České vysoké učení technické v Praze

Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Knihovna pro aplikace typu virtuální naučné stezky na iOS

2017 Jiří Rychlovský

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Jiří Rychlovský

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: Knihovna pro aplikace typu virtuální naučné stezky na iOS

Pokyny pro vypracování:

Analyzujte stávající knihovnu pro vytváření virtuálních naučných stezek (dodá vedoucí práce) implementovanou pro operační systém (OS) Android. Knihovnu pro OS Android rozšiřte o funkcionalitu spojenou s trasováním GPS a zobrazováním úkolů vzhledem k pozici.
Dále analyzujte možnosti převodu této knihovny do OS iOS a navrhněte a implementujte její plnou funkcionalitu v tomto OS.
Funkčnost knihovny ověřte převodem dvou aplikací z OS Android. Pro telefony se jedná o aplikaci 'Z Kralup za Antonínem Dvořákem', pro tablety o aplikaci 'Satelitní navigace' (zdroje obou aplikací dodá vedoucí práce).

Seznam odborné literatury:

[1] Learning Swift: Building Apps for OS X and iOS. Paris Buttfield-Addison, Jon Manning, Tim Nugent. O'Reilly Media, Inc., 2016.
[2] https://play.google.com/store/apps/details?id=cz.scientica.kralupy
[3] https://play.google.com/store/apps/details?id=esero.scientica.cz.satnav
[4] https://developer.apple.com/reference/

Vedoucí: Ing. David Sedláček, Ph.D.

Platnost zadání do konce letního semestru 2017/2018

[redacted]
prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

L.S.

[redacted]
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 31.1.2017

ABSTRAKT

Tato diplomová práce pojednává o tvorbě a následném použití knihovny sloužící pro tvorbu aplikací typu naučné stezky na operačním systému iOS. Nejprve je popsána analýza již existujícího řešení pro Android systémy. Práce využívá nabytých poznatků získaných z této analýzy pro návrh převodu existující knihovny na cílený operační systém. Je zde popsána implementace aplikace „Satelitní navigace“ a aplikace „Z Kralup za Antonínem Dvořákem“, pomocí kterých je ověřena funkcionalita vytvořené knihovny. Dále je zde zhodnoceno, do jaké míry původní funkcionality bylo možné knihovnu převést a na závěr je přiblížen způsob testování funkčnosti aplikací a postup pro nasazení jejich beta verze.

KLÍČOVÁ SLOVA

knihovna, android, iOS, interaktivita, tablet, GPS

ABSTRACT

This diploma thesis informs about a project of developing a library for interactive trail games for iOS. At the beginning of the thesis there is an analysis of existing solution for Android platform and then it uses information, which analysis has discovered, to transform library to a target operating system. Thesis contains a description of an implementation of two applications, “Satellite navigation” and “From Kralupy to Antonín Dvořák”, which are used for testing a created solution. Then there is an evaluation how much is transformed library similar to the existing one. At the end of the thesis is a chapter, which describes testing of implemented applications and a procedure of deployment beta versions to app store.

KEYWORDS

library, android, iOS, interactivity, tablet, GPS

RYCHLOVSKÝ, Jiří. *Knihovna pro aplikace typu virtuální naučné stezky na iOS*.
Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická.
Diplomová práce.
Vedoucí práce: Ing. David Sedláček, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma *Knihovna pro aplikace typu virtuální naučné stezky na iOS* jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Praze dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Davidovi Sedláčkovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při jejím zpracování.

V Praze dne ....

(podpis autora)

OBSAH

ÚVOD

V moderní době při vývoji aplikací panuje snaha o jejich dostupnost co nejširší škále lidí. Napříč uživateli se ale bohužel objevuje rozdílnost v jejich vybavení, které k běhu aplikace mohou použít. Existuje více druhů platforem, které mobilní zařízení podporují. V případě vývoje aplikací na tato zařízení nastává problém ve vzájemné nekompatibilitě. Vývojáři se tedy musí potýkat s řešením, jak vyvíjet aplikace multiplatformě. Každá platforma je svým způsobem specifická a nelze jednoduše vyvinout aplikaci rovnou běžící na všech systémech.

Tématem této diplomové práce je vývoj knihovny, kterou bude možné využít k tvorbě interaktivních aplikací jak na systému Android tak na systému iOS. Knihovna má již funkční řešení pro Android a tak je nutné navrhnout její převod nově i na iOS. Pro zvolení vhodného postupu převodu knihovny je provedena analýza již existujícího řešení. Vytvoření verze pro jiný operační systém je možné vícero způsoby. Proto jsou zde prezentovány výsledky rešerše ohledně multiplatformního vývoje. Je zde zhodnocen nativní vývoj pro jednotlivé OS oproti několika vybraným frameworkům, které umožnují vývoj napříč platformami.

Z provedené analýzy poté vychází návrh implementace iOS verze knihovny. Práce obsahuje detailnější popis převodu původní funkcionality pod nový operační systém. Řešení by mělo zachovat funkcionalitu existující verze knihovny a musí podporovat snadné přidání nových typových úkolů, které bude možné v interaktivních aplikacích využívat.

Požadované vlastnosti knihovny jsou ověřeny implementací aplikací „Satelitní navigace“ a „Z Kralup za Antonínem Dvořákem“. Na závěr práce je popsáno testování těchto programů a postup vydání jejich beta verzí do App „Storu“.

1 ANALÝZA EXISTUJÍCÍHO ŘEŠENÍ

Na začátku této kapitoly je definováno, co znamená pojem aplikace typu virtuální naučné stezky. Pro tyto druhy aplikací aktuálně existuje knihovna, která slouží k vytváření nových stezek s co nejmenším vynaloženým úsilím. Pro práci s touto knihovnou, existující pod operačním systémem Android, je nutné zanalyzovat její implementaci, aby bylo následně možné správně navrhnout její převod na druhý operační systém. Tato kapitola se zabývá tím, jak stávající řešení funguje a jak je použito v již fungujících aplikacích, pomocí kterých bude nutné dle zadání otestovat funkčnost převedené verze knihovny.

1.1 Aplikace typu virtuální naučné stezky

Virtuální naučná stezka je aplikace na mobilní zařízení, kterou uživatel použije pro získání zajímavých informací o lokalitě, pro kterou je stezka určena. Různé aplikace tohoto typu mohou mít vzájemně podobný základ, pouze je třeba jednotlivé použití přizpůsobit obsahem konkrétnímu místu, kudy stezka vede. Zpravidla se jedná o aplikace, kde má uživatel zobrazenou mapu oblasti a vyznačený určitý počet důležitých míst. Zároveň v mapě vidí pro snadnější orientaci i svou GPS polohu. Jeho úkolem většinou bývá projít postupně všemi zobrazenými body a splnit na nich připravené úkoly.

V případě, že se uživatel přiblíží dostatečně blízko k nějakému konkrétnímu styčnému bodu, umožní mu aplikace vstup do množiny obrazovek, na kterých mu jsou poskytnuty dodatečné informace o místě, kde se aktuálně nachází, anebo musí splnit úkol, který se k jeho pozici tematicky váže. Jednotlivé stezky mohou sdílet množinu podobných znovupoužitelných obrazovek, pouze se budou lišit v konkrétním obsahu, který obrazovka zobrazuje.

1.2 Popis Android knihovny

Na základě myšlenky snadného vytváření nových stezek pro nové lokace byla vytvořena knihovna, která obsahuje znovupoužitelné obrazovky pro v mapě vymezené body. Aby byla tvorba nových aplikací ještě jednodušší, tvoří se sled předpřipravených obrazovek dle předpisu ve formátu XML.

Množina úkolů patřící k nějakému vyznačenému bodu, bude dále nazývána „kapitola“. Kapitola obsahuje úkoly jako je například výběr jedné správné odpovědi, výběr více správných odpovědí, rozhodování, zda zobrazený výrok je pravdivý či nikoliv a jiné. Zároveň je možné do kapitoly umístit i negenerické obrazovky, které budou vyvinuty pouze pro patřičnou stezku.

Každá obrazovka je složena z uživatelského rozhraní a k němu se váže logika, dle které obrazovka funguje. Je možné vybrat z několika různých rozložení, do kterých se skládají předdefinované funkční komponenty.

Knihovnu lze využít i pro vytvoření interaktivní aplikace, která nebude generována z předpisu. V takovém případě se do aplikace zanesou pouze předpřipravené úkoly. Knihovna tak není použitelná pouze pro jednodušší opakující se motivy, ale může fungovat pro usnadnění práce jen jako doplněk ke specifickým složitým úkolům.

Mapování XML elementů na funkční prvky

Struktura předlohového XML se řídí dle DTD předpisu umístěného uvnitř knihovny. Jedná se o soubor *question.dtd* a je v něm definováno, které tagy může předpis obsahovat, možné zanoření prvků do sebe, množina atributů, které lze tagu určit ale také hodnoty, které může tag či atribut mít. Tento předpis je popsán v tabulkách 1.1 a 1.2.

Kořenovým prvkem XML předpisu je tzv. *questionset*. Ten obsahuje množinu stanovišť, které se skládají z dílčích obrazovek. Z atributu *maxTries* tohoto elementu je zjištěno, kolik pokusů na splnění úkolu uživatel má. Potomci q*uestionsetu* jsou elementy *questionslide*, které jsou abstraktním objektem pro jednotlivé kapitoly. Každý *questionslide* má definovaný svůj název, titulek a typ rozložení stránky, který mu přísluší – tzv. „layout". Z každého *questionslide* lze získat hlavičku kapitoly, potřebné obrázky či videa a prvky *question*.

Question reprezentuje jeden úkol, který musí uživatel splnit. Má parametry rozdílné dle toho, který typ úkolu obsahuje. Druh úkolu je ale u všech *question* definován stejně a to pomocí atributu *type*. Například pro výběr více správných odpovědí se jedná o typ *multichoice* atpod.

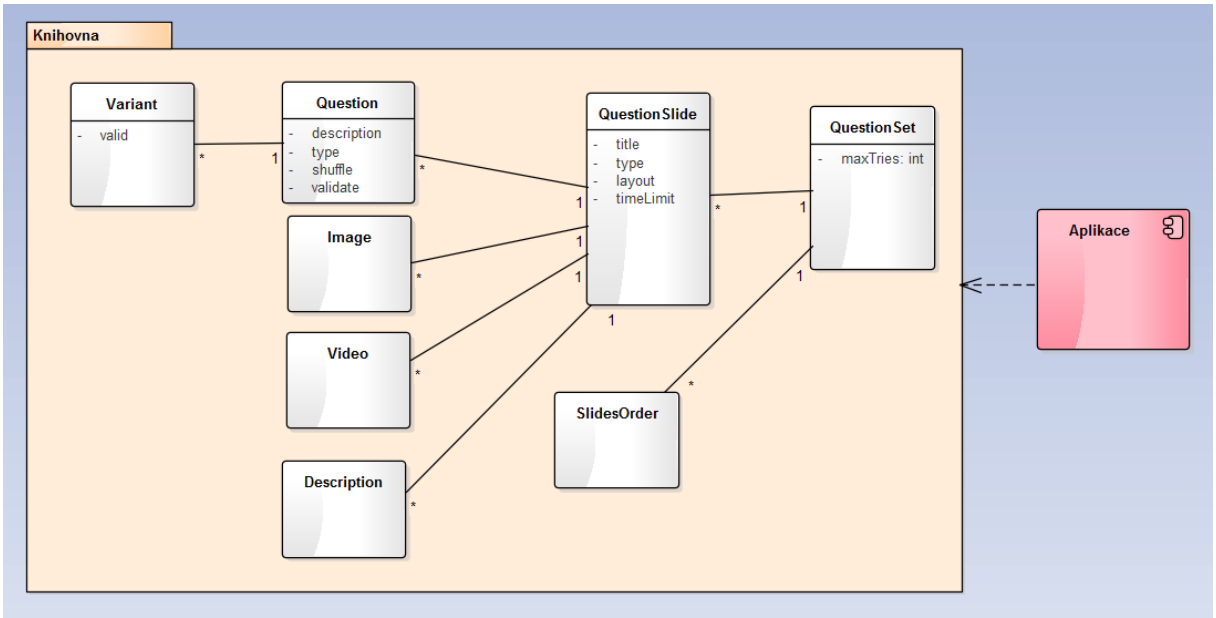
Pro zobrazení popisného textu na obrazovku slouží objekt *description*, který může příslušet jak objektu *question* tak celému *questionsetu*. Nejdůležitějšími potomky elementu „*question*" jsou tagy *variant*, které představují možnosti, ze kterých bude moci uživatel vybírat. Zároveň má tento element například atributy *shuffle* (značí, zda se mají možnosti před zobrazením promíchat) nebo *validate* (značí, zda se má úkol vyhodnocovat).

Tag	Význam tagu	Možní potomci
questionset	kořenový prvek	task
task	množina obrazovek (kapitola)	questionslide
questionslide	obrazovka uvnitř kapitoly	description, header, images, video, questions
header	nadpis kapitoly	žádní
description	informativní text na obrazovce	žádní
images	názvy obrázků	žádní
video	názvy videí	žádní
questions	množina úkolů	question
question	úkol jednoho typu	description, variant
variant	možná odpověď na otázku	žádní

Tab. 1.1 Tabulka povolených tagů v XML předpisu

Tag	Atribut	Význam atributu	Příklad hodnoty
questionset	maxTries	počet možných pokusů	5
task	name	jméno kapitoly	task01
questionslide	layout	rozložení prvků na obrazovce	v_quest_img
	maxTries	počet pokusů	2
	name	jméno obrazovky	task01_01
	title	nadpis obrazovky	Úkol 1
	type	generovaný úkol / vlastní úkol	custom / normal
question	shuffle	možné odpovědi v náhodném pořadí	true / false
	type	typ úkolu	singlechoice, multichoice, …
	validate	zda se má úkol vyhodnocovat	true / false
	precision	tolerance vepsané odpovědi	caseSensitive, strict, …
	buttonsOrientation	rozložení tlačítek	vertical, horizontal, …
	align	zarovnání prvků	centerAll, leftAll, …
	variantsCount	počet možných odpovědí	5
variant	valid	zda je možná odpověď správná	true / false

Tab. 1.2 Tabulka povolených atributů v XML předpisu



Obr. 1.1 Diagram namapovaných tříd na XML předlohu

1.3 Typy znovupoužitelných úkolů

Knihovna má v sobě připravené úkoly, které je možné ve vytvářených stezkách využít. Po naparsování XML předpisu odpovídají tyto generické úkoly třídě *Question*. Typ úkolu je v předloze určen atributem *type*. Všechny úkoly můžou mít textové zadání v prvku *description*. Možnosti, ze kterých v úkolu bude uživatel vybírat, jsou zde reprezentovány *objektem* variant. V této podkapitole je výčet těchto typů a zároveň krátký popis, k čemu se který úkol využívá.

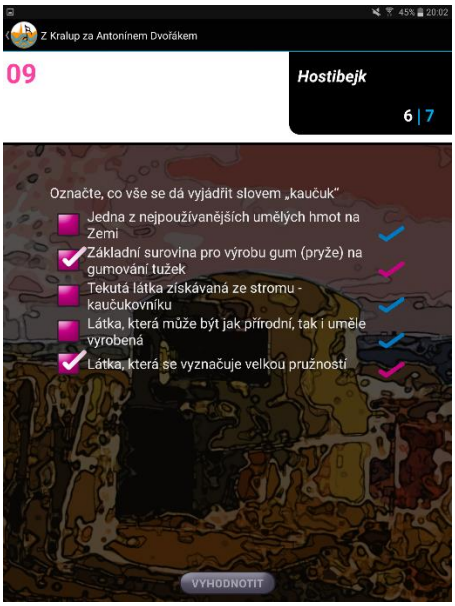
1.3.1 Singlechoice

V těchto úkolech bude uživateli zobrazena otázka a několik možných odpovědí. Uživatel bude moci vybrat pouze jednu z připravených odpovědí a pouze jedna z odpovědí bude správná.

Obr. 1.2 Ukázka úkolu Singlechoice

1.3.2 Multichoice

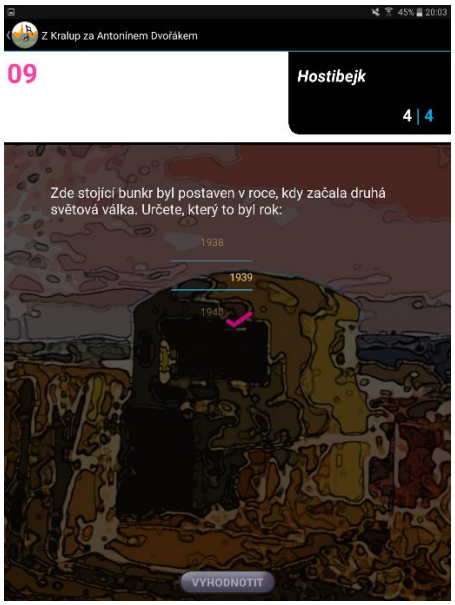
Jedná se o úkol, kde na položenou otázku může být více správných odpovědí a uživatel musí vybrat více odpovědí tak, aby se jím vybrané možnosti shodovali s množinou správných řešení.



Obr. 1.3 Ukázka úkolu Multichoice

1.3.3 Intervalquestion

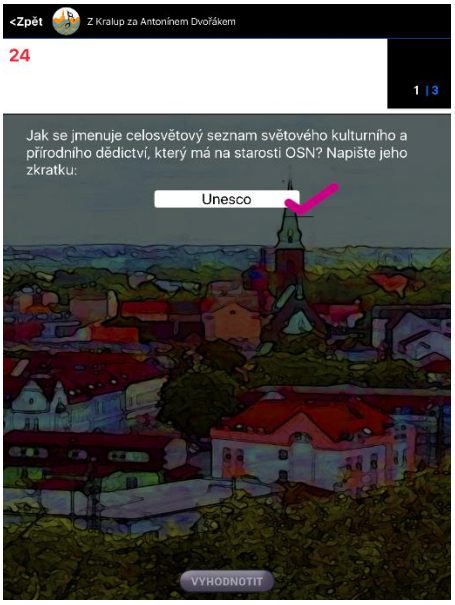
Uživateli je zobrazena rolovací nabídka s možnostmi, které jsou vygenerované v určitém rozsahu po určitém kroku. Jeho úkolem je správně zvolit číslo v intervalu, který je definován jako přijatelný.



Obr. 1.4 Ukázka úkolu Intervalquestion

1.3.4 Filltext

V tomto úkolu bude muset uživatel pomocí klávesnice doplnit do připravených oblastí vlastní text. Doplněný text pak musí odpovídat jedné z možností z množiny správných odpovědí určených pro konkrétní doplňovací oblast.



Obr. 1.5 Ukázka úkolu Filltext

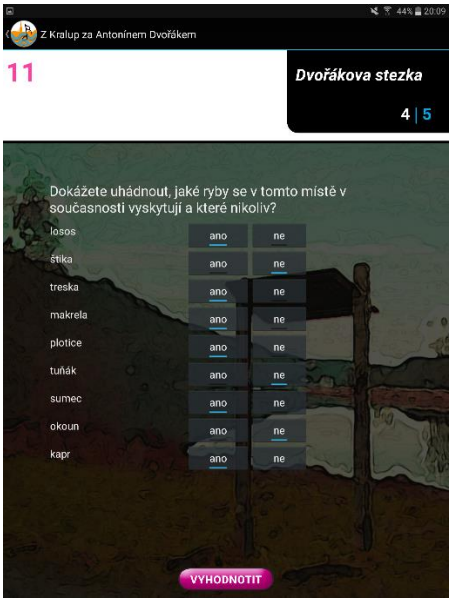
1.3.5 Numberquestion

Uživateli je zobrazena otázka, na kterou je správná číselná odpověď. Jeho úkolem je do předpřipravené oblasti pomocí klávesnice vyplnit správné číslo. Jedná se například o úlohy, kde má uživatel za úkol spočítat nějaký příklad.

Obr. 1.6 Ukázka úkolu Numberquestion

1.3.6 Togglebuttonsgrid

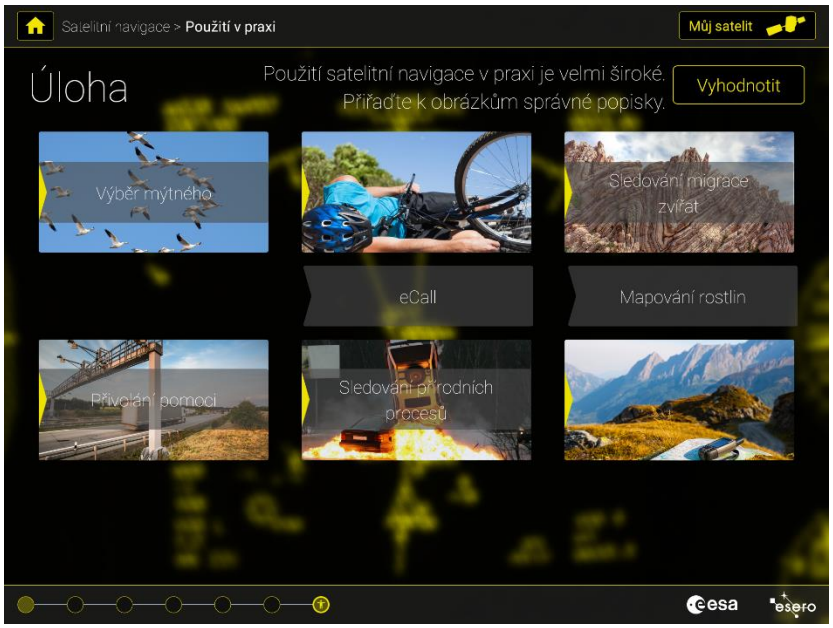
Uživatel musí v této úloze rozhodnout, zda se zobrazeným výrokem souhlasí či nikoliv. Pro každý připravený výraz tedy může zvolit svou odpověď pomocí kliku na tlačítko ANO resp. NE.



Obr. 1.7 Ukázka úkolu Togglebuttonsgrid

1.3.7 Dragdrop

Uživatel musí v této úloze tažením prstu přesunout objekty na správné místo. Jedná se například o úkol přiřazení dvojic, kde je nutné přetáhnout objekt vedle zobrazeného prvku, ke kterému správně patří.



Obr. 1.8 Ukázka úkolu Dragdrop

1.4 Použití knihovny v aplikacích

Po připojení knihovny k vytvářené aplikaci je možné využít některý z množiny předpřipravených úkolů, které knihovna obsahuje. Pokud aplikace potřebuje pouze tyto funkční celky, je třeba pro ně vytvořit v aplikaci *layout*, který se pak připravenému úkolu předá. Na různé úkoly jsou kladeny různé požadavky, takže pro některé objekty, se kterými se má v úkolu pracovat, je nutné specifikovat tag, aby knihovna mohla důležité objekty rozpoznat. Některým úkolům se ale musí předat odkazy (číslo ID) na použité prvky, protože je úkol sám vyhledat neumí.

Zároveň mohou aplikace využít funkcí, které analyzovaná knihovna nabízí. Ta obsahuje parser, který namapuje XML elementy na obdobné Java třídy. Parser využívá funkcionalit z připojené třídy *XMLPullParser*. Ta umožnuje projít jednotlivé tagy předpisu a získat z nich množiny obrazovek pro kapitoly, pořadí úkolů a jejich typ a zároveň data, která na jednotlivé obrazovky patří.

Aplikace si zavolá tento parser při jejím spuštění a dále využívá získané objekty typu *QuestionSet, QuestionSlide* atpod. ke složení obsahu kapitol. V aktivitě obsahující mapu zanese na definované GPS souřadnice body do mapy a napáruje tyto body s předvytvořenými aktivitami, které reprezentují kapitoly. Když je uživatelem spuštěna kapitola, aplikace najde ve třídě *QuestionSet* správný *QuestionSlide*. V něm projde postupně jednotlivé obrazovky, a pokud má být úkol vlastního typu, tak ho vezme připravený přímo z aplikace. Pokud se jedná o generický úkol je do aktivity vložena třída „*QuestionFragment*“. Ta v sobě obsahuje reference na všechny předpřipravené rozložení stránky a reference na do stránky umístitelné objekty. Z *QuestionSlide* je vybrán požadovaný layout a jsou do něj vloženy příslušné funkční celky.

Níže je pro lepší představu ukázána část předlohového XML. Tato ukázka obsahuje jednu kapitolu jménem „*testTask*“, která se skládá z jedné obrazovky jménem „*task01kde*“. Na té je zobrazen nadpis a úkol, ve kterém je možné vybrat více správných odpovědí z předložených možností. Hodnotou atributu „*valid*“ je určeno, zda je nabídnutá varianta správně či nikoliv.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE questionset SYSTEM "..\..\..\android_core\assets\questions.dtd">
<questionset maxTries="2">

<task name="testTask">
 <questionslide name="task01kde" title="výskyt vazby" layout="h_quest_img">
        <images>task01_bond</images>
        <questions>
               <question type="multichoice" shuffle="true" validate="true">
                   <description>
                       Kde se můžeme setkat s chemickou vazbou?
                   </description>
                   <variant valid="true">v lidském těle</variant>
                   <variant valid="true">v potravinách</variant>
                   <variant valid="true">v lécích</variant>
                   <variant valid="true">ve škole</variant>
                   <variant valid="true">v ovzduší</variant>
                   <variant valid="true">ve vesmíru</variant>
               </question>
           </questions>
        </questionslide>

</task>
</questionset>
```

Zdrojový kód 1.1 Ukázka XML předpisu

1.5 Aplikace Z Kralup za Antonínem Dvořákem

Tato mobilní aplikace určená pro zařízení s Android systémem byla realizována jako vítězný návrh v soutěži středoškolských studentů „Poznávejme Středočeský kraj“. Podklady připravili studenti z Dvořákova gymnázia SOŠE v Kralupech nad Vltavou.

Aplikace pomocí mapy a GPS navigace vede uživatele naučnou trasou od nádraží v Kralupech nad Vltavou až k zámku v Nelahozevsi. Cestou uživatel projde centrem města, přes vrch Hostibejk a část Dvořákovy stezky. Základní trasa má délku 6 km, ale existuje i další prodloužená varianta trasy a to na 9 km [1].

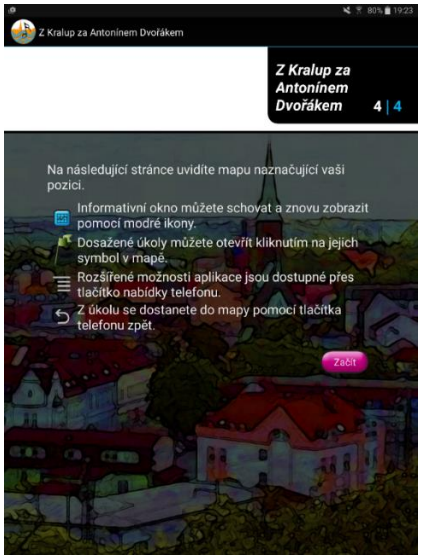
Uživatel má k dispozici offline mapu s vyznačenou trasou, na které je zobrazeno několik stanovišť. V momentě, kdy se uživatel ocitne s určitou tolerancí na GPS souřadnicích přiřazených danému úkolu, otevře se mu sekce s úkolem, informačním textem, audio nahrávkami apod., které se vztahují k lokalitě, kde se nachází.

V hlavní obrazovce aplikace (Obr. 1.10) je uživateli zobrazena mapa s vlaječkami, které jsou umístěny v oblastech, kde musí hráč plnit připravené úkoly. V horní části obrazovky je umístěna vysouvací lišta, ve které je možné se dozvědět, v jaké nadmořské výšce se zařízení nachází, jakou rychlostí se pohybuje, jakou vzdálenost od začátku uživatel ušel a jeho aktuální úspěšnost v plnění stezky. Z ostatních aplikací může mít uživatel zvyklost podobné lišty vytahovat tahem prstu směrem dolu, bohužel to není správné řešení pro tuto naučnou stezku. Lišta se zde vysouvá pomocí kliku na zobrazené tlačítko, což nemusí každého uživatele napadnout. Mapa je přehledně zpracována a zvolené vlaječky jsou pro uživatele dobrým řešením, protože jejich tyčka je umístěna do mapy s naprostou přesností.

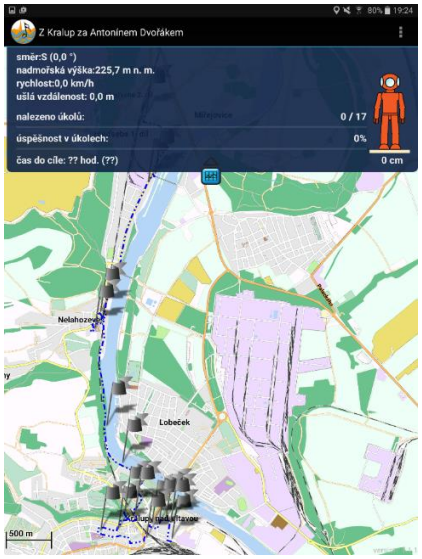
V hlavičce aplikace, která obsahuje její název, se v pravé část vyskytuje rozklikávací menu, pomocí kterého se může uživatel podívat na jízdní řády či si nechat ukázat restaurace v okolí.

Řešení aplikace:

Obsah stezky Z Kralup za Antonínem Dvořákem je generován z předpřipraveného XML předpisu. Aplikace se stará o zobrazení mapy, bodů na ní a spouštění příslušných kapitol. To, jaké obrazovky kapitola obsahuje je získáno přes knihovnu z XML. Úkoly pro uživatele jsou použity převážně z předpřipravených druhů, ale zároveň se zde objevují i obrazovky speciální, které jsou dostupné pouze v této stezce. Do předlohy jsou tak zaneseny jako „*custom*“.



Obr. 1.9 Úvodní obrazovka aplikace Z Kralup za Antonínem Dvořákem



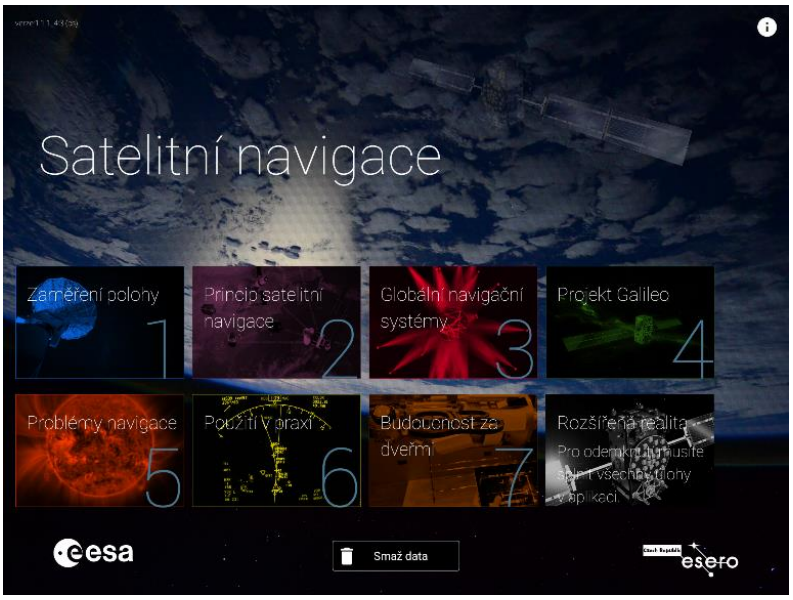
Obr. 1.10 Hlavní obrazovka aplikace Z Kralup za Antonínem Dvořákem

1.6 Aplikace Satelitní navigace

Aplikace Satelitní navigace vznikla s podporou mezinárodní agentury ESA a jejím českým partnerem agenturou ESERO. Obě organizace se zabývají vzděláváním lidí v oblasti vesmírného výzkumu a jemu přidruženým tématům.

Projekt, kterým se zabývá tato podkapitola, se snaží čtenáře seznámit s tím, jak funguje satelitní navigace. Zároveň jsou zde zmíněny různé její typy nebo upřesněny problémy, se kterým se fungování navigace musí potýkat. Nechybí zde ani praktické ukázky využití navigačních systémů v každodenním životě a případný výhled, jaké by mohly nastat změny do budoucna. Protože ESA je agentura evropská, zvýšená pozornost je věnována systému Galileo, který je provozován ze stejného kontinentu [2].

Z úvodního menu (Obr. 1.11) je možné smazat dosavadní postup, zobrazit si informace o vývoji aplikace a vstoupit do jedné ze 7 kapitol. Ty obsahují převážně informační obrazovky složené z textů a obrázků, ale na konci kapitoly je vždy připraven interaktivní úkol, za který uživatel obdrží část modelu satelitu, který si postupem aplikací skládá. Po úspěšném absolvování všech úkolů je otevřen vstup do modulu, který umožnuje zobrazení rozšířené virtuální reality.

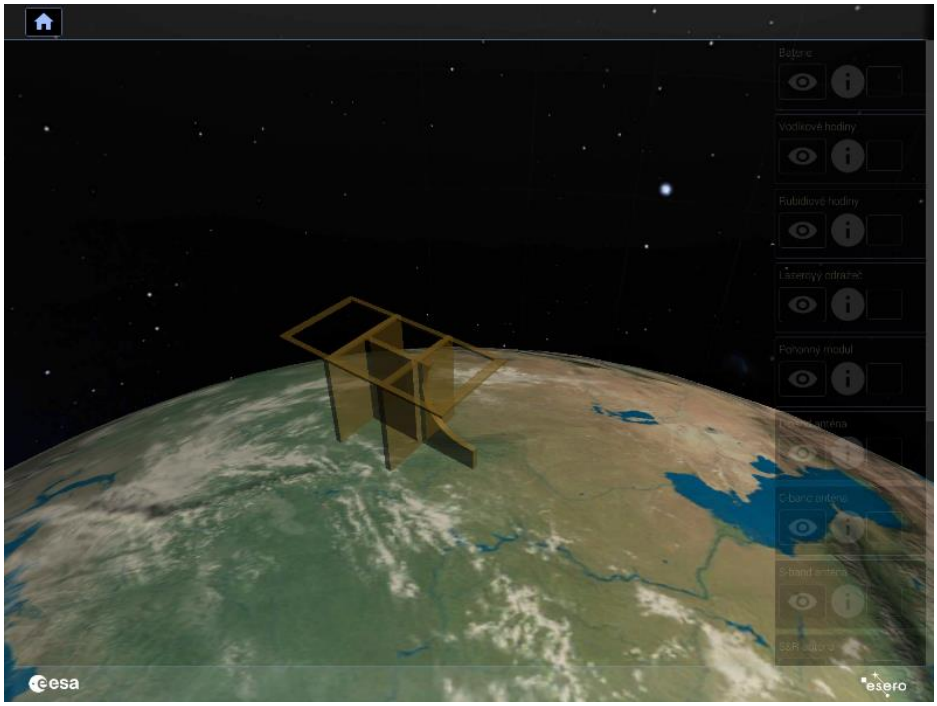


Obr. 1.11 Úvodní menu aplikace Satelitní navigace

Řešení aplikace:

Aplikace není generována dle XML předpisu, pouze využívá úkoly, které knihovna poskytuje. Pořadí a umístění úkolů do kapitol je řešeno přímo v Satelitní navigaci. Aplikace obsahuje pro každou kapitolu vlastní aktivitu, která je spouštěna z hlavního menu. V aktivitách je definován typ a pořadí jednotlivých obrazovek. Každé obrazovce je nutno předat rozdílné informace, co se týče obsahu. Například pro úkoly s přetahováním objektů na správné místo je nutné předat reference na všechny přetahovatelné prvky a specifikovat místo, kam má být objekt přesunut. Naproti tomu pro informační úkoly, které obsahují pouze obrázek s textem, stačí definovat znění zobrazeného nápisu a odkaz na obrázek, který se má do úkolu umístit.

Z aplikace je možné spustit modul, který podporuje funkce rozšířené virtuální reality. Dalším zábavným prvkem může být skládání modelu satelitu (Obr 1.12), ke kterému uživatel postupně získává různé části. Tato sekce je v aplikaci tvořena pomocí *Unity*.



Obr. 1.12 Skládání modelu satelitu v aplikaci Satelitní navigace

2 ROZŠÍŘENÍ EXISTUJÍCÍHO ŘEŠENÍ

Stávající řešení knihovny nepodporuje načítání GPS souřadnic definovaných v XML předpisu úloh. Původní řešení mělo tyto souřadnice definované v aplikaci fixně v místě tvoření bodů, které se vloží do mapy. To komplikuje práci a zhoršuje znovupoužitelnost kódu. Součástí zadání této diplomové práce je rozšířit Android knihovnu o načítání GPS z předpisu společně s úkoly. Tato kapitola popisuje způsob, jakým je požadovaná funkcionalita přidána.

2.1 Definování GPS souřadnic v XML předpisu

Nejprve je nutné určit, kde a jak má být GPS lokace v předpisu zadána. Své souřadnice potřebuje definovat každá kapitola zobrazená bodem na mapě. Kapitola je reprezentována tagem *questionslide*. Vhodné místo pro vymezení polohy je tedy uvnitř tohoto tagu. Zeměpisná délka a šířka jsou určeny v elementech <*latitude*> a <*longitude*>. Ty jsou uzavřeny do obalujícího prvku <*location*>. Níže je ukázka předlohy jedné kapitoly včetně definovaných GPS souřadic.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE questionset SYSTEM "..\..\..\android_core\assets\questions.dtd">
<questionset maxTries="2">

<task name="testTask">
 <location>
        <latitude>50.0767562</latitude>
        <longitude>14.4221933</longitude>
 </location>
 <questionslide name="task01kde" title="výskyt vazby" layout="h_quest_img">
        <images>task01_bond</images>
        <questions>
                <question type="multichoice" shuffle="true" validate="true">
                    <description>
                        Kde se můžeme setkat s chemickou vazbou?
                    </description>
                    <variant valid="true">v lidském těle</variant>
                    <variant valid="true">v potravinách</variant>
                    <variant valid="true">v lécích</variant>
                    <variant valid="true">ve škole</variant>
                    <variant valid="true">v ovzduší</variant>
                    <variant valid="true">ve vesmíru</variant>
                </question>
           </questions>
        </questionslide>

</task>
</questionset>
```

Zdrojový kód 2.1 Ukázka XML předpisu s GPS lokací

2.2 Načítání GPS souřadnic z XML předpisu

Uložení rozparsované lokace probíhá pomocí hashmapy, která využívá jako klíč název kapitoly a jako hodnotu instanci třídy *LatitudeLongitude*, která je v knihovně obsažena. Tato třída obsahuje dva atributy reprezentující zeměpisnou délku a šířku.

Pro získání potřebných dat, bylo nutné přidat do třídy *QuestionSet*, která v knihovně zajišťuje parsování XML, funkcionalitu rozpoznání nově přidaných tagů se souřadnicemi. Během parsování projíždí třída postupně všechny tagy z předpisu a pokud element rozpozná, tak si uloží patřičné informace. Je zde tedy přidáno na úroveň rozlišení tagu *questionslide* rozpoznání nového tagu <*location*>. V případě nalezení tohoto tagu je zavolána nová metoda *readLocation()*, která se do tohoto tagu zanoří a získá z něj informaci o GPS lokaci. Ta je následně pomocí názvu kapitoly z tagu <*header*> uložena do připravené hashmapy.

Pro využití v aplikaci jsou uloženy takto namapované lokace do statické třídy *ApplicationState*, která obsahuje veškerá naparsovaná data. Při samotném vkládání styčného bodu do mapy jsou pak souřadnice brány z uložené hashmapy oproti předchozímu řešení, kde byly určené fixně.

3 ANALÝZA MOŽNOSTÍ PŘEVODU KNIHOVNY

Tato kapitola obsahuje analýzu multiplatformní vývoje. Před každým vývojem multiplatformní aplikace nastává otázka, jakou cestu zvolit. Aplikaci lze vyvíjet nativně (pro každou platformu zvlášť), vyvinout projekt pomocí HTML 5 (multiplatformní – „napsat jednou a spustit kdekoliv“) či vyvinout hybridní aplikaci (viz kapitola 3.3). V roce 2013 bylo až 94% vyvíjeno nativně a pouze 6% cíleno na HTML5 [3].

3.1 Nativní řešení

Nativní vývoj znamená, že aplikace bude vyvíjena přímo pro konkrétní operační systém, pomocí nástrojů pro platformu přímo určených a to tak, aby byly využity všechny výhody a možnosti dané platformy.

V případě, že má být mobilní aplikace funkční na více mobilních platformách a je zvolen nativní vývoj, znamená to v praxi, že je třeba vyvinout aplikaci pro každý systém zvlášť, protože každý systém je svým způsobem specifický. To znamená, že vývojový tým musí ovládat různá vývojová prostředí, SDK, různé jazyky pro každou platformu. V praxi by to nejspíše znamenalo mít pro každou platformu vlastní vývojářský tým.

Zároveň vede nativní vývoj oproti ostatním způsobům především v oblasti uživatelského prožitku („*User experience*“). Z pohledu inovativnosti jednotlivých řešení je nativní vývoj nejvhodnější metodou. Apple vydává aktualizaci ročně, Android několikrát ročně.

Nativní aplikace mohou využít přímo zdroje, kterými disponuje operační systém. Myšleno například použití kamery, geolokace, animace a další.

Výhody nativního vývoje:

- Nativní uživatelské rozhraní – vývojové prostředí podporuje prvky UI přímo v telefonu, aplikace mají správný „feeling“ a zajištují nejlepší uživatelský prožitek
- Podpora aktuálního API v OS – snadné využití zabudovaných funkcí pro všemožné služby
- Výkon – aplikace je optimalizovaná přímo na danou platformu

Nevýhody nativního vývoje:

- Aplikace běží pouze na jedné platformě
- Drahý vývoj – pro vývoj na více platforem je nutné většinou využít vývojový tým pro každou platformu zvlášť, případné úpravy je nutné provést pro každou platformu separátně

Obr. 3.1 Možnosti nativního vývoje mobilních aplikací [4]

3.2 Mobilní web

Druhým způsobem, jak vyvinout projekt multiplatformní, je použít HTML5 typicky v kombinaci s kaskádovými styly CSS a skriptovacím jazykem JavaScript, případně nějakým frameworkem určeným pro použití v JavaScriptu. Tento způsob je vhodný především pro jednoduché aplikace, kde je kladen důraz na nízkou výrobní cenu.

HTML 5 má oproti nativnímu vývoji značné slabiny. Především v oblasti výkonu, horší možnosti monetizace a v nevyužití veškerých výhod cíleného systému. Také díky rozdílnosti podpory funkcí v různých prohlížečích platí v dnešní době spíše heslo „napsat jednou, optimalizovat všude“. Hlavní výhodou jsou tedy pouze nižší náklady na vývoj, což ale v tomto projektu nehraje roli. Z hlediska uživatelského prožitku je vývoj mobilního webu dostačující, ale nativní vývoj je z pohledu prožitku ještě o něco lepší, a v dnešní době je na spokojenost uživatele kladen důraz.

Co se týče optimalizování a aktualizací vývoj v HTML 5 pokulhává oproti nativnímu vývoji, kde jsou platformy udržované průběžně. Oproti tomu schválení standardu HTML 5 trvalo organizaci *The W3C* pět let [3].

Bezpečnost v HTML 5 má v mnohém slabiny. Získat a prolomit kód HTML 5 je mnohem jednodušší než kód nativní aplikace. Dále například přenos dat lze zabezpečit v obou způsobech vývoje, ale při použití HTML 5 je díky šifrování dosaženo menších rychlostí [3].

Výhody mobilního webu:

- Univerzálnost – platí zde „jednou napiš, spusť všude“, případné úpravy se dělají pouze na jednom místě, běží na více platformách
- Není nutná instalace přímo do mobilního zařízení
- Cena – levný vývoj – vyvíjí se pouze jednou i pro více platforem

Nevýhody mobilního webu:

- Horší uživatelský prožitek – mobilní weby nedokážou přinést stejný prožitek při používání aplikace jako nativní aplikace (gesta, plynulé scrollování atpod.)
- Nutné připojení k internetu pro používání aplikace
- Nenativní rozhraní – mobilní weby nepoužívají zabudované UI prvky pro danou platformu a je nutné si je zvlášť nastylovat

3.3 Hybridní aplikace

Hybridní aplikace je webová aplikace optimalizovaná pro mobilní zařízení. Bývá tedy napsána pomocí CSS, HTML a Javascriptu, a běží v kontejneru v nativním prostředí - tzv. *webview* (což je v podstatě jednodušší webový prohlížeč). Teoretickou hlavní výhodou hybridních aplikací je, že se aplikace vyvine pouze jednou a může být bez úprav spuštěna jak pod Androidem tak pod iOS nebo třeba na Windows Phone. V praxi se ale stejně musí aplikace ještě dále optimalizovat a upravovat a tak heslo čistě „napsat jednou, spustit všude“ neplatí.

Oproti mobilnímu webu umožňují hybridní aplikace (díky běhu v kontejneru) pracovat se speciálními funkcemi platformy či přistupovat k hardwaru (kamera, gesta atpod.). Tento druh aplikace je dostupný v app storech dané platformy a je ukládán přímo do zařízení [5].

Pro zvolení možnosti vývoje aplikace hybridní cestou může přispět nedostatek zdrojů či například to, že k jejich výrobě stačí umět známé front-end technologie, vývojáři nemusí znát speciální nástroje či jazyky pro konkrétní platformu a ani nepotřebují k vývoji konkrétní prostředí a SDK.

V kapitole 3.3 jsou dále blíže popsány vybrané frameworky, které se k vývoji hybridních aplikací používají.

Výhody hybridního vývoje:

- Multiplatformnost – stačí jeden vývoj pro vícero platforem najednou
- Menší nároky na programátory – HTML a JS jsou známější větší skupinou programátorů než jazyky pro nativní vývoj

Nevýhody hybridního vývoje:

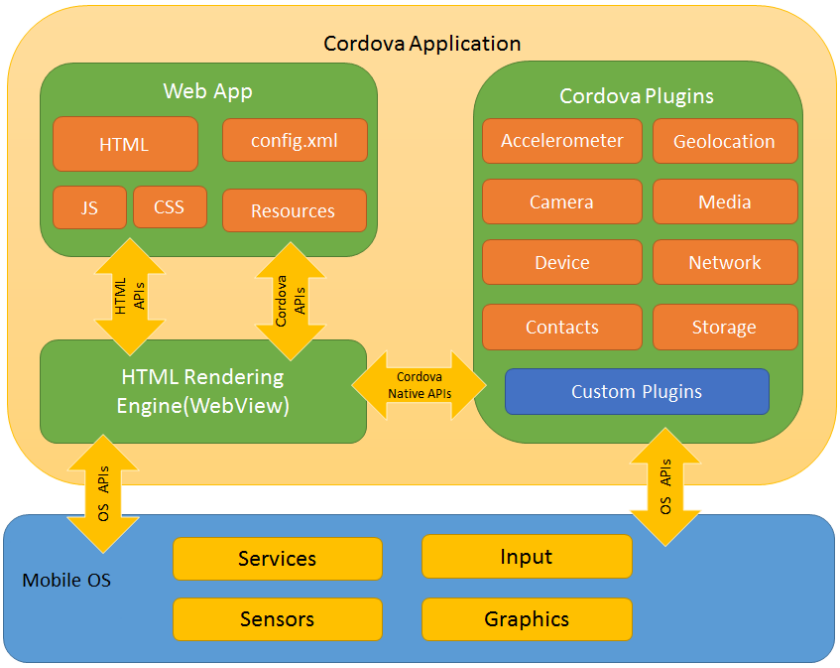
- Nenativní rozhraní – mobilní weby nepoužívají zabudované UI prvky pro danou platformu a je nutné si je zvlášť nastylovat
- Nefunkční požadavky – jedná se o webové aplikace běžící v kontejneru prohlížeče, což přináší jisté HW omezení, tato nevýhoda se snižuje se zvyšujícím se výkonem mobilních zařízení

3.3.1 Apache Cordova

V roce 2009 vytvořil startup Nitobi projekt nazvaný PhoneGap. Jednalo se o open-source API k přístupu k nativním mobilním zdrojů s cílem umožnit vývojářům vytvořit mobilní aplikaci pracující s hardwarovými funkcemi zařízení při použití standardních mobilních technologií. V roce 2011 bylo pak open-source jádro využito organizací Apache Software Foundation a přejmenováno právě na Apache Cordovu [6].

Apache Crodova je open-source framework pro vývoj mobilních aplikací. Tak jako většina cross-platform frameworků využívá webových technologií HTML 5, CSS3 a JavaScriptu. Apache cordova je vhodný pro mobilní vývojáře, kteří chtějí rozšířit aplikace na více než jednu platformu bez nutnosti implementovat aplikaci vícekrát. Zároveň díky tvorbě hybridní verze je vhodný pro webové developery, kteří mají zájem o to, mít svou aplikaci uloženou v platformových storech a nebo chtějí, aby jejich webová aplikace měla přístup i k hardwarovým funkcím zařízení jako je fotoaparát, senzory apod.

Architektura aplikace napsané v Cordově se skládá ze tří hlavních částí, které následně pomocí OS API komunikují se samotným operačním systémem platformy. Hlavní logiku řídí samotná webová aplikace, která pracuje s daty z *Resources* a nastavením z config.xml. Pomocí HTML API a API Cordovy, se pracuje s renderovacím enginem (*WebView*), který se stará o uživatelské rozhraní. Díky enginu Cordovy je možné využívat nativní API pluginy, které dále pracují s funkcemi daného zařízení (síť, media, akcelerometr apod.) [7].



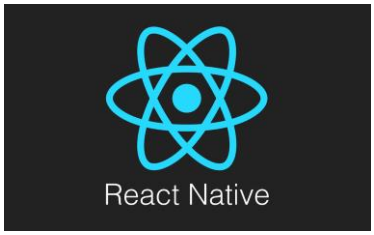
Obr. 3.2 Architektura aplikace při použití Apache Cordova [8]

3.3.2 React native

Tento open-source framework podporuje vývoj plných nativních aplikací. Vývoj s tímto frameworkem probíhá kompletně pomocí JavaScriptu a Reactu. React native je určený především pro zkušené webové vývojáře, má ale poměrně rozsáhlou nápomocnou komunitu.

V React native existuje podpora pro vývoj na iOS i Android. Tato knihovna umožňuje renderovat uživatelské rozhraní pro obě platformy. Při tvoření uživatelského rozhraní je možné využívat znovupoužitelné bloky komponent, které se skládají nativně. Komponenty, které je možné využívat při nativním vývoji, mají obdobné verze i v React native, a tak je možné dosáhnout konzistentního nativního vzhledu. Právě to je hlavní prioritou při vývoji tímto frameworkem [9].

Nicméně použití React native je vhodné zatím pouze pro úzký typ aplikací. Především pro malé aplikace o pár obrazovkách nebo jako malou funkční část uvnitř nějaké většího celku. React native je poměrně mladý framework a zatím se velmi aktivně vyvíjí a s tím souvisí i jeho nevyzrálost. Je plný chyb, které se zatím musejí poměrně často opravovat, což se děje většinou pod taktovkou komunity namísto výrobce – firmy Facebook. Samotný Facebook ve svém programu tuto vyvinutou technologii používá pouze na malém jednoduchém opakujícím se motivu [10].

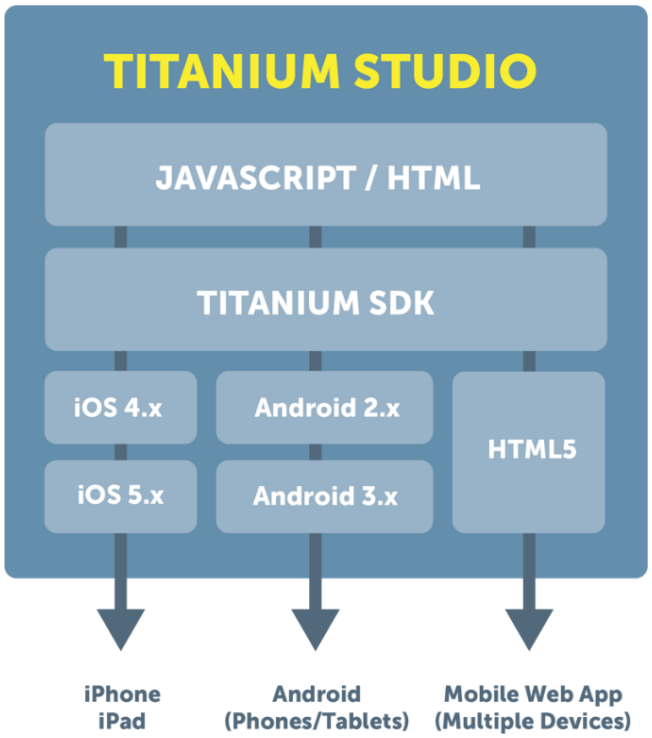


Obr. 3.3 Logo React native [11]

3.3.3 Appcelerator

Tento framework pomáhá vyvíjet nativní aplikace pomocí jediného javascriptového kódu. Podporuje snadnou a rychlou tvorbu prototypu s malými náklady na otestování uživatelského rozhraní. Obsahuje datové uložiště zvané *ArrowDB*, které umožňuje využívat datové modely bez komplikovaného počátečního nastavení. Velkou nevýhodou je, že se o frameworku uvažuje jako o plném chyb a často „zasekaném“. Typickým problémem při použití Appceleratoru je například *memory leaking*. Zároveň nemá tak kvalitní komunitní podporu [12].

Použití Appceleratoru je vhodné především z pohledu úspory zdrojů. Při vývoji nativních aplikací se v tomto framewroku často používá až 90% kódu opakovaně [13], což vede ke značně nižším nákladům - finančním i časovým.



Obr. 3.4 Struktura vývoje aplikace pomocí Titanium Appcelerator [14]

3.3.4 Ionic

Pomocí tohoto front-endového frameworku lze díky CSS vytvořit vzhled aplikace značně podobný nativnímu designu. Nejčastěji se využívá v kombinaci s AngularJS. Hlavním bonusem využití Ionicu je rozhraní příkazového řádku, které disponuje rozličnými užitečnými funkcemi včetně integrovaného emulátoru. Výhodou Ionicu je dobrá práce s předdefinovanými komponentami a především dobrá komunita. Nevýhodou je nutná znalost AngularJS pro vytvoření komplexnější aplikace [15].

Vývoj v Ionicu je pod open-source licencí. To znamená, že oproti mnohé placené konkurenci je zadarmo. Zároveň má tento framework vývojáře, kteří udržují jeho stav dle nejnovějších trendů.

Tento projekt je cílen především na vzhled a UI interakci aplikace. Není to tedy náhrada za Cordovu či PhoneGap, naopak slouží k jejich doplnění o zjednodušenou práci při vývoji front-endu.

Obr. 3.5 Logo Ionic [16]

3.3.5 OnsenUI

Open-source framework založený na Cordově, který umožňuje vývojářům vytvořit aplikaci nativního vzhledu pomocí kombinování předdefinovaných komponent. Je poměrně jednoduchý na použití a dá se zkombinovat s prací s AngluarJS. Výhodou je detailní dokumentace plná příkladů a layoutů nejčastějších typů stránek. V minulosti bylo jeho velkou nevýhodou podpora pouze iOS vzhledu. Nicméně aktuálně framework podporuje i material design, který využívá Android [15].

Znalce jQuery technologie potěší, že framework obsahuje základní jQuery komponenty.

Obr. 3.6 Logo Onsen UI [17]

3.3.6 Unity

Tento framework je v podstatě z jiné kategorie než všechny předchozí. Je ale důležité ho zmínit, protože ve svém tematickém zaměření značně dominuje. Unity 3D slouží především ke tvoření her. Jedná se o herní engine vhodný, pokud od aplikace požadujeme nadprůměrně kvalitní grafiku a herní zážitek.

Použití tohoto cross-platform vývojového nástroje je mnohem komplexnější než čistě překlad mezi platformami. Vyvíjí se ve speciálním jazyku UnityScript či v C#. Následně je možné tento kód exportovat jako hru na 17 různých platforem včetně iOS, Android, Windows nebo třeba Playstation. Unity také podporuje následně snadnější distribuci mezi patřičnými app stores.

Obr. 3.7 Logo Unity [18]

3.4 Zvolený způsob vývoje

Úkolem této diplomové práce, je realizace již existujícího řešení aplikací z OS Android na platformu iOS. Při volbě z možností typu vývoje je nutné tento fakt vzít v potaz. Při volbě implementace mobilní aplikace je nutné posoudit, k jakému účelu má aplikace sloužit, jak velký a zkušený bude vývojářský tým, případně časové náklady. Pro každou aplikaci je tedy vhodný jiný přístup.

Možnost napsání čistě webové aplikace slouží především k jednodušším kancelářským aplikacím, které nepotřebují využívat služby operačního systému. Zároveň je u nich nutné, aby běžely ve webovém prohlížeči a aplikace by pak nebyla dostupná ke stažení v patřičných app storech.

Výhodou zpracování hybridní aplikace je především fakt „napiš jednou, spusť kdekoliv“. V tomto případě však tato výhoda odpadá, protože již existuje napsané nativní řešení pro Android, které bude nutné jen vylepšit. Zároveň se jedná o aplikaci, která má sloužit především ke zprostředkování zážitku - zábavy - uživateli. A proto by měl být jeho prožitek na prvním místě. Z hlediska prožitku vyhrávají z uvedených možností nativní aplikace. Přestože se hybridní aplikace snaží dosáhnout srovnatelného zážitku s aplikací nativní, často narazí na drobné problémy, které uživateli zážitek kazí. Například práce se scrollováním, chování klávesnice, navigace v aplikaci a podobně. Zároveň nejsou schopné hybridní aplikace využít plně efektivně zdroje zařízení, čehož následky mohou v uživateli zanechat také negativní dojem. Rendrování moderního HTML a CSS využívající pokročilé funkce jako gradient aj., kladou velké nároky na CPU a GPU zařízení [19]. Aplikace založené na HTML spotřebovávají znatelně více baterie než nativní aplikace, což při použití v exteriéru s využitím funkce GPS by mohlo mít v této aplikaci následky.

Při uvážení těchto faktů vychází pro toto konkrétní řešení nejvýhodněji nativní způsob vývoje.

4 NATIVNÍ VÝVOJ IOS APLIKACÍ

Z analýzy v kapitole 3 vyšla nejlépe možnost nativního vývoje aplikace pro operační systém iOS. Kapitola 4 stručně popisuje nástroje a postupy, bez kterých se nativní vývoj pro iOS neobejde. Jedná se například o striktně určené vývojové prostředí, které běží pouze na MacOS, jazyk, který byl vyvinutý speciálně pro nativní vývoj společností Apple nebo množina dostupný frameworků, které významně ulehčují vývojáři práci.

4.1 XCode

Nejdůležitějším nezbytným nástrojem pro nativní vývoj na iOS je vývojové prostředí XCode. Stejně jako samotný operační systém je XCode vyvinut společností Apple. Je dostupný ke stáhnutí zdarma na App Store. Hlavním omezením vývoje pomocí XCode je nutnost pracovat výhradně na Mac OS. Na ostatních operačních systémech je nutné zprovozňovat virtuální stroj, na kterém potřebný operační systém poběží. Svým způsobem si tak Apple udržuje svůj monopol a chrání svou značku, že ten, kdo chce vyvíjet na jejich mobilní platformu, musí i vývoj provést na zařízení od této společnost.

Aktuální nejnovější verze XCode 8 obsahuje vše potřebné pro vývoj na všechny Apple zařízení. Na telefony iPhone, tablety iPad, počítače Mac, Apple Watch i televize Apple TV.

Důležitou část při tvorbě aplikací zde tvoří tzv. „*Interface builder*“. V tomto nástroji se definují veškerá uživatelská rozhraní, kterými bude následně aplikace disponovat. V nové verzi byla tato funkce značně modifikována pro lepší ovladatelnost a především vyšší rychlost. Existuje zde možnost prohlídnout navržené rozhraní na různých zařízení s volbou orientace displeje [20].

XCode podporuje vývoj v mnoha programovacích jazycích. Konkrétně se jedná o jazyky C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, Rez a nově nejaktuálnější jazyk pro vývoj na iOS – Swift.

4.2 iOS Simulátor

Určitou variantou emulátoru pro spuštění vyvíjené aplikace na virtuálním stroji je nástroj iOS simulátor. Simulátor umožnuje spustit velké množství variant simulující různé reálné zařízení od společnosti Apple. Samotné testování aplikace by mělo proběhnout před vydáním i na reálných zařízeních, ale je možné do určité míry chování aplikace odzkoušet i jen na takto vytvořených virtuálních strojích.

Velkou výhodou této funkce je možnost simulovat i akce, které jsou v reálném zařízení prováděny fyzicky. Je možné simulovat gesto zaklepání, otočení zařízení. Zajímavými funkcemi jsou například manuální nastavení simulované geografické polohy nebo simulace nedostatku paměti.

Při používání simulátoru je důležité mít na paměti, že přestože umožňuje vyzkoušet běh aplikací i bez reálných strojů, jeho chování se liší. Je rychlejší a má více paměti, protože využívá paměti počítače, na kterém je spuštěný [21].

4.3 Swift

Společnost Apple si uchovává kontrolu nad veškerým vývojem a přístupem ke svým zařízením. Pro účely implementace aplikací na iOS nově prosazuje vývoj ve svém vlastním programovacím jazyku Swift. Aktuálně nejnovější verzí je Swift 3.

Jedná se o open-source jazyk, který má být novější variantou k vývoji v Objective-C, ale měl by umožňovat méně chyb vinou programátora. Swift je kompilován pomocí LLVM a je možné, aby se vyskytoval ve stejném programu společně s jazyky C, Objective-C nebo C++. Apple Swift označuje jako moderní programovací jazyk, který je snadný k použití, přehledný a vhodný pro začátečníky.

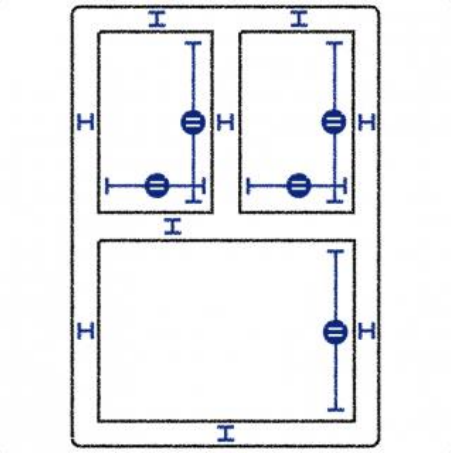
4.4 Autolayout

Vývoj pro mobilní zařízení značně komplikuje jejich rozmanitost. Existuje mnoho různých velikostí displejů a na nich různé rozlišení. Proto nastává otázka, jak určit rozložení prvků uživatelského rozhraní na obrazovce. Pro vývoj aplikací na OS Android rozdělil Google existující displeje do několika kategorií. Pro optimální funkčnost na všech zařízení je nutné pro každou kategorii zvlášť uzpůsobit XML layout předpis, který rozhraní definuje. To je časově náročné a poměrně komplikované, obzvlášť pokud si chce vývojář svou práci otestovat i na reálném stroji. Zařízení, která obsahují iOS mají také různé velikosti a tak se společnost Apple pokusila návrh rozhraní vývojářům zjednodušit tak, aby stačilo prvky rozmístit pouze jedenkrát a aplikace běžela použitelným způsobem rovnou na většině zařízení. K tomu slouží tvorba rozhraní pomocí autolayoutu.

Velikost zobrazovací plochy se liší nejenom druhem hardwaru, na kterým je aplikace spuštěna, ale rozhraní se musí uzpůsobit i v případě, že uživatel zařízení otočí (na výšku / na šířku). Autolayout je navržen i pro tento případ. Při použití této formy tvorby rozhraní je velikost a pozice v hierarchii objektů na obrazovce vypočítávána dynamicky na základě omezení vůči ostatním prvkům či celé obrazovce. Omezení může být například to, že má být objekt umístěn vertikálně nebo horizontálně doprostřed obrazovky či jiného objektu [22].

Dalším faktorem pro umístění může být relativní pozice vůči jinému objektu. Je možné nastavit například tlačítku, aby bylo 10 bodů pod horním okrajem konkrétního obrázku. V případě, že chceme mít nějaké objekty stejně velké, dá se jim nastavit omezení na stejnou šířku či výšku (Obr. 4.1). Každému omezení jde zároveň nastavit váhu, s jakou má platit. Takže když se nastaví, že text má být stejně široký jako obrázek s váhou 0.5, tak objekty nebudou mít rozměry stejně, ale uzpůsobí se zvolenému kvantifikátoru.

Autolayout je mocný nástroj, který šetří čas a práci vývojářů. Zároveň je ale náročný na přesné specifikování podmínek, které pro objekt platí, protože musí být pro OS naprosto jednoznačné, jak se má prvek na obrazovku umístit.



Obr. 4.1 Obrázek znázorňující omezení při použití autolayoutu [23]

4.5 CocoaPods

Ve světě informačních technologií se opakovaně objevují navzájem podobné problémy nebo je nutné využít navzájem podobné funkcionality. Je velmi pravděpodobné, že problém, na který vývojář narazí, už řešil někdo před ním. Je dobré se řídit heslem, že není třeba znovu vytvářet kolo. Pro vzájemné sdílení svých řešení obecně fungují frameworky a knihovny. Aby bylo snadné znovupoužitelné implementace vyhledat, vznikají různí manažeři závislostí. Pro projekty psané v jazyku Java je notoricky známý manažer Maven. Na podobném principu pro iOS vývoj funguje takzvaný Cocoapods.

Cocoapods je manažer závislostí pro Swift a Objective-C projekty. Aktuálně (2017) obsahuje přes 32 tisíc knihoven a využívá ho přes 2.1 milionů aplikací [24]. Cocoapods je tvořen pomocí Ruby a je pomocí jeho i instalovatelný, protože Ruby je defaultně dostupný v Mac OS. Použití Podů je opravdu snadné a při správném definování závislostí lze využívat jejich funkcionalit v podstatě ihned. V případě, že je nutné v aplikaci použít funkčnost, u které se dá odhadnout, že už jí někdo v minulosti používal, je vysoce pravděpodobné, že se dá dohledat použitelný Pod namísto náročného vývoje od začátku.

Postup pro použití Cocoapods

1) Pro používání Cocoapods je nutné ho nejprve na stroj, na kterém se aplikace vyvíjí, nainstalovat. To je díky linuxovému prostředí v Mac OS poměrně jednoduché a učiní se tak pomocí příkazu *sudo gem install cocoapods*,
2) Na stránce *cocoapods.org* se vyhledá Pod, který má být v aplikaci použit.
3) U každého vyhledaného *Podu* bývá název, přes který se vyhledává závislost a aktuální verze, ve které je *Pod* vyvinutý,
4) Do vytvářeného projektu je nutné přidat tzv. *Podfile*, který obsahuje seznam všech Podů, které má aplikace využívat. Příklad obsahu takového *Podfile* je uveden ve zdrojovém kódu 4.1. Tento soubor je možné nechat vytvořit s předdefinovanou šablonou pomocí příkazu *pod init* zavolaném v adresáři s cíleným projektem.
5) Instalace definovaných závislostí se pak již provede jen pomocí příkazu *pod install* uvnitř adresáře s cíleným projektem.
6) Důležité je pro správnou funkčnost vývoje s Pody, spouštět vývojové prostředí XCode pomocí podem vygenerovaného souboru *NazevSoubor.xcworkspace* namísto původního *NazevSouboru.xcodeproj*
7) Pro použití vybraného Podu už ho jen stačí pomocí názvu importovat do souboru s třídou, kde se má Pod využívat.

```
platform :ios, '8.0'
use_frameworks!

target 'MyApp' do
        pod 'AFNetworking', '~> 2.6'
        pod 'ORStackView', '~> 3.0'
        pod 'SwiftyJSON', '~> 2.3'
end
```

Zdrojový kód 4.1 Ukázka definování závislostí v Podfile [25]

4.6 Struktura projektu

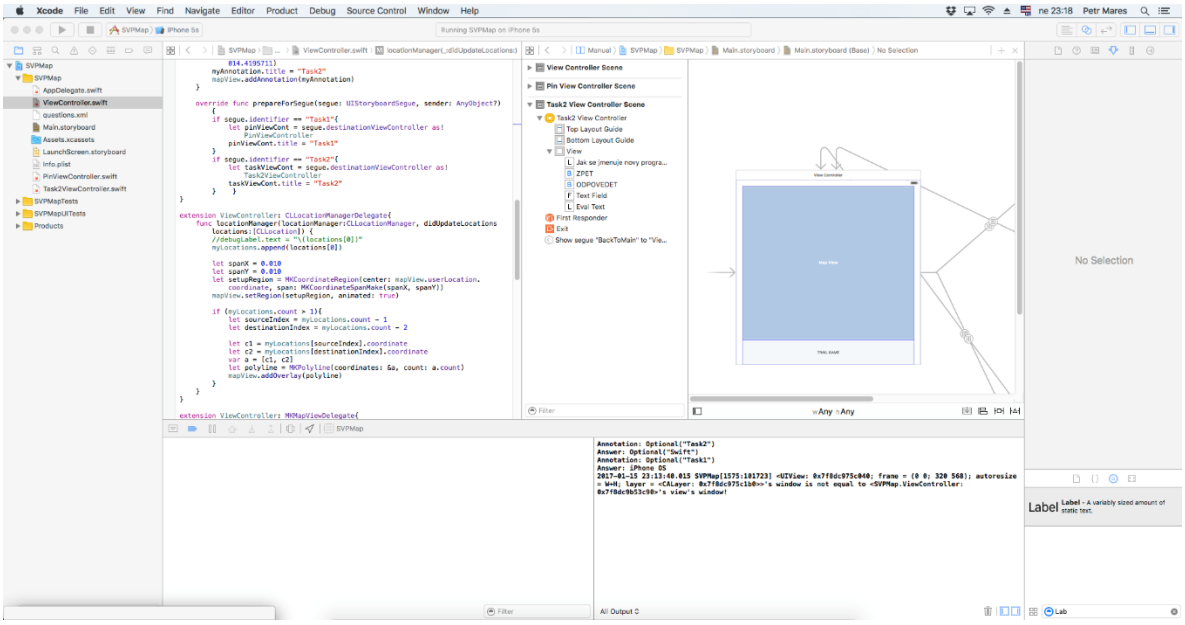
Adresářová struktura projektu je tvořena z modulů, kterým se říká „*target*“. Defaultně se jedná o *target* aplikace jako takové, *target* pro unit testování a *target* pro testování uživatelského rozhraní. V kořenovém adresáři aplikace má každý *target* vlastní adresář a dále je zde soubor NazevProjektu.xcodeproj, pomocí kterého je projekt definován pro vývojové prostředí.

Hlavní třída definující chování aplikace se jmenuje *AppDelegate.swift*. Je označena anotací *@UIApplicationMain* a dědí od tříd *UIResponder* a *UIApplicationDelegate*. Je zde umístěna proměnná window typu *UIWindow*, která reprezentuje kontejner, ve kterém spuštěná aplikace poběží. Třída *AppDelegate* slouží k definování chování aplikace po startu, když aplikace přejde z active stavu do inactive, když je aplikace předána do pozadí nebo když je znovu probuzena k činnosti či naopak ukončena.

Pro definování vzhledu aplikace je důležitý soubor *Main.storyboard*. S tímto souborem se v Xcode pracuje pomocí *Interface builderu* (dále IB) a slouží k návrhu rozhraní oken aplikace a přechodů mezi nimi. Navrhuje se v něm, na jaké obrazovce má aplikace začít (určeno pomocí přetahovatelné velké šipky na pracovní ploše), a do jakých dalších z první obrazovky lze přejit. V levé části okna IB je výpis všech navrhnutých oken společně s hierarchicky uspořádanými objekty daného okna. IB slouží k navržení rozložení jednotlivých prvků, jejich seskupování nebo testování vzhledu pro různá zařízení. Dále se zde dají definovat přechody mezi okny aplikace.

Veškerá zobrazená funkčnost aplikace se skládá z hierarchicky do sebe vnořených elementů. Jedním z nejvýše postavených objektů je již zmíněné *Window*, do kterého se umísťuje celá funkčnost aplikace. Funkčnost je z pravidla složena z určitého množství obrazovek.

Každá obrazovka aplikace musí mít definovaný svůj vzhled a logiku, kterou se obrazovka řídí. Základní stavební kámen okna je prvek *ViewController*. Ten reprezentuje jeden větší funkční celek aplikace. Zjednodušeně se dá říct, že jeden ViewController představuje jednu obrazovku aplikace. Ta se však může skládat ze široké škály velkého množství prvků. Ke každému *ViewControlleru* by pak měla být přiřazena třída, která bude dědit od *UIViewController*. V ní je určena logika, s jakou má controller fungovat. Obdobně tak platí pro každého potomka prvku uživatelského rozhraní, který do controlleru patří, ať už se jedná o předem integrovanou funkcionalitu (například třídou UIButton), a nebo vlastní nově vytvořenou, která od takto integrované třídy podědí.



Obr. 4.2 Ukázka aplikace otevřené v XCode

5 NÁVRH ŘEŠENÍ

Tato kapitola obsahuje popis převodu logiky a struktury do nové knihovny. Detaily funkčnosti převedených dílčích částí jsou blíže popsány v kapitole 6.

5.1 Návrh převodu funkcionalit

Převedená knihovna by po vzoru existujícího řešení měla poskytovat možnost vytváření úkolů z předem definovaného XML předpisu, funkcionalitu namapování předlohy na předem předpřipravené Swift objekty, ze kterých bude možné dílčí částí aplikace skládat a zároveň možnost použití pouze připravených úkolů, které půjdou do vytvářené aplikace vložit na libovolné místo.

Pro parsování předpisu bude knihovna využívat stažené závislosti na Podu AEXML. Ten umožňuje ze souboru XML formátu číst jednotlivé tagy, jejich potomky, atributy, hodnoty. Takto získaná data převede na vytvořené Swift objekty, se kterými budou moci dále aplikace pracovat bez nutnosti zkoumání XML předlohy.

Základní UI prvek, který se na obrazovce dá zobrazit je prvek typu *UIView*. Vytvořené úkoly tedy budou reprezentovány právě jedním objektem *UIView*. Každý *UIView* objekt by měl mít k sobě příslušnou Swift třídu, která definuje logiku, jak se má objekt chovat. Vzhled úkolů, které se nebudou tvořit dynamicky, bude definován pomocí souborů typu *.xib. Ty se vytvářejí podobně jako *Main.storyboard* pomocí IB. V případě, že je třeba nějaká akce s objektem v nich umístěným (například přiřadit text do prvku *UITextView* či určit chování tlačítko při kliknutí), tak je nutné spojit objekt v XIB souboru s outletem, se kterým bude pracovat přiřazená Swift třída. *Outlet* je proměnná, která funguje jako reference ke spojení s graficky zobrazeným objektem. Díky *outletu* je pak možné objektu například měnit velikost či pozadí, anebo objekt schovat [26].

Knihovna také bude obsahovat objekty, které vytvořené obrazovky budou využívat - konkrétně se jedná o objekty *CheckBox* a *ToggleWithLabel*. *CheckBox* funguje na stejném principu jako stejně pojmenovaný prvek v jiných jazycích, ale nativně není v iOS umístěn a bylo proto třeba ho zvlášť vytvořit. Slouží jako tlačítko, které je možné kliknutím zaškrtnout a dalším kliknutím přivézt zpět do původního stavu. *ToggleWithLabel* je *UIView* objekt, který ve svém XIB souboru obsahuje popisek a dvě tlačítka. Kliknutím na jednotlivá tlačítka se volí, které z nich se má vybrat. Tento objekt slouží například pro otázky, kdy je třeba volbou mezi ANO / NE určit, zda je zobrazený popisek pravdivý či nikoliv.

Pro některé funkce, které budou potřeba ke správnému fungování aplikací, na kterých se má ověřit funkčnost převedené knihovny, zde bude vytvořeny soubory *StringStyler* a *ToastExtension*. Budou umístěny do knihovny místo do aplikací, protože je možné, že bude vhodné využít jejich funkčnost i v dalších vytvářených aplikacích. V iOS není zanesena funkčnost, které je docíleno v OS Android pomocí tzv. Toastů. Jedná se o krátké upozornění na obrazovce, které je zobrazeno na pár vteřin a následně opět zmizí. Tato funkcionalita bude k dispozici pomocí *ToastExtension*, který doplňuje obecnou třídu *UIViewController* o možnost používat podobné sdělení jako je na OS Android.

V aplikacích je používán postup, kdy je zobrazený text napsán v HTML formátu a OS Android ho umí zobrazit ve stejné podobě jako by se stalo po přeložení webovým prohlížečem. Této funkčnosti nelze na iOS docílit podobně jednoduchým způsobem. V knihovně bude tedy vytvořena třída *StringStyler*, které se předloží text v HTML formátu a ona ho zbaví HTML tagů a vrátí objekt typu *NSAttributedString*, který bude přeformátovaný do podobného vzhledu, kterého by bylo docíleno HTML kódem.

5.2 Návrh mapování tříd na XML prvky

Pro omezení nutnosti zkoumat XML předpis za běhu programu bude vhodné data získaná parsováním uložit do Swift objektů, se kterými bude knihovna a odvozené aplikace pracovat. Níže v této podkapitole je tento převod znázorněn pomocí tabulek (Tab. 5.1 a Tab. 5.2), která obsahuje název XML tagu, třídu do které je informace převedena a atribut, pod kterým je v ní informace dostupná. Následně je zde umístěn diagram (Obr 5.1), jak jsou na sobě namapované objekty vzájemně závislé. Kapitola 5.2 vychází z popisu Android knihovny v kapitole 1.

Mapování bude realizováno třídou *QuestionParser*, která bude využívat Pod *AEXML*. Detailnější popis převodu obsahuje kapitola 6.3.

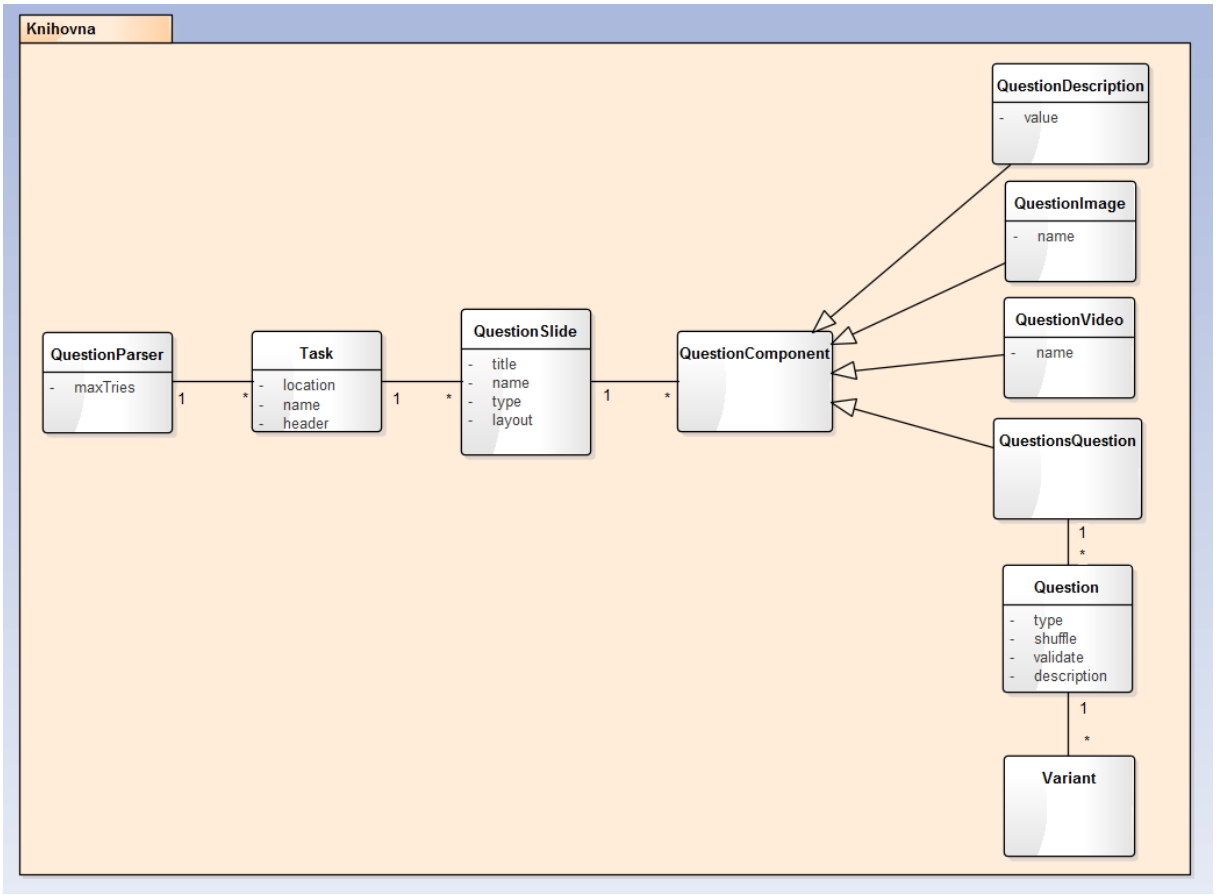
XML tag	Třída	Atribut
questionset	QuestionParser	pouze kořenový tag
task	QuestionParser	[Task] tasks
location	Task	[Double] location
questionslide	Task	[QuestionSlide] slides
header	Task	String? header
description	QuestionDescription	String? value
images	QuestionImage	[String] name
video	QuestionVideo	[String] name
questions	QuestionSlide	QuestionQuestions component
question	QuestionQuestions	[Question] values
variant	Question	[String] variants

Tab. 5.1 Tabulka návrhu převodu XML tagů do Swift objektů

XML tag	XML atribut	Třída	Atribut
questionset	maxTries	QuestionParser	maxtries
task	name	Task	name
questionslide	layout	QuestionSlide	layout
	name	QuestionSlide	name
	title	QuestionSlide	title
	type	QuestionSlide	type
question	shuffle	Question	shuffle
	type	Question	type
	validate	Question	validate
	precision	Question	precision
	buttonsOrientation	Question	buttonsOrientation
	align	Question	align
variant	valid	Question	answer

Tab. 5.2 Tabulka návrhu převodu XML atributů do Swift objektů

Pro správnou funkcionalitu dynamického tvoření obrazovek je nutné dodržovat pořadí, ve kterém jsou prvky v XML předpisu uvedeny. Aby bylo možné toto dodržet, budou prvky, které je možné přidat na obrazovku v různém pořadí realizovány jako potomci třídy *QuestionComponent*. Uvnitř třídy *QuestionSlide* je pak drženo pole těchto komponent a z něho jsou prvky přidávány na obrazovku s respektováním původního pořadí. I to je znázorněno v diagramu závislostí namapovaných tříd (Obr 5.1).



Obr. 5.1 Diagram návrhu tříd vzniklých z XML předlohy

Druh úkolu	Název UIView
Singlechoice	SinglechoiceQuestion
Multichoice	MultichoiceQuestion
Intervalquestion	IntervalQuestion
FilltextQuestion	FilltextQuestion
NumberQuestion	NumberQuestion
ToggleButtonsGrid	ToggleButtonsQuestion
DragDrop	DragDropToLineSlide
	DragDropToMiddleSlide

Tab. 5.3 Tabulka návrhu mapování druhu úkolu na vytvořené UIView v knihovně

6 IMPLEMENTACE KNIHOVNY

V této kapitole je popsáno, jak byla knihovna převedena nativní vývojem na iOS. Nejprve je popsáno, jak knihovna funguje, z jakých komponent se skládá a jakých využívá závislostí. Je zde popsána struktura projektu a detailněji popsán způsob, jakým je XML předpis převeden na Swift objekty pro jejich další použití aplikacemi. V kapitole 6.4 je výčet naimplementovaných úkolů, popsána jejich logika, požadavky na jejich funkčnost a ukázky z použití v reálných aplikacích.

Aplikace, které knihovnu budou používat, musí mít možnost na určená místa vložit svůj vlastní úkol, který není v knihovně k dispozici. Jak této možnosti dosáhnout je popsáno v části 6.5. Zároveň je vhodné, aby knihovna umožňovala přidání nových typů znovupoužitelných úkolů. Na závěr kapitoly je tedy popsán postup, jak do knihovny integrovat nový druh úkolu.

6.1 Popis funkčnosti

Objekty, které jsou v knihovně umístěné, by se dalo rozdělit do čtyřech kategorií. Jsou zde úkoly, které se generují jako *UIView* a je možné je vložit libovolně do aplikací, aniž by bylo nutné obsah generovat z předpisu. Druhým typem jsou generické úkoly, se kterými se počítá při tvorbě z XML. Dále jsou zde objekty s minimální logikou, které nejsou v iOS integrovány (např. *CheckBox*). A poslední kategorii tvoří utility na formátování textů, zobrazení hlášek či předpis, který bude umožňovat *ViewControllerům* zobrazení tzv. *modalu* přes celou obrazovku.

Knihovna využívá aktuálně tří Podů. Jedná se konkrétně o *AEXML*, *PureLayout* a *DLRadioButton*. První zmíněný Pod je určen pro snadnější procházení souboru XML formátu, jak již bylo opakovaně zmíněno v předchozích kapitolách.

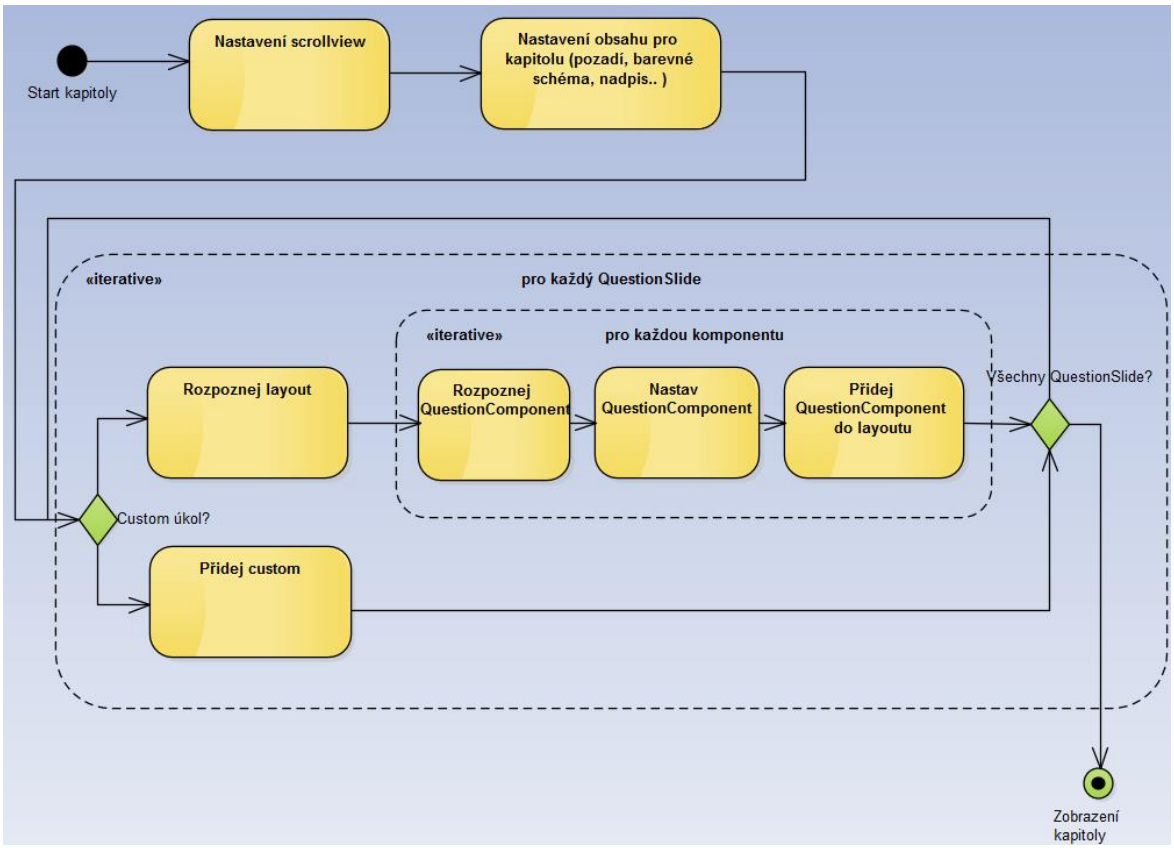
PureLayout je Pod, který slouží k jednoduššímu definování omezení, které se aplikují na prvky na obrazovce při tvoření rozhraní pomocí autolayoutu. V minulosti se vyvíjelo na cílovou platformu pouze v jazyce Objective-C a v aktuální době nejsou ještě úplně všechny funkcionality převedeny na jazyk Swift. Jedním z těchto případů je například využití *NSLayoutConstraints* třídy, jejíž účelem je definovat omezení pro *UIView*. Při práci s tímto objektem se jedná o pseudokombinaci obou jazyků a tak pro Swift vývojáře může být tento přístup lehce komplikovaný. Manuální nastavení autolayout omezení je oproti práci v IB mnohem složitější obecně, a tak je vhodné pro usnadnění práce použít nějaký Pod, který k tomuto problému přistupuje jednodušeji.

Mezi nativními prvky uživatelského rozhraní chybí i objekt *RadioButton*, který je známý třeba z aplikací psaných v HTML či pro Android. Pro dosažení stejné funkcionality používá knihovna Pod *DLRadioButton*, který se chová jako instance *UIButton* rozšířená o speciální funkce. Tento typ tlačítka se využívá především v úkole, kde uživatel volí jednu správnou odpověď z více možností a je nutné kliknutím přepínat mezi několika tlačítky najednou.

Použití knihovny v aplikacích

Negenerované úkoly jsou typu *UIView* a tak je možné je do aplikace umístit kamkoliv, kam je možné objekt na obrazovku dát. Typickým použitím je vložení několika úkolů vedle sebe do *UIScrollView* se zapnutým atributem *isPagingEnabled*. Tak se dá mezi jednotlivými úkoly přeskakovat pomocí gesta s*wipe* a scrollview se vždy zastaví na konkrétní stránce. Pro tento přístup je důležité nastavit ve *ViewControlleru* aplikace, kde je scrollview umístěn, správně jeho *frame* a *contentSize*. Frame je velikost kontejneru, ve kterém bude scrollview umístěno. Ovšem obsah uvnitř něj může mít „teoreticky" velikost nekonečnou a je nezávislá na velikosti framu. Pro uvedený případ použití se nastaví contentSize tak, aby jeho výška stejně velká jako výška framu a šířka obsahu je dána násobkem počtu obrazovek se šířkou framu. Neboli to v důsledku znamená, že jako obsah se do scrollview umístí všechny obrazovky horizontálně vedle sebe a bude se mezi nimi přejíždět.

Tento postup je nutný i pro případ, že jsou úkoly generovány dynamicky dle předpisu. Rozdílný je ale způsob, jakým se scrollview úkoly naplní. Příklad postupu, jak je vytvořen obsah kapitoly dynamicky v aplikaci Z Kralup za Antonínem Dvořákem, je znázorněn diagramem na obr. 6.1.



Obr. 6.1 Diagram vytvoření obsahu kapitoly z XML předpisu

6.2 Struktura knihovny

V kořenovém adresáři targetu knihovny se nachází hlavičkový soubor *ios_core.h* a soubor *info.plist* obsahující nastavení. Dále je zde Podfile obsahující definované závislosti z Cocoapods a *Assets.xcassets*. Assets je soubor, který funguje jako katalog obrázkových zdrojů, které knihovna využívá. Existují různé druhy displejů na mobilních zařízeních, a přestože autolayout je této rozmanitosti uzpůsoben, pro správné zobrazení je vhodné mít připravené i různě velké obrázky, tak aby na každém displeji byl zobrazený obrázek ostrý a správně velký. Proto se využívá tento katalog, kde se pod jeden název obrázku mohou určit varianty různě velké na různé displeje. Knihovna obsahuje obrázky pro vzhled tlačítka vyhodnotit nebo třeba klikatelnou část objektu CheckBox. Obrázky, které jsou do katalogu umisťovány, patří do skupiny souborů v projektu nazvané *res*.

Zbytek souborů je v projektu rozdělen do skupin dle toho, o jaký druh souboru se jedná. Ve skupině *utils* je umístěný *StringStyler*, *ChapterProtocol* a *ToastExtension* (viz kapitola 5.1). Druhá skupina *parser* obsahuje soubory, které jsou spojeny s parsovaným XML. Tzn. *QuestionParser*, třídy, na které se mapují XML tagy, ale také soubory, které tvoří layout při dynamickém skládání úkolu.

Do skupiny *views* by se měly umísťovat všechny objekty, které jsou následně využívány při tvorbě úkolů a nejsou to nativní komponenty v iOS. Aktuálně je zde umístěn *CheckBox* a *ToggleWithLabel*. Poslední skupinou jsou úkoly, které je možné využívat bez předpisu a jejich soubory jsou ve skupině *tasks*.

6.3 Parsování XML předpisu

Převod informací z XML předpisu do Swift objektů obstarává třída *QuestionParser*. Ta má importovaný Pod AEXML, který umožňuje snadnější procházení souboru ve XML formátu. Aplikace by si měla od této třídy vytvořit instanci pouze jednou při jejím startu a následně si získaná data udržovat již v paměti. Aplikace předá *QuestionParseru* předpis jako parametr *xmlDocument* v konstruktoru. Tento parametr je typu *AEXMLDocument*. Vytvořená instance obsahuje metody *getTasks()* a *getTries(),* které vrací hodnoty získané z předloženého soboru. Funkce *getTasks()* vrací pole objektů typu *Task*, které obsahují všechna další data z XML. Funkce *getTries()* vrací hodnotu typu *Int*, která má význam maximálního počtu pokusů odpovědět na zobrazené úkoly.

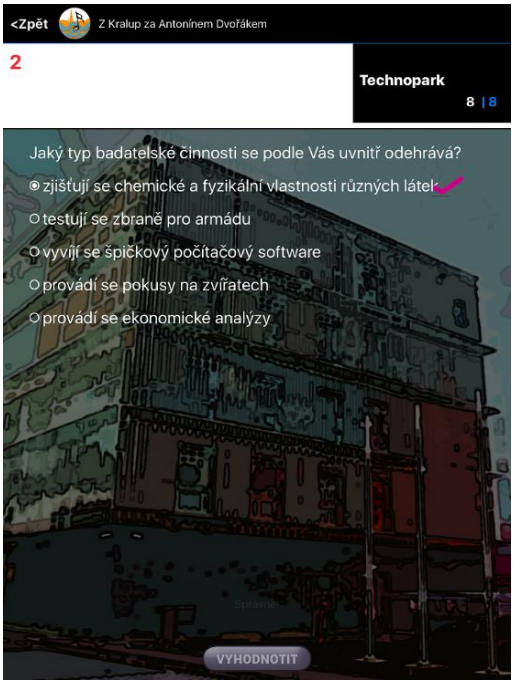
QuestionParser nejprve získá hodnotu *maxTries*. Dále pro každý z přímých potomků kořenového tagu (což jsou tagy *task*) vytvoří instanci třídy *Task* a přidá jí do pole *tasks*. Poté této instanci přiřadí zjištěnou lokaci a název. Každý *task* obsahuje elementy *QuestionSlide*. Pro každý z těchto prvků je vytvořena instance *QuestionSlide* a přidaná do pole *slides* v instanci *task*. Pro každý vnořený prvek tagu *questionslide* jsou získány jeho atributy (name, type, layout, …) a zkoumá se, jaké potomky *questionslide* obsahuje. Ty jsou přidávány jako instance *QuestionComponent* do pole *components*. V případě, že takto nalezený potomek je tag *questions*, projde parser vnořené tagy a pro každý z těchto tagů typu *question* uloží jeho atributy (type, shuffle, validate, …) a jemu vnořené tagy typu *variant,* které značí možné odpovědi v úkolu.

6.4 Znovupoužitelné úkoly

Tato část práce popisuje výčet úkolů, které lze z knihovny použít. U každého úkolu je popsáno, za jakých objektů se reálně *UIView* skládá, jaká ho obsluhuje logika. U některých úkolů je napsáno, jaké informace je mu třeba k běhu dodat. Typicky se jedná o definování obsahu u úkolů, které obsahují prázdné objekty bez obsahu. Následuje u každého úkolu ukázka, jak vypadá úkol po použití v aplikaci již implementované na iOS platformě.

6.4.1 SinglechoiceQuestion

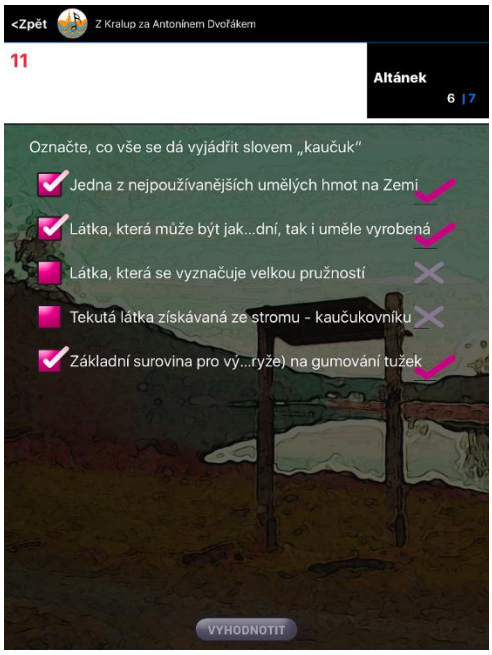
Jedná se o úkol, kde je uživateli zobrazen výčet možných odpovědí a on musí vybrat právě jednu správnou. Tento úkol je tvořen dynamicky a je mu třeba do konstruktoru poskytnout objekt typu *Question*. Z něho si vybere hodnoty nabrané z tagu *variant* a ke každé je do layoutu dynamicky přidán *DLRadioButon*. Pro vyhodnocení úkolu projde program radiobuttony a prozkoumá, zde je vybrán prvek na stejném indexu, jako měl tag *variant* v atributu *valid* hodnotu *true*.



Obr. 6.2 Ukázka úkolu Singlechoice

6.4.2 MultichoiceQuestion

V tomto úkolu je zobrazeno několik možnosti a více odpovědí může být správných. Tomuto úkolu je třeba předat v konstruktoru objekt typu *Question* a na základě hodnot z pole *variants* jsou do layoutu dynamicky přidány v knihovně vytvořené objekty typu *Checkbox*. Při vyhodnocení úkolu jsou postupně procházeny *Checkbox* objekty a je porovnávána jejich hodnota *isChecked* s hodnotou atributu *valid* u tagu *variant*.



Obr. 6.3 Ukázka úkolu Multichoice

6.4.3 IntervalQuestion

V tomto úkolu musí uživatel odpovědět na zobrazenou otázkou pomocí správného nastavení zobrazeného rolovacího menu. Pro tento účel je využívána nativní komponenta *UIPickerView*. Aby tento objekt fungoval, musí úkol implementovat protokol *UIPickerViewDelegate* a *UIPickerViewDataSource* [27].

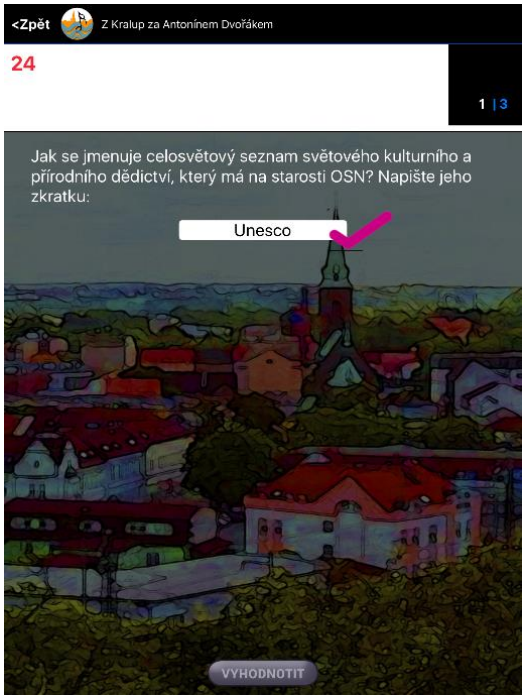
Této úloze je nutné v konstruktoru předat objekt typu *Question*. Ten obsahuje v atributu variant 7 hodnot oddělených středníkem. Jde o hodnoty správné odpovědi, začátek uznatelného intervalu, konec uznatelného intervalu, hodnotu, od které se mají generovat možné odpovědi do pickerview, hodnotu, do které se mají generovat možné odpovědi a jako poslední hodnota je velikost kroku, po kterém mají být možnosti odstupňovány. Po zpracování tohoto vstupu jsou získané hodnoty uloženy do třídy *IntervalQuestionAnswer*, dle které je úkol kontrolován.

Hodnoty uznatelného intervalu mohou být určeny jako „x" a pak musí uživatel zvolit odpověď naprosto přesně.

Obr. 6.4 Ukázka úkolu Intervalquestion

6.4.4 FilltextQuestion

Pomocí *FilltextQuestion* je pod zobrazenou otázku na obrazovku vygenerováno textové políčko jako objekt typu *UITextField*. Úkolem uživatele je pomocí klávesnice do tohoto políčka vepsat správnou odpověď. Aby mohl tento úkol fungovat, musí implementovat protokol *UITextFieldDelegate*. Úkolu je nutné v konstruktoru předat objekt typu *Question*. Dle hodnoty v něm uložené hodnoty variant je vyhodnocováno, zda uživatel vyplnit odpověď správně či nikoliv.



Obr. 6.5 Ukázka Filltext úkolu

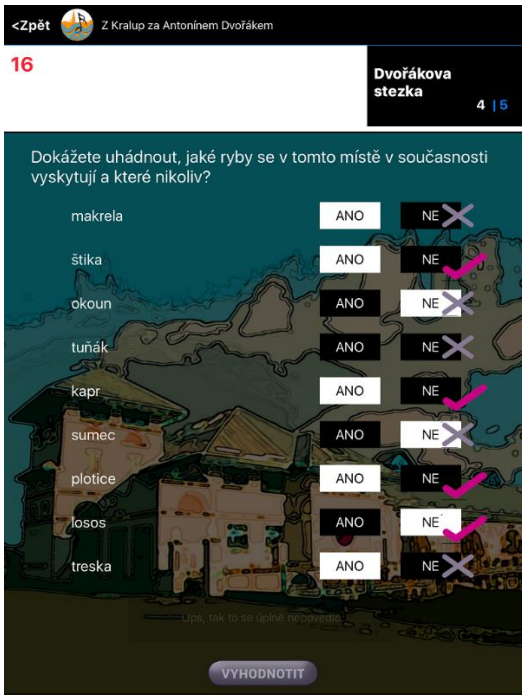
6.4.5 NumberQuestion

Úkol je velmi podobný *FilltextQuestion*, s tím rozdílem, že správnou odpověď zde tvoří číslo. Z konstruktoru vezme úkol vložený *Question* objekt. Ten v prvku variant obsahuje tři hodnoty oddělené středníkem. První značí přesnou správnou odpověď, další dvě jsou hranicí intervalu, ve kterém je možné vepsanou odpověď uznat. Tyto poslední dvě hodnoty mohou být „x“ a pak je nutné odpovědět naprosto přesně. Úkol si rozparsované hodnoty udržuje ve třídě *NumberQuestionAnswer*.

Obr. 6.6 Ukázka úkolu NumberQuestion

6.4.6 ToggleButtonsQuestion

Tento druh úkolu vezme hodnoty variant z konstruktoru vloženého *Question* objektu a pro každou z nich vloží do layoutu v knihovně vytvořený objekt *ToggleWithLabel*. Tomu je nastavena hodnota popisku dle hodnoty určené ve variant. Každý vložený objekt je typu *UIView* a obsahuje v sobě dvě přepínatelná tlačítka a jeden textový popisek. Dle kliknutí na první či druhé tlačítko je v *ToggleWithLabel* ukládána hodnota, které tlačítko je aktuálně vybrané. Tato hodnota je ukládána do atributu *isChecked*. Pro vyhodnocení úkolu jsou postupně procházeny všechny přidané objekty a je kontrolována jejich hodnota *isChecked* s hodnotou atributu *valid* v tagu *variant*.



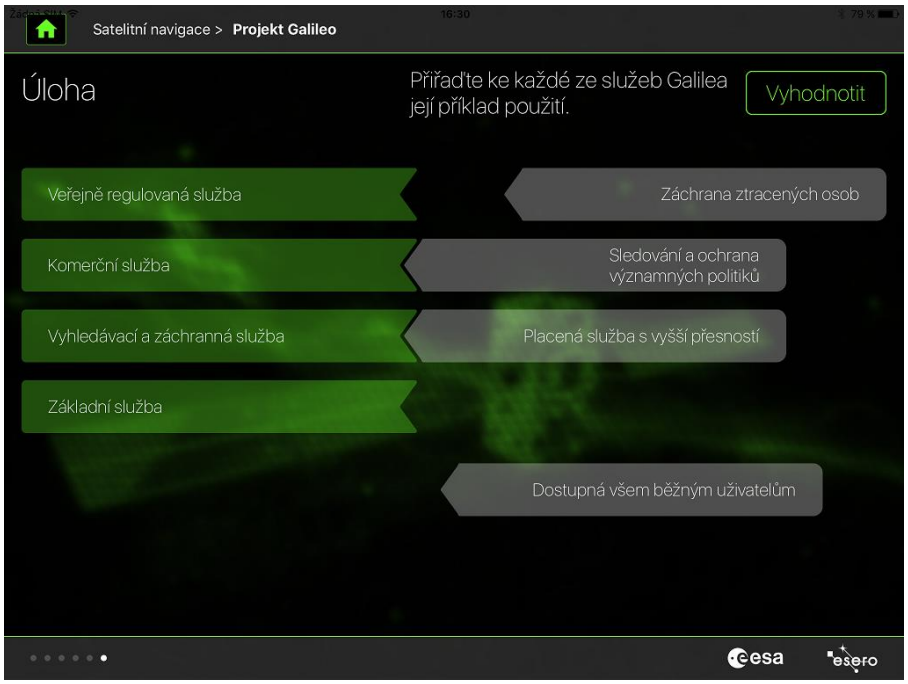
Obr. 6.7 Ukázka úkolu ToggleButtonsQuestion

6.4.7 DragDropToLineSlide a DragDropToMiddleSlide

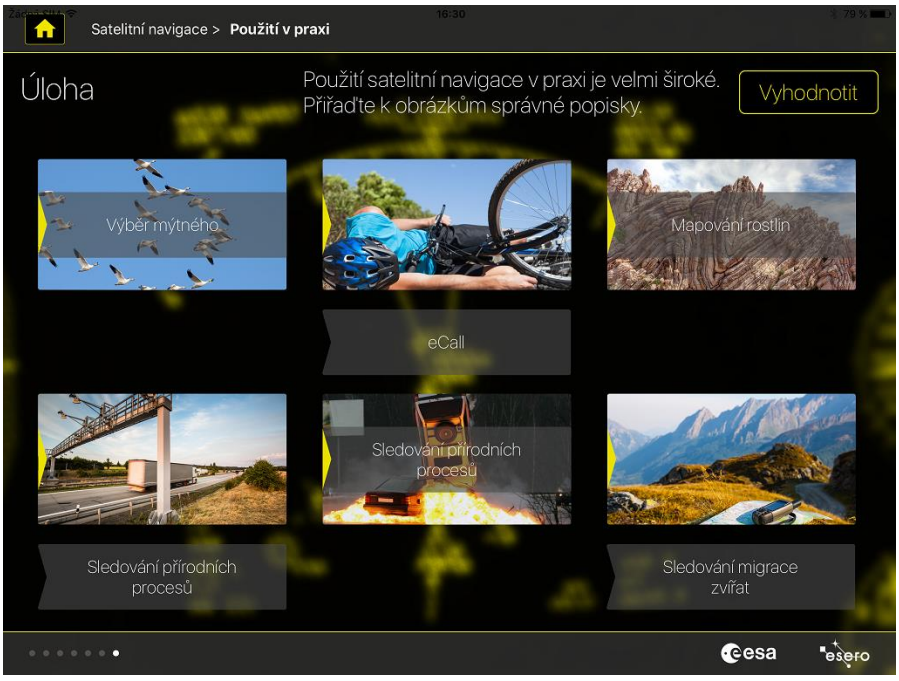
Tyto dva úkoly fungují na stejném principu, pouze se liší pozice, dle které se přetažené objekty vyhodnocují a pozice, kam objekty doskakují při jejich puštění. V *DragDropToLineSlide* je nutné objekty přetahovat do stejné výšky jako jsou fixně zobrazené prvky, ke kterými se přiřazují. Oproti tomu v *DragDropToMiddleSlide* je nutné přetahovat do prostředka určených objektů.

Pro dragdrop úkoly se používá Pod *DragDropUI*. V těchto úkolech jsou do XIB souborů, které budou využívat připravenou Swift třídu, vloženy prvky, které jsou určeny jako instance typu *DDView* z připojeného Podu. Při spuštění úkolu jsou prvky rozložené pomocí autolayoutu, je uložena jejich pozice a z přetahovatelných objektů jsou smazány všechny omezení – aby se s nimi mohlo pohybovat.

Po přetažení prstu je prozkoumáno, zda v nějaké toleranci dopadl objekt na místo, kam třeba prvky přesunout a pokud ano, je automaticky upravena jeho pozice tak, aby bylo jednoznačné, kam byl prvek umístěn. Při vyhodnocení jsou zafixovány odpovědi, které jsou na správném místě a ty, které ne, jsou vráceny na pozici, kde byly na začátku.



Obr. 6.8 Ukázka úkolu DragDropToLineSlide



Obr. 6.9 Ukázka úkolu DragDropToMiddleSlide

6.4.8 SplitImageDescSlide

Jedná se o informativní obrazovku, která zobrazuje nadpis a pod ním na stejně velké rozpůlené části obrázek a text. Úkol potřebuje do metody *initSlide()* dodat v parametrech text nadpisu, text do pravé části a *UIImage*, který se má zobrazit.

Obr. 6.10 Ukázka úkolu SplitImageDescSlide

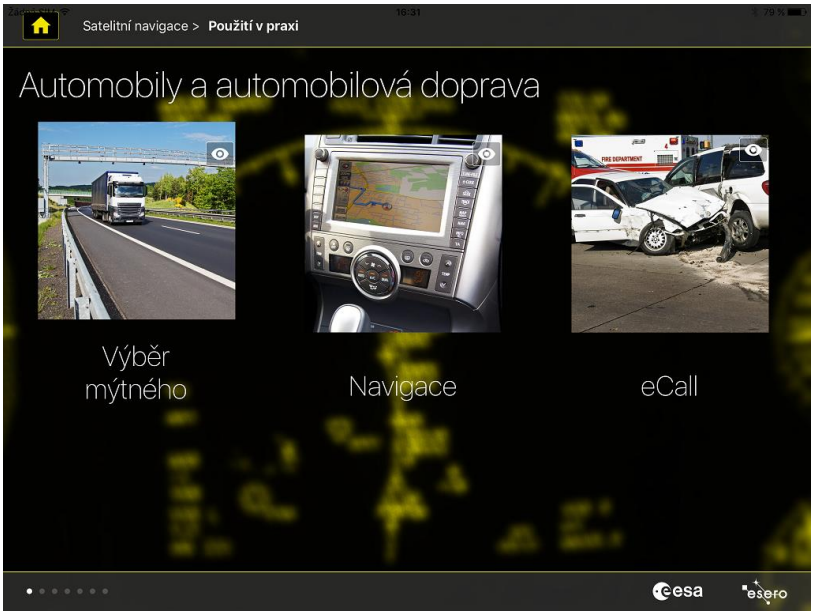
6.4.9 SplitImageDescSubtitleSlide

Tato obrazovka má stejný základ jako úkol 6.4.8, ale pod nadpisem je zde ještě umístěn nadpis druhé úrovně rozdělený na dvě části. Pro tento nadpis je nutné určit také znění textů.

Obr. 6.11 Ukázka úkolu SplitImageDescSubtitleSlide

6.4.10ThreeImagesModalSlide

Jedná se o obrazovku, kde je zobrazen nadpis a pod ním v na třetinu rozdělených částech obrázek s podnadpisem. Na každý obrázek je umístěna ikona oka, která značí, že je možné obrázek rozkliknout pro získání dalších informací. Po kliknutí na tuto ikonu nebo samotný obrázek je vyvolán *FullScreenModal*, který animací zobrazí *modal* přes celou obrazovku, na kterém je obrázek s popisem.

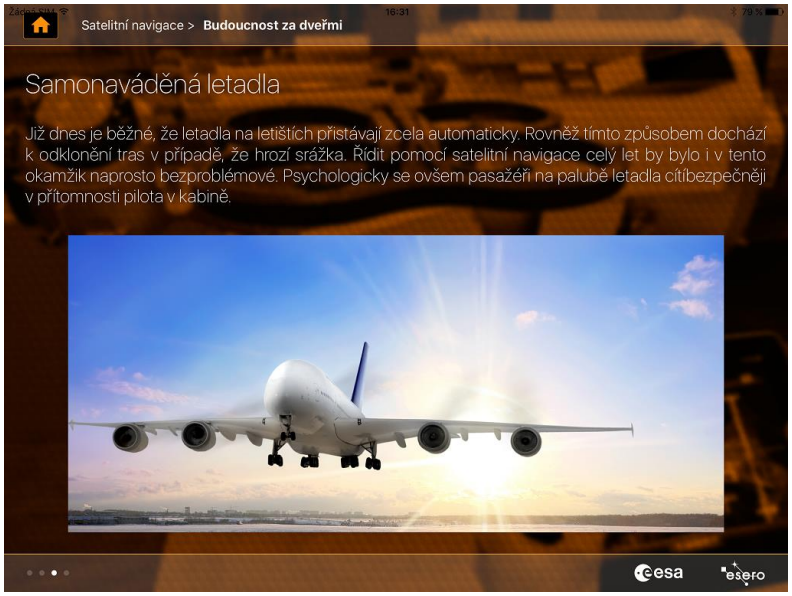


Obr. 6.12 Ukázka úkolu ThreeImagesModalSlide

Obr. 6.13 Ukázka FullScreenModal

6.4.11TitleTextBottomImageSlide

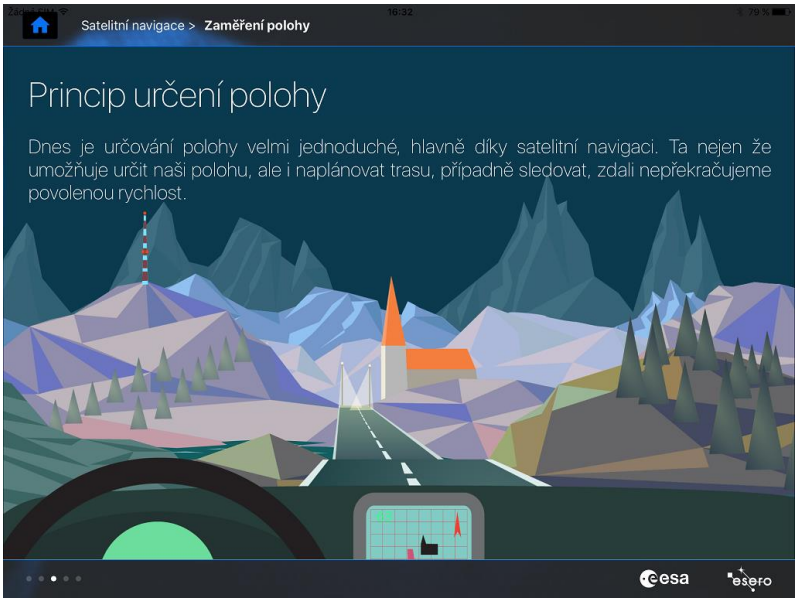
V tomto informativním okně je zobrazen pod nadpisem text a do zbytku stránky vložen obrázek. Do metody *initSlide()* je nutné v parametrech určit hodnotu nadpisu, textu a *UIImage* jako obrázek, který se má zobrazit.



Obr. 6.14 Ukázka úkolu TitleTextBottomImageSlide

6.4.12TitleTextSlide

TitleTextSlide je okno, ve kterém je na celé ploše pozadí vložen obrázek a nad ním je do hořejší části obrazovky umístěn nadpis a pod něj popisný text. Podobně jako úkoly výše je nutné určit obsah těchto objektů pomocí metody *initSlide()*.



Obr. 6.15 Ukázka úkolu TitleTextSlide

6.5 Použití vlastních úkolů

Použití vlastních úkolů, které nejsou obsaženy v množině úkolů z knihovny, je demonstrováno na příkladu v aplikaci Z kralup za Antonínem Dvořákem. Pro tvoření obrazovek, které se mají přidat do scrollview v kapitole je využita třída *SlideFactory*. Ta má instanci ve *ViewControlleru* obsluhující celou kapitolu. Nad touto instancí je volána metoda *prepareSlides(),* které je nutné předložit v parametrech pole objektů typu *QuestionSlide*, maximální počet pokusů na odpověď a referenci na *ViewController*, který *SlideFactory* volá.

V metodě *prepareSlides()* je procházeno pole *QuestionSlide* a pro každý z nich je zkoumáno, zda má atribut *type* rovno hodnotě „*custom*“. Pokud ano, je v této testové aplikaci přidán místo hotového úkolu objekt *UIView* jménem *CustomView*. Na tomto místě by se dalo například dle atributu *name* volat potřebnou třídu pro vlastní úkol.

6.6 Přidání nových znovupoužitelných úkolů do knihovny

Přidání nového generického úkolu do knihovny se skládá z několika kroků.

- V prvé řadě je nutné do dtd předpisu zanést možnost nového typu úkolu. Konkrétně je nutné vložit pro atribut *type* tagu *question* nový název druhu úkolu.
- Dále je nutné tento úkol přidat do samotného xml předpisu, tak aby splňoval dtd předpis.
- Pro úkol je nutné připravit *UIView* třídu, která bude úkol reprezentovat, případně i XIB soubor, ve kterém bude definovaný jeho vzhled, pokud nebude tvořen dynamicky.
- Ve třídách vytvářejících rozložení je nutné do výčtu úkolů přidat instanci nového úkolu, a pokud se u objektu *Question* atribut *type* rovná novému úkolu, tak ho přidat do objektu *qContainer*, do kterého se skládají všechny *QuestionComponents*.

6.7 Knihovna jako Pod

Vytvořená knihovna využívá pro své účely frameworků, na které získává závislosti pomocí Cocoapods. Při připojení knihovny přímo do aplikace, která se jí chystá využívat, vzniká složitá provázanost závislostí, se kterými si vývojové prostředí neumí poradit. Obecně řešeno, využíváním frameworků uvnitř frameworků mohou vzniknout problémy. Řešením tohoto problému je postup, kdy se z implementované knihovny vytvoří Pod do Cocoapods a do aplikace je knihovna přidána společně s ostatními závislostmi. Existuje i možnost, vytvoření privátního Podu, který nebude veřejně distribuován ke stažení. Tohoto bylo využito i při tvorbě této diplomové práce. V této podkapitole následuje postup, jak lze privátní Pod z knihovny vytvořit. Tento postup vychází z návodu [28].

Postup vytvoření privátního Podu

- Cocoapods jsou založeny na systému Git. Pro vytvoření Podu je třeba pro něj mít založený Git repozitář. Tento postup předpokládá, že knihovna je v takovém repozitáři již umístěna.
- Nejprve je třeba vytvořit soubor se specifikací (verze, název, umístění repozitáře apod.) vytvářeného Podu. Tento soubor je typu *podspec*. Vytvoření souboru se provede vykonáním příkazu

  pod spec create NazevSouboru

  uvnitř kořenového adresáře knihovny.
- Dále je potřeba vložit obsah do právě vytvořeného souboru. To se dá udělat například pomocí Xcode a soubor se v něm dá otevřít příkazem

  open –a Xcode NazevSouboru.podspec

- Uvnitř specifikačního souboru je možné určit mnoho věcí, nicméně není třeba používat všechny možná nastavení. Příklad obsahu takového souboru je uveden ve zdrojovém kódu 6.1.
- Každý Pod musí mít uveden typ licence a licenční znění. Je třeba vytvořit soubor s licenčním ujednáním, na který se bude v *podspec* souboru odkazovat. Tento soubor se typicky jmenuje *LICENSE* bez žádné přípony.
- Takto upravený projekt je poté nutné nahrát do vzdáleného repozitáře uvedeného v *podspec* souboru a přiřadit *tag*, který bude příslušný aktuální verzi Podu.
- Tímto je vytvořený Cocoapod, který ale ještě není možné používat v *Podfile* projektu. Je třeba vytvořit lokální Cocoapod referenci. To se provede příkazem

  pod repo add NazevPodu URLrepozitare

- Do takto připravené reference je třeba nahrát *podspec* soubor příkazem

  pod repo push NazevPodu NazevSouboru.podspec

- Nyní je připraven již privátní Pod k přidání závislosti pomocí Podfile. Jediným rozdílem oproti normálním Podům je, že privátnímu Podu je třeba určit cestu, na které se Pod lokálně nachází. Takže je závislost v *Podfile* například definována jako:

  pod 'ios_core', :path => '~/Documents/rychljir/ios_core'

```
Pod::Spec.new do |s|

#target platform
s.platform = :ios

#target version of platform
s.ios.deployment_target = '9.3'

#name of pod
s.name = "ios_core"

#description of pod
s.summary = "Library for creating interactive apps"
s.requires_arc = true

#pods version
s.version = "0.2.0"

#pods license, path to license file
s.license = { :type => "MIT", :file => "LICENSE" }

#name of author, contact
s.author = { "Jiri Rychlovsky" => "rychljir@gmail.com" }

#pods website, for example Github page
s.homepage = "https://github.com/rychljir/ios_core"


#path to repository, tag of commit
s.source = { :git => "https://github.com/rychljir/ios_core.git", :tag =>
"#{s.version}"}

#dependencies
s.framework = "UIKit"
s.dependency 'AEXML'
s.dependency 'PureLayout'
s.dependency 'DLRadioButton'
s.dependency 'DragDropUI'

#path to source files
s.source_files = "ios_core/**/*.{swift}"

#path to resources
s.resources = "ios_core/**/*.{png,jpeg,jpg,storyboard,xib}"
end
```

Zdrojový kód 6.1 Příklad podspec souboru

Postup aktualizace Podu po provedení změn v knihovně

Oproti přidání knihovny přímo do pracovního prostoru aplikace je lehce komplikovanější načíst provedené změny v knihovně připojené pomocí Cocoapods. Pro každou změnu v knihovně je nutné aktualizovat i používaný Pod. Je tedy nutné po každých úpravách v knihovně postupovat následovně:

- Nahrát veškeré změny do repozitáře knihovny
- Takto nahrané změny si musí odpovídat tagem s tagem uvedeným v podspec souboru Podu
- Přidat provedené změny do vytvořeného Podu příkazem

pod repo update NazevPodu

- Aktualizovat závislosti v aplikaci, která knihovnu využívá standardním způsobem - příkazem

pod install

7 OVĚŘENÍ FUNČKNOSTI

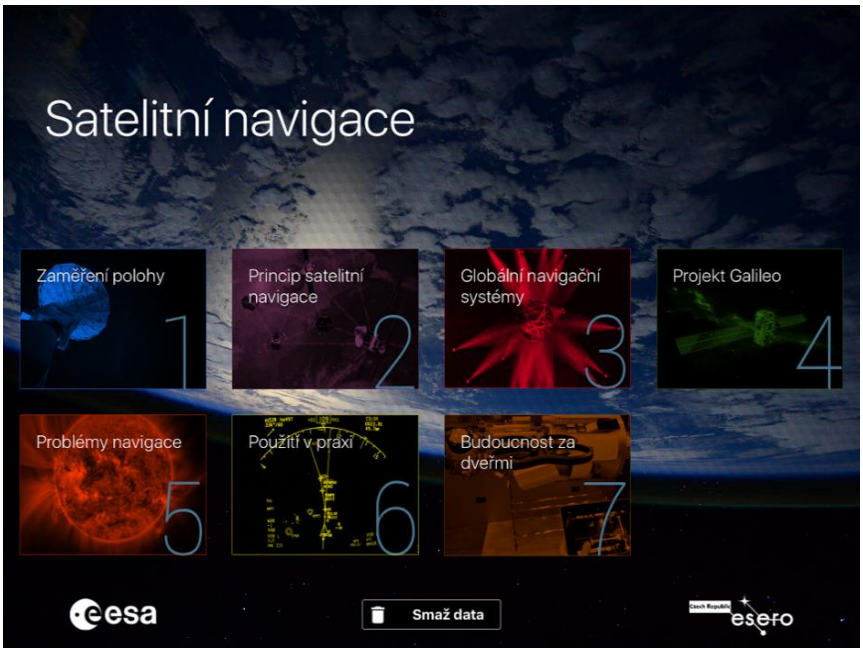
V zadání této práce je určeno, že se má převod knihovny ověřit pomocí implementace aplikace Satelitní navigace a aplikace Z Kralup za Antonínem Dvořákem. V kapitole 7 jsou popsány detaily implementace obou aplikací a zároveň je zde zmíněn způsob, jakým knihovnu využívají. U obou aplikací je popsána jejich logika fungování, struktura projektu, základní návrh UI. V závěru kapitoly je dle jednotlivých úkolů zhodnoceno, do jaké míry se povedlo úkoly převést, případně v čem se implementace navzájem liší.

7.1 Aplikace Satelitní navigace

7.1.1 Struktura aplikace

Základní jednotkou aplikací v iOS je *UIViewController*. Zjednodušeně by se dalo říci, že se jedná o obdobu *Activity* v OS Android. Satelitní navigace je složena pouze ze dvou objektů *UIViewController*. Oba mají svůj vzhled definovaný v *Main.storyboard* a třídy, které k nim patří, umístěné v kořenovém adresáři targetu.

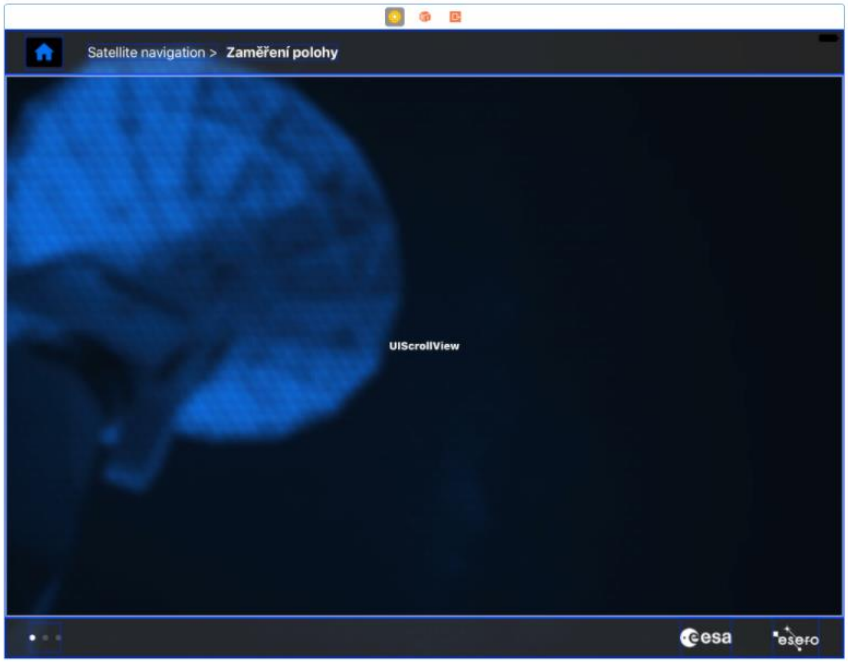
První *UIViewController* je použit po startu aplikace. Zobrazuje úvodní obrazovku, stará se o hlavní menu aplikace a tlačítko mazání uloženého stavu. Úvodní obrazovka (Obr 7.1) se skládá z pozadí, dvou obrázků, titulku a sedmi tlačítek, kterými se vstupuje do jednotlivých kapitol.



Obr. 7.1 Úvodní obrazovka aplikace Satelitní navigace

O úvodní okna se stará třída v souboru *ViewController.swift*. Ta po startu aplikace načte uložený stav a umí ho i pomocí tlačítka *Smaž data* z paměti zařízení smazat. Zároveň je tato třída spojena s tlačítky menu pomocí *outletu* a po kliknutí na tlačítko je zavolána metoda *startChapter()* s příslušným indexem. V té je vytvořena instance druhého *UIViewControlleru* v aplikaci – *ChapterViewController* a je vyvolán přechod na právě vytvořenou instanci.

ChapterViewController zde reprezentuje jednotlivé kapitoly. Má svůj vzhled určen pomocí IB v *Main.storyboard*. Obrazovka aplikace uvnitř kapitoly (Obr. 7.2) se skládá ze tří částí. V hlavičce je umístěno tlačítko, kterým se lze vrátit do úvodního menu a nadpis kapitoly. Hlavní část obrazovky je tvoří *UIScrollView*, do kterého se přiřazují jednotlivé *UIView* kapitoly. Ve spodní části je umístěna patička, která obsahuje *UIPageControl*, ve kterém se zobrazuje, na kolikátém úkolu se uživatel nachází a jde z něj vyčíst i celkový počet úkolů. Z takto definované hierarchie jsou spojeny objekty, se kterými je potřeba pracovat, pomocí *outletů*.



Obr. 7.2 Rozložení prvků uvnitř kapitoly Satelitní navigace

ChapterViewController používají všechny kapitoly aplikace, pouze se mění jeho nastavení. Proto jsou po startu načteny všechny barevné motivy kapitol, obrázky pro tlačítko zpět, pozadí a názvy kapitol. Po startu kapitoly je tedy controller stylově uzpůsoben dle toho, jaká kapitola je spuštěna. Poté jsou do pole objektů typu *UIView* přiřazeny úkoly příslušející spuštěné kapitole. Dále je správně nastavena funkčnost hlavního scrollview dle počtu obrazovek a aktualizován *PageControl*.

Úkoly, které se mají do kapitoly přiřadit, jsou definovány přímo v aplikaci a jedná se o kombinaci oken, kterými disponuje knihovna a oken, které jsou vytvořeny pouze speciálně pro Satelitní navigaci.

Aplikace si ukládá do zařízení její stav vyplnění uživatelem. Děje se tak ve *struct ApplicationState*, který obsahuje statické pole *Bool* proměnných, které značí, zda je kapitola splněna či nikoliv. Stav je ukládán pomocí integrované třídy *UserDefaults*.

7.1.2 Lokalizace

Aplikace Satelitní navigace je plně lokalizovaná do dvou jazyků - čeština a angličtina. To, který jazyk má aplikace využít je řešeno automaticky dle toho, jaký jazyk je nastaven na zařízení. Aktuálně bude využit anglický jazyk vždy, pokud není na zařízení nastavena čeština.

V případě, že je obrazovka navržena pomocí XIB souboru či v *Main.storyboard* souboru, je lokalizace tvořena pomocí XCodu a okno se překládá uvnitř IB. Pokud se jedná o úkoly z knihovny, kde aplikace nemá definovaný vzhled přímo u sebe, jsou data brána z *Localizable.strings*, což jsou soubory XML formátů, které obsahují texty pro jednotlivé jazyky.

7.1.3 Zhodnocení převodu

Naimplementované úkoly fungují bez funkčních problémů. Odlišnosti dělá práce se zdroji obrázků, protože ty byly navrženy na míru na obrazovku pro Android a rozlišení si navzájem neodpovídají, proto některé prvky nejsou správně zalícované navzájem apod. Zároveň není implementovaná část, která je v Androidu vyrobená pomocí Unity a modul rozšířené reality, protože tyto funkce jsou nad rámec ověření funkčnosti knihovny.

7.2 Aplikace Z Kralup za Antonínem Dvořákem

7.2.1 Struktura aplikace

Tato aplikace se skládá ze tří ViewControllerů. První ViewController má na starosti intro aplikace. Obsahuje úvodní obrazovky, na kterých je uživatel seznámen se základními informacemi a stručným návodem k původní aplikaci. Tento úvodní segment je tvořen stejným principem jako následně kapitoly, takže jednotlivé obrazovky jsou umístěné do stránkovatelného UIScrollView. Na poslední obrazovce tohoto controlleru je tlačítko, které spouští přechod do hlavního controlleru aplikace.

Hlavní okno aplikace obsluhuje třída ze souboru ViewController.swift. Ta obsahuje instanci controlleru reprezentující kapitoly, pole GPS lokací, na kterých jsou kapitoly umístěny a pole objektů typu Task, které mají informace o složení jednotlivých kapitol. Jednou z hlavních funkcí tohoto controlleru je ale obsluha mapy, pomocí které se uživatel pohybuje. Vzhled tohoto hlavního okna je tvořen pouze z hlavičky obsahující logo aplikace a její název. Zbytek plochy zaplňuje *MkMapView*, do kterého je vykreslena mapa.

Po startu hlavního ViewControlleru je vytvořena instance *CLLocationManager*, který slouží k zobrazení aktuální polohy uživatele. Poté je díky *QuestionParseru* prozkoumán XML předpis a z něho uložené lokace kapitol s jejich obsahem. Na získané souřadnice jsou do mapy přidány zvýrazněné body. Pro testovací účely aplikace (testování neproběhne v místě, kam je určena reálná stezka) se spouští kapitoly kliknutím na jeden z těchto bodů.

Po provedení takového kliku je dle jeho anotace nastavena instance třetího controlleru - *ChapterViewController*. Ten má definován vzhled v *MainStoryboard*. Rozhraní (Obr. 7.3) tvoří hlavička obsahující tlačítko zpět, logo aplikace a její název. Pod hlavičkou je informační sekce, ve které se zobrazuje číslo kapitoly, název kapitoly, počet kolik obrazovek kapitola obsahuje a pořadí, na které obrazovce se aktuálně uživatel nachází.

Před tím, než je kapitola zobrazena, je nastaveno její pozadí, název a číselné pořadí. Poté jsou pomocí třídy *SlideFactory* vygenerovány jednotlivá okna, vloženy do hlavního scrollview, nastaven mu *contentSize* a aktualizován počet obrazovek. Následně je kapitola zobrazena.

Obr. 7.3 Rozložení prvků uvnitř kapitoly Z Kralup za Antonínem Dvořákem

7.2.2 Lokalizace

Při generování aplikace z aktuálního formátu předlohového XML je poměrně složité zanést varianty pro více jazyků. Aktuální verze dtd tomu není uzpůsobena. Tato aplikace funguje tedy pouze v českém jazyce, ale to platí i o původní vzorové verzi pro Android.

7.2.3 Zhodnocení převodu

Původní aplikace podporovala i takové funkce jako je hledání v jízdních řádech, zobrazení zajímavostí o povodních v lokaci stezky, obsahuje speciální obrazovky určené pouze pro danou stezku. Tyto funkce jsou nad rámec zadání ověření funkčnosti knihovny a tak je aktuální verze aplikace nepodporuje. Úkoly dynamicky tvořené pomocí knihovny fungují bez problémů, včetně jejich vyhodnocení.

7.3 Zhodnocení převodu generovaných úkolů

Úkol	Hodnocení
Singlechoice	Úkol převeden pomocí DLRadioButton podu, díky tomu nemá definovaný vzhled tlačítka. Funguje bez problémů. Správně vyhodnocován.
Multichoice	Úkol převeden v plném rozsahu, funguje bez problémů, správně vyhodnocován.
Intervalquestion	Úkol převeden v plném rozsahu, funguje bez problémů, správně vyhodnocován.
FilltextQuestion	Úkol převeden v plném rozsahu, funguje bez problémů, správně vyhodnocován.
NumberQuestion	Úkol převeden v plném rozsahu, funguje bez problémů, správně vyhodnocován.
ToggleButtonsGrid	Úkol převeden v plném rozsahu, funguje bez problémů, správně vyhodnocován.
DragDrop	Úkol převeden v plném rozsahu, funguje bez problémů, správně vyhodnocován. Rozdílný je způsob vyhodnocení, původní řešení hodnotilo úkol dle barevné podkladové mapy, nové řešení vyhodnocuje úkol dle relativní pozice vůči ostatním objektům.

Tab. 7.1 Tabulka zhodnocení výsledků po převedení úkolů

8 TESTOVÁNÍ

V rámci této kapitoly je vytvořená knihovna podrobena testování. Protože aplikace mají své existující řešení na OS Android, nebylo prioritní aplikaci podrobit testování použitelnosti, protože to již proběhlo na jejich předchozích verzích. Je tedy důležité pouze prověřit existenci programových chyb a otestovat, zda knihovna správně převádí chování z původní platformy.

8.1 Unit testování

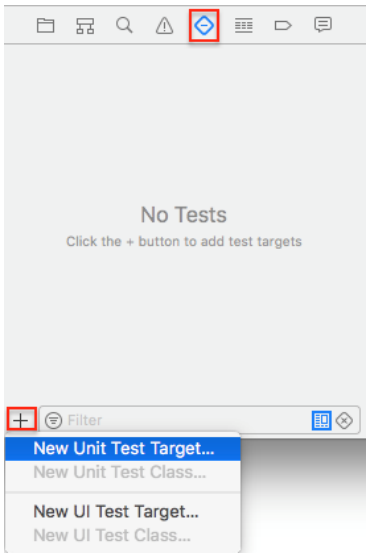
Tyto testy slouží k automatizovanému ověření, zda jsou třídy programu korektně naimplementované. Pojem „*unit test*“ vznikl z anglického pojmu „*unit*“, což v překladu znamená jednotka. Testovanou jednotkou se v tomto případě rozumí samostatně testovatelná část aplikačního programu. Typicky se jedná například o ověření správnosti vracení výsledků výpočtů či ověření správnosti práce s nějakými objekty.

Psané unit testy by měly splňovat následující vlastnosti, aby měly smysl a jejich používání bylo efektivní [29].

- Rychlé – unit testy by měly proběhnout rychle, aby jejich rychlost neodrazovala od jejich používání
- Nezávislé – každá jednotka by na sobě měla být nezávislá a běžet izolovaně od ostatních, aby si navzájem neovlivňovaly výsledky
- Opakovatelné – každý běh unit testu by měl vracet pokaždé stejný výsledek
- Plně automatizované – unit testy by měly běžet plně automatizovaně a měly by poskytovat výsledek ve formě správně / špatně. Výsledek ve formě logu, který by musel vyhodnocovat sám programátor, je zde nevhodný.

Tvorba unit testů v XCode

Vývojové prostředí XCode předpokládá využívání unit testů a je navrhnuto, k jejich co nejsnadnějšímu vytváření a jejich následnému spouštění. Obsluha testů zde probíhá v tzv. *XCode test navigatoru*. Zde se vytvoří testovací *target* a následně se zde může i spustit. Nový testovací *target* se vytvoří v levém navigačním menu prostředí v záložce *Test navigator* pomocí plus v levém dolním rohu obrazovky. Po zobrazení kontextového menu je třeba zvolit položku *New Unit Test Target…*(Obr. 8.1).



Obr. 8.1 Vytvoření nového testovacího targetu [30]

Po zvolení jména nově vytvořeného testovacího targetu je možné si otevřít právě vytvořenou Swift třídu. Tato třída obsahuje šablonu s prázdným unit testem. Importuje modul *XCTest* a musí být potomkem od třídy *XCTestCase*. Defaultně je z šablony vygenerováno několik metod testu. Metoda *setUp()* slouží k definování počátečních podmínek testu, což je například vymezení defaultního nastavení či vytvoření instance testovaného objektu. První příkaz v této metodě musí být *super.setUp()*. Druhou vygenerovanou metodou je *tearDown()*. V té se naopak musí určit, co se má stát pro proběhnutí testu. Typicky se zde mažou instance vytvořených objektů a podobně. Díky těmto metodám lze dodržovat nezávislost testů na sobě tím, že po skončení testu je vždy vše uvedeno do původního stavu. Posledním příkazem v *tearDown* funkci musí být volání *super.tearDown()*. Dále je zde třeba již definovat funkce, které budou provádět samotné testování.

Pro testování chování objektů z nějakého *targetu* v projektu je nutné tento *taget* do testovací třídy importovat. To se dělá pomocí anotace *@testable*. Výsledný import pak vypadá například: *@testable import ios_core*. Z takto vloženého modulu lze nadále využívat všechny třídy v něm obsažené, pokud mají zvolenou správnou dostupnost.

Každá testovací metoda by se měla skládat ze tří abstraktních částí. Tyto části budou demonstrovány na příkladu, kdy se ověřuje správné provedení nějakého výpočtu. V první by se měla nastavit podmínka, která bude testována. Například přiřazení hodnot, nad kterými bude probíhat nějaký výpočet. V druhé fázi by se měl daný výpočet provést a třetí fázi tvoří ověření, zda výpočet vrátil správný výsledek. Ověření správnosti se provádí pomocí různých variant metod *XCTAssert*. Je možné využívat například funkci *XCTAssertEqual(hodnota1, hodnota2, zpráva)*, která ověří, zda se hodnota 1 a hodnota 2 sobě rovnají, a pokud ne, tak označí test jako neúspěšný a je zobrazena definovaná chybová zpráva.

Spouštění unit testů v XCode

Vytvořené unit testy lze spouštět třemi způsoby. Tyto způsoby se liší především rozsahem, jaké všechny dílčí testy budou provedeny. Pomocí kliknutí na šipku vedle názvu testovací třídy v *Test navigatoru* se spustí všechny testovací metody obsažené v konkrétní třídě. Dále jsou v rozbalovacím menu navigatoru zobrazeny pod třídami jednotlivé testovací funkce a lze je spustit obdobným způsobem jako test celé třídy. Poslední možností je spuštění přímo v sekci s kódem, kde v levé části na řádku, na kterém začíná testovací metoda, je zobrazen diamant a kliknutím na něj se spustí pouze test funkce příslušející konkrétnímu diamantu.

Důležité je také konstatovat, že pokud se unit testy tvoří na target, který využívá závislostí z manažeru Cocoapods, je nutné, aby testovací target měl používané závislosti také k dispozici. Toho jde docílit jeho definováním v Podfile se stejnými závislostmi jako má testovaný target. Po zanesení těchto změn je nutné standardně provést příkaz *pod install*.

Použití unit testů v implementaci knihovny

Knihovna vytvořená v této práci umožňuje aplikacím využívat především objekty *UIView*. Ověření jejich funkčnosti je popsáno v kapitole 8.2. Pomocí unit testů je v knihovně otestována například funkčnost *StringStyleru*, který převádí v parametru dodaný text v HTML formátu do objektu typu *NSAttributedString*. Ve zdrojovém kódu 8.1 je ukázka vybrané části testovací třídy, která ověřuje, zda *StringStyler* odstraňuje ze zdrojového kódu HTML značky. Další důležitou částí, kterou unit testy v knihovně ověřují je testování korektnosti naparsování XML předlohy. Tyto testy probíhají na menším testovacím vstupu a je vhodné ověřovat jejich správnost při každém přidání nové funkčnosti do knihovny.

```
import XCTest
@testable import ios_core

class testStringStyler: XCTestCase {

    var styler: StringStyler!

    override func setUp() {
        super.setUp()
        styler = StringStyler()

    }

    override func tearDown() {
        styler = nil
        super.tearDown()
    }

    func testStylerTagSub(){
        let testText = "<sub>test</sub>"

        let result = styler.convertText(inputText: testText)

        XCTAssertEqual(result.string, "test", "Tag <sub> wasnt removed!")
    }
}
```

Zdrojový kód 8.1 Příklad unit testu na odstranění tagu <sub>

8.2 Testování funkcionality

Původní verze aplikací byly podrobeny uživatelskému testování a tak, pokud bude funkcionalita verzí převedených na platformu iOS shodná, není nutné rozhraní již podrobovat tomuto testování znovu. Faktická funkčnost částí knihovny byla ověřena pomocí unit testů. Krátké zhodnocení míry úspěšnosti převedení generovatelných úkolů bylo popsáno již v kapitole 7.3. Tato podkapitola obsahuje výsledky otestování všech úkolů, kterými knihovna disponuje. Prověření jejich funkcionality proběhlo porovnáním jejich funkčnosti s existujícím řešením na platformě Android. Srovnání proběhlo na zařízeních téměř stejné velikosti, aby byly možné předpokládat, co nejpodobnější výsledky. Výsledky tohoto testování jsou zaneseny do tabulky 8.1.

Název v knihovně	Zjištění	Komentář příčiny, návrh na zlepšení
obecné dynamicky generované obrazovky	plynulejší chod iOS verze	výkonnější iOS stroje, lepší práce s pamětí
	obrázky iOS verze působí vizuálně na stránce lépe	použití autolayoutů
SinglechoiceQuestion	jiné grafické zpracování puntíku	použití externího Podu, vytvoření vlastního objektu simulujícího chování RadioButtonu
MultichoiceQuestion	shodná funkcionalita	
IntervalQuestion	jiné grafické zpracování pickeru	použitá nativní komponenta UIPickerView
FilltextQuestion	jiné grafické zpracování vyplňujícího políčka	použití nenastylovaného UITextField, použít vlastní UITextField s upraveným vzhledem
NumberQuestion	jiné grafické zpracování vyplňujícího políčka	použití nenastylovaného UITextField
	jiná verze klávesnice	použitá klávesnice i s písmeny, změna typu klávesnice na číselnou
ToggleButtonsQuestion	jiné grafické zpracování tlačítek	chybějící iOS nativní ToggleButton
DragDropToLineSlide	shodná funkcionalita	
DragDropToMiddleSlide	shodná funkcionalita	
SplitImageDescSlide	špatně zalícované okraje obrázků s textem	iOS verze nemá sobě uzpůsobené velikosti zdrojových obrázků a tak je frame jinak velký než samotný obsah, přidat do aplikace všechny varaianty potřebné velikosti zdrojových obrázků a správně je uzpůsobit poměru displeje
SplitImageDescSubtitleSlide	shodná funkcionalita	
ThreeImagesModalSlide	jinak umístěna ikonka značící náhled obrázku	iOS verze nemá sobě uzpůsobené velikosti zdrojových obrázků a tak je frame jinak velký než samotný obsah, přidat do aplikace všechny varaianty potřebné velikosti zdrojových obrázků a správně je uzpůsobit poměru displeje
TitleTextBottomImageSlide	shodná funkcionalita	
TitleTextSlide	text iOS verze působí vizuálně lépe	iOS umožňuje zarovnání do bloku, zatímco Androidu tato funkce chybí

Obr. 8.2 Tabulka nálezů testování funkcionality

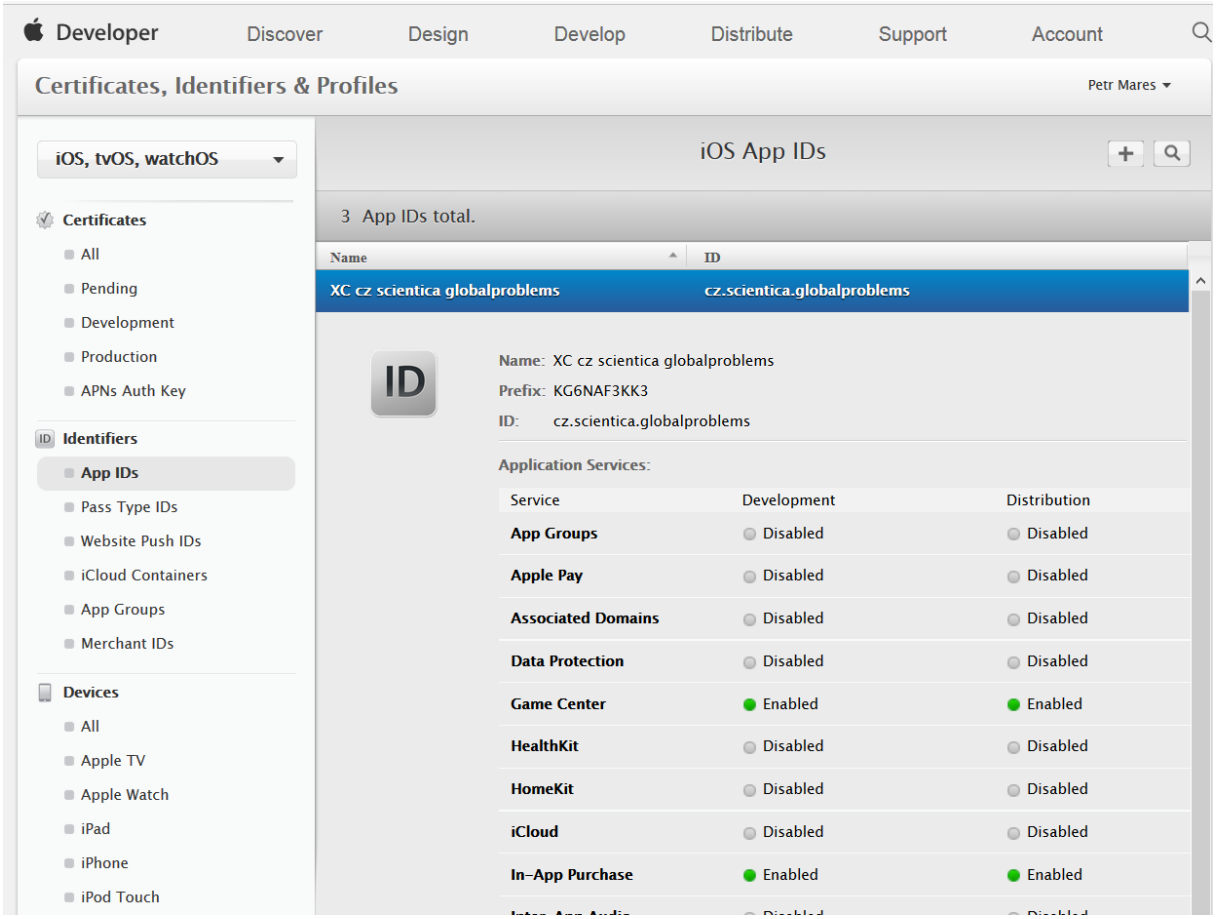
9 NASAZENÍ BETA VERZE

Testování vytvořené aplikace pro iOS před případným nahráním programu do App storu je vhodné aplikaci nejprve nechat prověřit uživateli. Proces vytvoření testovací verze není napoprvé úplně triviální, a proto tato kapitola stručně popíše postup, jak dostat vytvořený projekt k testovacím subjektům.

Při vyvíjení projektu v Xcode je nutné mít v Apple developer prostředí vytvořený účet, se kterým bude projekt svázaný. V případě, že chceme projekt dostat do App storu, musí bohužel mít účet zaplacené předplatné. Tím si u společnosti Apple vývojář zpřístupní zakázané funkce. Pro možnost vydání aplikace je tedy nutné mít projekt přiřazený ke správnému developer účtu nebo například mít správně nastavený bundle identifier, aby splňoval potřebný formát a zároveň byl unikátní.

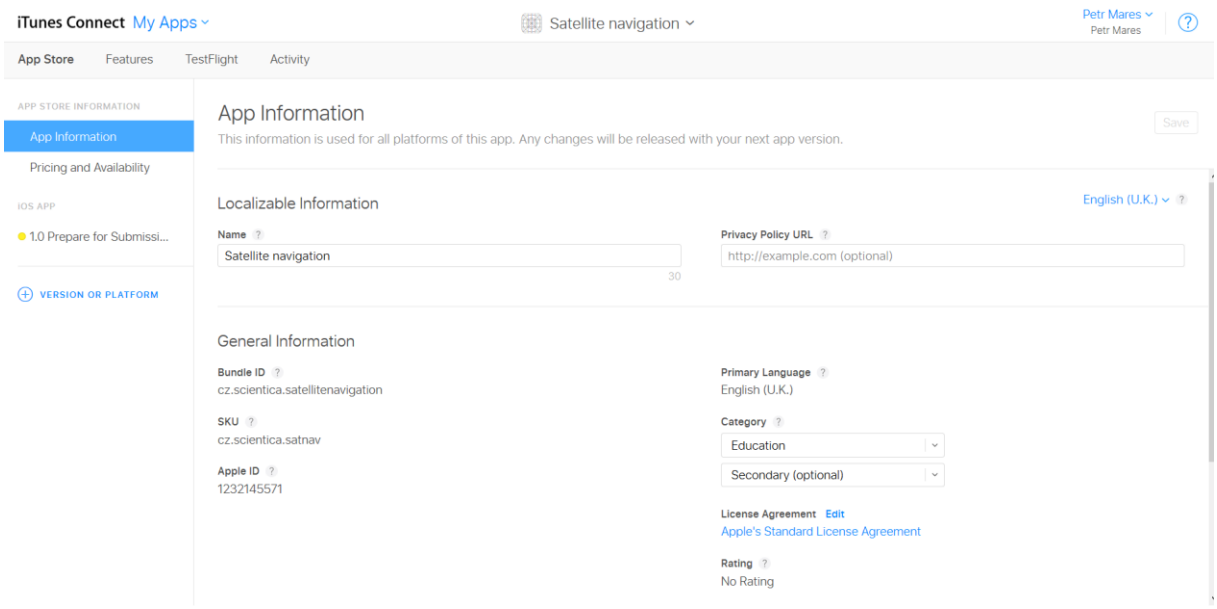
Pokud jsou tyto podmínky splněny, je třeba si na webovém portálu developer.apple.com vytvořit iOS certifikát. Na ten se pomocí stáhnutého souboru vytvoří požadavek v aplikaci Klíčenka na vývojovém Mac stroji. Získaný certifikát je pak třeba nahrát ke svému developerskému účtu.

Dalším krokem je na zmíněném webovém portálu vytvoření app ID, pod kterým se bude aplikace do storu nahrávat. Je zde třeba určit unikátní identifikátor aplikace, jaké služby operačního systému aplikace vyžaduje nebo třeba název aplikace (Obr. 9.1).



Obr. 9.1 Ukázka vytvořeného App ID

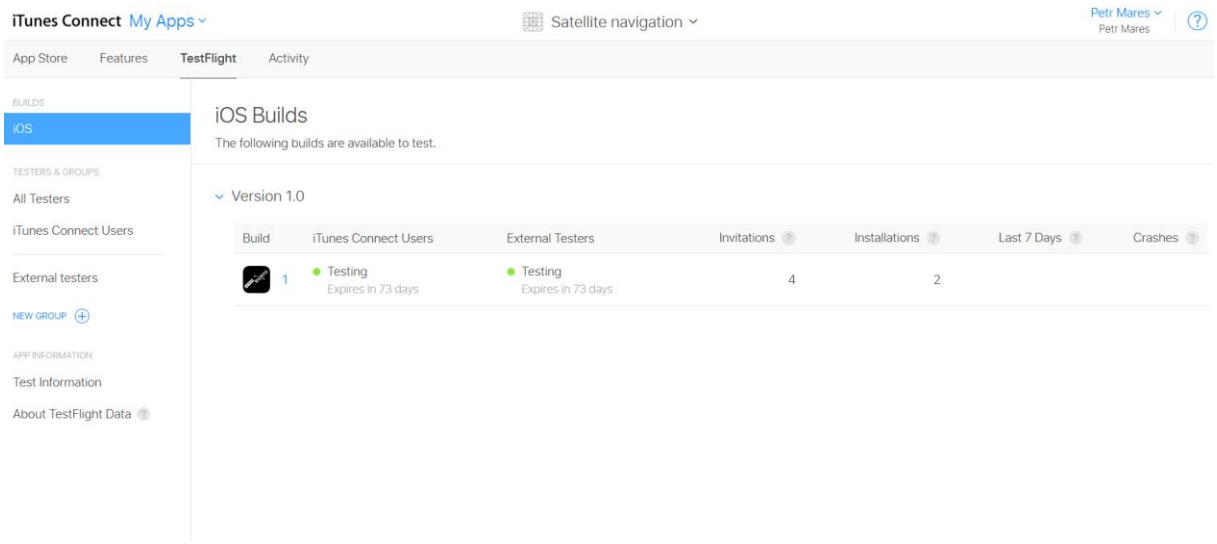
V případě, že je vše správně vyplněno, je možné jít na webový portál iTunes Connect (Obr. 9.2), kde se přidá nová aplikace do My Apps. Poté se upload projektu provede přímo z XCode. Před tímto uploadem nesmí být porušeny podmínky storu (aplikace musí mít vlastní ikonu ve všech velikostech apod.).



Obr. 9.2 Ukázka vytvořené aplikace v iTunes Connect

Po úspěšném nahrání buildu do iTunes connect je třeba vytvořit TestFlight. Pro ten se zadají informace o testování – popis verze testu, email, na který můžou testeři volat, URL adresu na licenční ujednání projektu apod.

Při správném nastavení výše zmíněných kroků je předán build k beta test review, kde pracovníci Applu musí projekt shledat nezávadným. Pokud aplikace schvalovacím procesem úspěšně projde, stačí pak už jen poslat emailové pozvánky testovacím uživatelům



Obr. 9.3 Ukázka běžícího TestFlight v iTunes Connect

10 ZÁVĚR

Cílem této diplomové práce bylo převést řešení existující knihovny pro tvorbu interaktivních aplikací typu naučné stezky z operačního systému Android na operační systém iOS. V rámci práce byla provedena analýza dodané knihovny a dvou Android aplikací, které tuto knihovnu využívají. Na začátku práce byl zhodnocen jejich návrh a blíže rozebrána jejich implementace.

Pro správný návrh způsobu převodu mezi operačními systémy byla provedena rešerše možných způsobů multiplatformního vývoje. Z ní vyplynulo, že je možné aplikace vyvíjet nativně pro každou platformu zvlášť, hybridně anebo aplikaci na mobilních zařízení zobrazovat jako web. Každá z těchto možností byla blíže popsána a byly zhodnoceny její výhody a nevýhody. Pro hybridní vývoj proběhla analýza několika vybraných frameworků a pro každý je uvedeno stručné shrnutí faktů o něm. Na základě těchto získaných informací bylo nutno zvolit pro praktickou část vhodný způsob vývoje a po zhodnocení všech výhod a nevýhod byla zvolena pro převod nativní implementace.

Kapitola 4 přibližuje čtenáři některé důležité postupy či programy, které se při nativním vývoji pro iOS běžně používají, či které značně vývojáři usnadňují práci. Konkrétně je zde popsáno například vývojové prostředí Xcode, iOS simulátor nebo manažer závislostí Cocoapods.

V praktické části této diplomové práce byla nativně vyvinuta knihovna, která je obdobou původního dodaného existujícího řešení na Android. Tato knihovna obsahuje množinu interaktivních či informativních úkolů, kterých je možné využít pro usnadnění práce při tvorbě interaktivních aplikací. Zároveň poskytuje možnost vytváření aplikací z předlohového XML předpisu, který obsahuje seznam úkolů roztříděných do kapitol, které mohou být spojeny s konkrétní GPS lokací. Pro ověření funkčnosti byly implementovány aplikace Satelitní navigace a Z Kralup za Antonínem Dvořákem, které jsou navrženy dle jejich již existujících verzí z OS Android.

Závěr práce tvoří popis testování vytvořené knihovny a popis postupu nasazení beta verze aplikací pro jejich testování pozvanými testery.

Aplikace ověřující funkčnost knihovny úspěšně fungují dle jejich původních řešení a ověřují, že knihovna byla funkčně nativně vyvinuta pro platformu iOS. Implementace těchto aplikací se omezuje pouze na ověření funkčnosti knihovny a tak v nich chybí některé funkce, které jejich Android verze mají (například hledání jízdních řádů v aplikaci Z Kralup za Antonínem Dvořákem).

LITERATURA

[1] Z Kralup za Antonínem Dvořákem - Aplikace pro Android ve službě Google Play. *Google Play* [online]. 2015 [cit. 2017-05-21]. Dostupné z: https://play.google.com/store/apps/details?id=cz.scientica.kralupy

[2] ESERO: Satelitní navigace - Aplikace pro Android ve službě Google Play. *Google Play* [online]. 2016 [cit. 2017-05-21]. Dostupné z: https://play.google.com/store/apps/details?id=esero.scientica.cz.satnav

[3] HTML5 vs Native Mobile App Development: Which option is best? *Slideshare* [online]. [cit.2017-01-02]. Dostupné z: http://www.slideshare.net/appcelerator/appcelerator-html5-prezfinal/3-125_MILLION_APPS_TODAY_650k

[4] MEHTA, Jalja. Native, HTML5 or Hybrid: Which Mobile Application Development Options are fit for you? In: *Hyperlink Infosystem* [online]. [cit. 2017-01-07]. Obrázek ve formátu JPG. Dostupné z: https://www.hyperlinkinfosystem.com/assets/blogimages/native-app-vs-html5-mobile-application.jpg

[5] Should You Build a Hybrid Mobile App? *Upwork* [online]. [cit. 2017-01-02]. Dostupné z: https://www.upwork.com/hiring/mobile/should-you-build-a-hybrid-mobile-app/

[6] Apache Cordova Tutorial: Developing Mobile Applications with Cordova. *Toptal* [online]. [cit. 2017-01-02]. Dostupné z: https://www.toptal.com/mobile/developing-mobile-applications-with-apache-cordova

[7] Architectural overview of Cordova platform. *Apache Cordova* [online]. [cit. 2017-01-02]. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

[8] Architectural overview of Cordova platform. In: *Apache Cordova* [online]. [cit. 2017-01-07]. Obrázek ve formátu PNG. Dostupné z: https://cordova.apache.org/static/img/guide/cordovaapparchitecture.png

[9] 7 Reasons Why Facebook's React Native Is the Future of Hybrid App Development. *Upwork* [online]. [cit. 2017-01-02]. Dostupné z: https://www.upwork.com/hiring/mobile/react-native-hybrid-app-development/

[10] 10 důvodů, proč (ne)používat React Native. *Ackee* [online]. [cit. 2017-01-02]. Dostupné z: https://www.ackee.cz/blog/10-duvodu-proc-nepouzivat-react-native-1-cast/

[11] Introduction to React Native: Building iOS Apps with JavaScript. In: *Appcoda* [online]. [cit. 2017-01-07]. Obrázek ve formátu PNG. Dostupné z: http://www.appcoda.com/wp-content/uploads/2015/04/react-native.png

[12] Top 10 Cross-Platform Mobile Development Tools. *Hongkiat* [online]. [cit. 2017-01-02]. Dostupné z: http://www.hongkiat.com/blog/cross-mobile-platform-framework-wora/

[13] Cross Platform Mobile Development Tools: Ending the iOS vs. Android Debate. *Think apps* [online]. [cit. 2017-01-02]. Dostupné z: http://thinkapps.com/blog/development/develop-for-ios-v-android-cross-platform-tools/

[14] Building native mobile apps with Titanium. In: *Titanium-mobile* [online]. [cit. 2017-01-07]. Obrázek ve formátu PNG. Dostupné z: http://cms.titanium-mobile.jp/tcad_certification/205/images/tistudio_architecture_med.png

[15] Comparing The Top Frameworks For Building Hybrid Mobile Apps. *Tutorial zine* [online]. [cit. 2017-01-02]. Dostupné z: http://tutorialzine.com/2015/10/comparing-the-top-frameworks-for-building-hybrid-mobile-apps/

[16] Introduction to mobile hybrid mobile app development with Ionic/AngularJS. In: *Beryl systems* [online]. [cit. 2017-01-07]. Obrázek ve formátu PNG.
Dostupné z: http://berylsystems.com/blog/mobile-app-development-with-ionic

[17] *Onsen.io* [online]. In: *Onsen.io*. [cit. 2017-01-07]. Obrázek ve formátu PNG.
Dostupné z: https://onsen.io/images/logo/onsen_with_text.png

[18] Unity Raises $181 Million Series C in Anticipation of VR/AR Growth. In: *Roadtovr* [online]. [cit. 2017-01-07]. Obrázek ve formátu PNG.
Dostupné z: http://www.roadtovr.com/wp-content/uploads/2015/03/Unity-Logo-featured.png

[19] Pros and Cons of Cross-Platform Mobile App Development. *InfoQ* [online]. [cit. 2017-01-02].
Dostupné z: https://www.infoq.com/articles/mobile-cross-platform-app-development

[20] XCode 8. *Apple developer* [online]. [cit. 2017-01-02].
Dostupné z: https://developer.apple.com/xcode/

[21] BUTTFIELD-ADDISON, Paris, Jon MANNING a Tim NUGENT. *Learning Swift: Building Apps for OS X and IOS* [online]. O'Reilly Media, 2017 [cit. 2017-05-10]. ISBN 978-1491940747.

[22] Auto Layout Guide: Understanding Auto Layout. *Apple Developer* [online]. [cit. 2017-05-15].
Dostupné z:
https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/AutolayoutPG/

[23] Autolayout square. In: *RayWenderlich* [online]. [cit. 2017-05-10]. Obrázek ve formátu PNG.
Dostupné z: https://koenig-media.raywenderlich.com/uploads/2014/11/autolayout-square-320x320.png

[24] What is Cocoapods. *Cocoapods* [online]. [cit. 2017-05-21]. Dostupné z: https://cocoapods.org

[25] Get started. *Cocoapods* [online]. [cit. 2017-05-21]. Dostupné z: https://cocoapods.org

[26] BUTTFIELD-ADDISON, Paris, Jonathon MANNING a Tim NUGENT. *Swift Development with Cocoa: Developing for the Mac and IOS App Stores* [online]. O'Reilly Media, 2014 [cit. 2017-05-10]. ISBN 9781491909706.

[27] NAHAVANDIPOOR, Vandad. *IOS 8 Swift Programming Cookbook* [online]. O'Reilly Media, 2014 [cit. 2017-05-10]. ISBN 978-1-4919-0869-3.

[28] How to Create a CocoaPod in Swift. *Ray Wenderlich* [online]. [cit. 2017-05-10].
Dostupné z: https://www.raywenderlich.com/99386/create-cocoapod-swift

[29] IOS Unit Testing and UI Testing Tutorial. *Ray Wenderlich* [online]. [cit. 2017-05-10].
Dostupné z: https://www.raywenderlich.com/150073/ios-unit-testing-and-ui-testing-tutorial

[30] IOS Unit Testing and UI Testing Tutorial.In: *Ray Wenderlich* [online]. [cit. 2017-05-10].
Obrázek ve formátu PNG.
Dostupné z: https://koenig-media.raywenderlich.com/uploads/2016/12/TestNavigator1.png

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

OS	operační systém
XML	eXtensible markup language
ESA	European Space Agency
ESERO	European Space Education Resource Office
Layout	grafické rozvržené plochy
DTD	Document Type Definition
Pod	knihovna dostupná v Cocoapods manažeru
iOS	operační systém zařízení firmy Apple
UI	uživatelské rozhraní
HTML	HypetText Markup Language