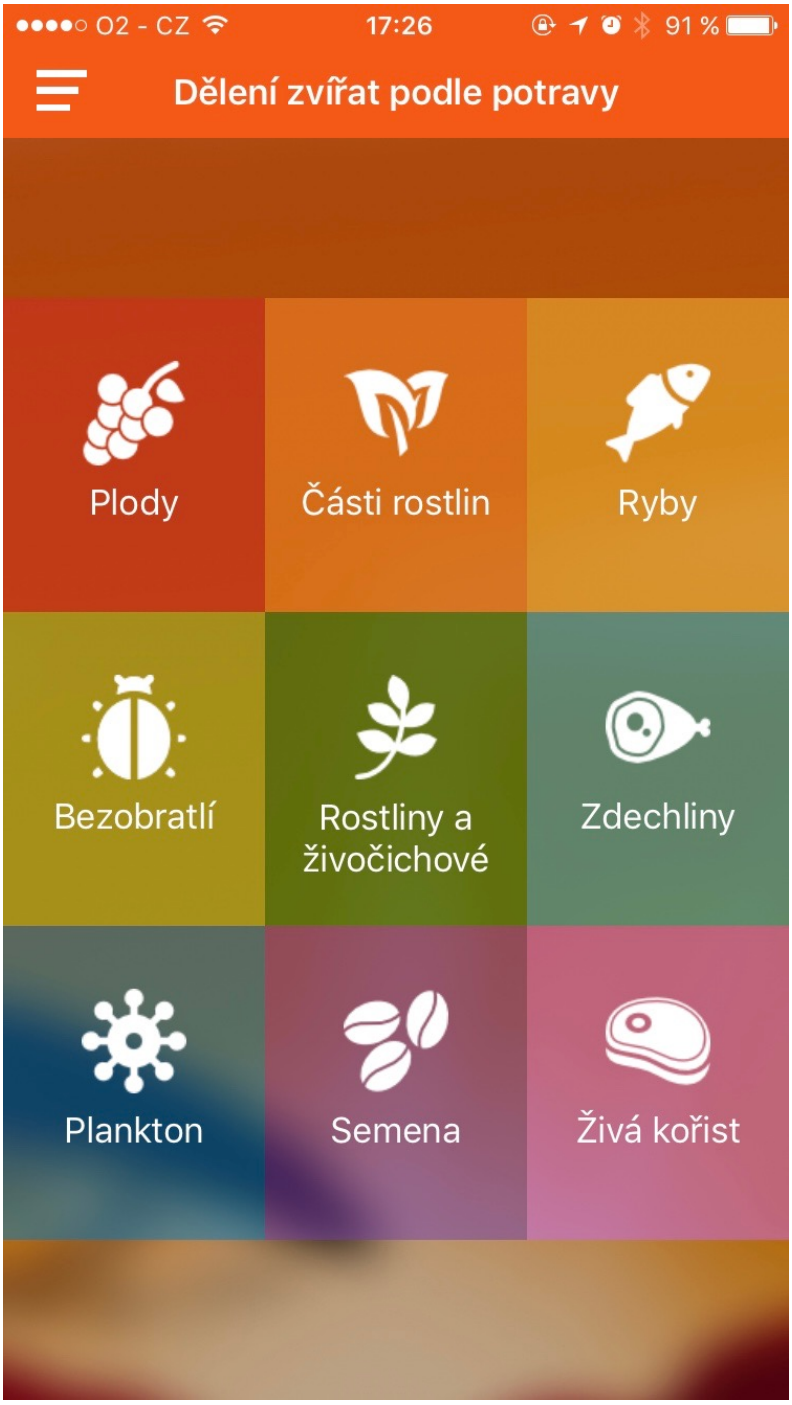
Ukázky aplikace



Obrázek C.1: Seznam zvířat



Obrázek C.2: Dělení zvířat podle kontinentů



Obrázek C.3: Dělení zvířat podle potravy

Příloha D

Obsah přiloženého USB

- `readme.txt` ..... stručný popis obsahu USB
- `src`
  - `impl` ..... zdrojové kódy iOS Aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `other` ..... přílohy