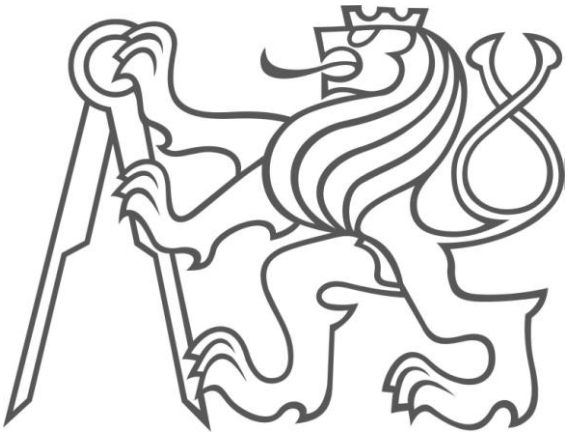
České vysoké učení technické v Praze

Fakulta elektrotechnická



DIPLOMOVÁ PRÁCE

Multiplatformní knihovna pro výukové aplikace

2017 Josef Veselý

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Josef Veselý

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: Multiplatformní knihovna pro výukové aplikace

Pokyny pro vypracování:

Analyzujte stávající knihovnu úkolů pro vytváření edukativních aplikací, která je implementovaná pro operační systém (OS) Android (knihovnu dodá vedoucí práce). Implementujte její funkcionalitu v multiplatformním vývojovém prostředí (frameworku) založeném na jazyce JavaScript. Analyzujte vhodné frameworky s primárním cílem implementace pro tablety s OS iOS a Android. Knihovnu implementujte ve vybraném frameworku.

Funkcionalitu knihovny ověřte přenesením výukové aplikace 'Globální problémy z nadhledu' (dodá vedoucí práce) z prostředí Android do vámi implementované knihovny. Ověřte funkčnost minimálně na dvou rozdílných zařízeních pro každý z uvedných OS.

Seznam odborné literatury:

[1] Learning React Native: Building Native Mobile Apps with JavaScript. Bonnie Eisenman. O'Reilly Media, Inc., 2015.
[2]https://cordova.apache.org/
[3] http://www.reactnative.com/
[4] https://play.google.com/store/apps/details?id=cz.scientica.esero.global

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 nejprve analyzuje existující knihovnu pro tvorbu výukových aplikací pro tablety s operačním systémem Android pro potřeby její následné reimplementace v multiplatformním prostředí. Následuje analýza *frameworků* pro tvorbu multiplatformních mobilních aplikací v jazyce *JavaScript*, je popsán rozdíl mezi nativními a hybridními *frameworky* a detailně jsou popsány *frameworky Apache Cordova* a *React Native*. Na základě analýzy a tvorby prototypů je zvolen *framework React Native*, v němž se knihovna následně znovu implementuje. Ve stejném *frameworku* je poté implementována aplikace *Globální problémy z nadhledu*, která knihovnu pro tvorbu výukových aplikací využívá. Nakonec je aplikace otestována na zařízeních s operačními systémy Android a iOS a jsou popsány nedostatky, kterými *framework*, především na OS Android, trpí.

KLÍČOVÁ SLOVA

Android, iOS, multiplatformní mobilní framework, React Native, Apache Cordova

ABSTRACT

The existing library for creating educative applications for Android OS tablets is analyzed in the beginning of this diploma thesis. Then, frameworks for multiplatform mobile development in *JavaScript* are analyzed. Frameworks *Apache Cordova* and *React Native* are described in more detail and application prototypes are then implemented in these frameworks. Based on the analysis, framework *React Native* is chosen and the library is re-implemented in this framework. Consequently application Global Problems from Above that uses the library is implemented in the chosen framework as well. Eventually, the application is tested on real devices with Android and iOS operating systems and the flaws that *React Native*, especially on the Android operating system, suffers from are described.

KEYWORDS

Android, iOS, multi-platform mobile framework, React Native, Apache Cordova

VESELÝ, Josef. *Multiplatformní knihovna pro výukové aplikace*. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická. Diplomová práce. Vedoucí práce: Ing. David Sedláček

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma *Interaktivní Multiplatformní knihovna pro výukové aplikace* 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 za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Praze dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato diplomová práce se v úvodní části věnuje analýze knihovny úkolů pro vytváření edukativních aplikací, která je implementována pro operační systém Android. Zmíněnou knihovnu poskytl vedoucí diplomové práce. Následující kapitoly se zabývají popisem multiplatformních vývojových prostředí (*frameworků*) založených na jazyce *JavaScript*. Z těchto *frameworků* je nejvhodnější vybrán k implementaci knihovny úkolů a také k implementaci aplikace *Globální problémy z nadhledu*, která knihovnu využívá.

Aplikace *Globální problémy z nadhledu* byla původně implementována pro tablety s OS Android a úhlopříčkou obrazovky minimálně 9". Aplikace implementovaná v rámci této práce může být provozována nejen na OS Android, ale také na operačním systému iOS. Vzhled aplikace je totožný s již existující aplikací pro OS Android (obrázky a texty použité v aplikaci dodal vedoucí diplomové práce).

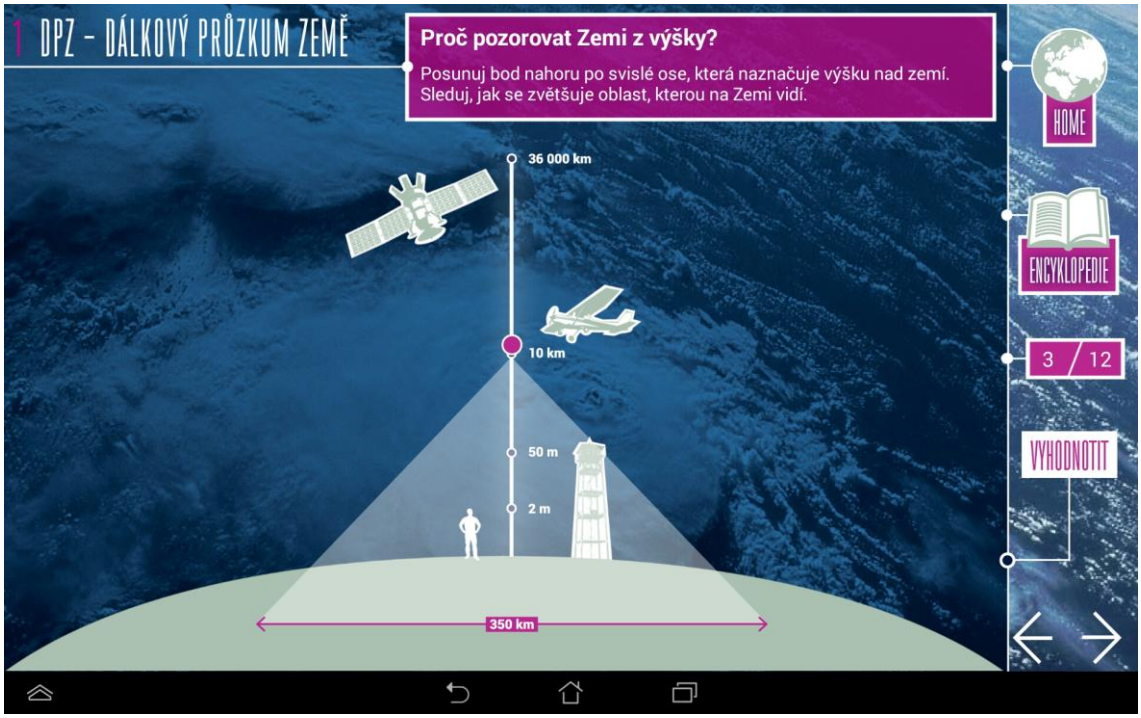
Po kapitole zabývající se implementaci následuje kapitola věnující se testování aplikace *Globální problémy z nadhledu* a komponent knihovny úkolů pro tvorbu výukových aplikací. Ergonomie uživatelského rozhraní aplikace byla již jednou testována, proto se v rámci předložené diplomové práce testuje pouze shoda s původní aplikací.

1 APLIKACE *GLOBÁLNÍ PROBLÉMY Z NADHLEDU*

V této kapitole je podrobně popsána aplikace *Globální problémy z nadhledu* tak, jak je implementována pro OS Android. Tato původní aplikace nebyla vyvinuta jako součást předložené diplomové práce, třebaže se autor na implementaci její části přímo podílel.

Aplikace využívá knihovnu úkolů pro vytváření edukativních aplikací. Znalost požadavků na funkcionalitu a znalost návrhu uživatelského rozhraní aplikace hraje důležitou roli při výběru multiplatformního *frameworku*, s jehož pomocí bude aplikace znovu implementována.

Globální problémy z nadhledu je výuková aplikace pro tablety od úhlopříčky obrazovky 9" (nově implementovaná multiplatformní aplikace by měla fungovat od úhlopříčky obrazovky 7") a verze OS Android 4 a vyšší. Student prochází aplikaci po jednotlivých kapitolách, plní zadané úkoly a po zvládnutí všech kapitol skládá závěrečný test.



Obr. 1.1: Snímek aplikace *Globální problémy z nadhledu*.

Z Obr. 1.1 je patrné, že se *layout* aplikace dělí na 3 části - horní pruh s nadpisem kapitoly a popisem úkolu, dále na pravý panel s tlačítky, číslem stránky a navigačními šipkami a poslední částí je centrální prostor pro jednotlivé úkoly. Zároveň je také patrné, že aplikace nerespektuje standardní vzhled Android nebo iOS aplikací, ale že se vzhledově blíží spíše herním aplikacím.

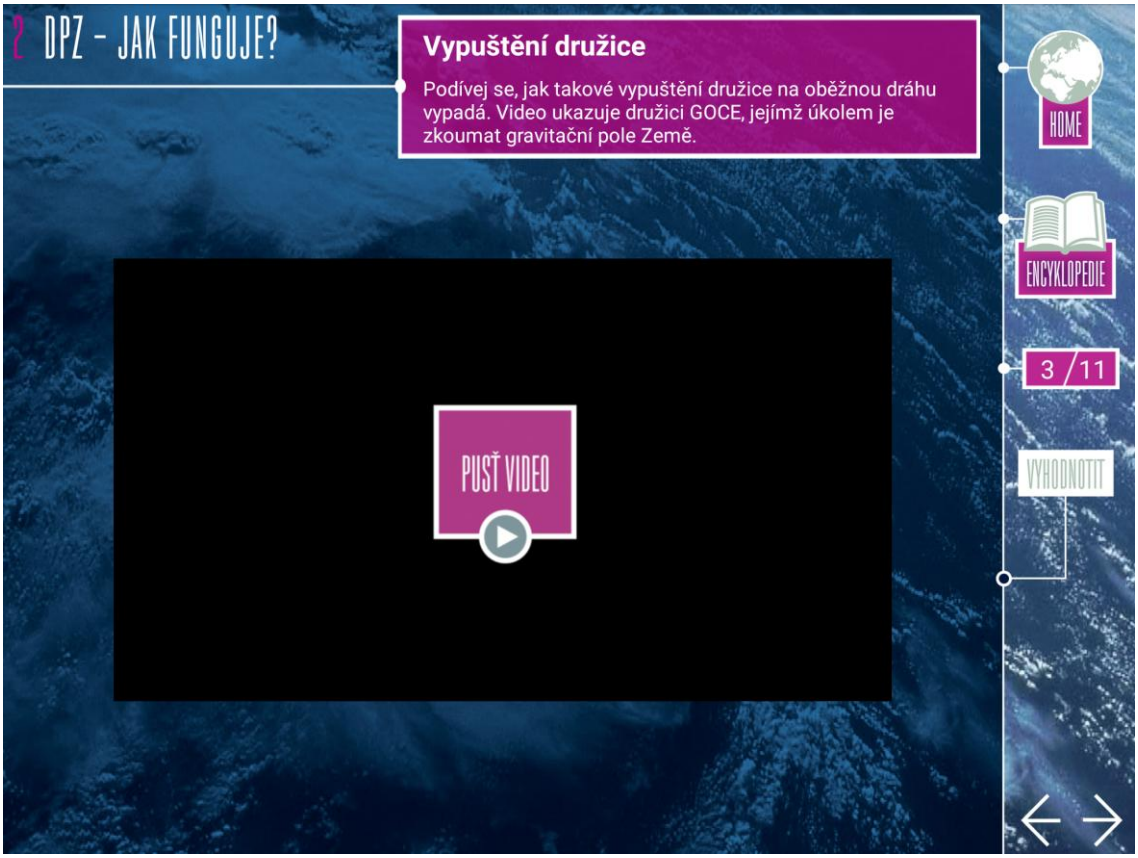
1.1 Typy úkolů

Úvodní obrazovka aplikace se skládá z animovaného 3D modelu Země a tlačítek pro zobrazení jednotlivých kategorií. Každá kategorie sestává z několika úkolů, které uživatel postupně prochází (na další úkol lze vstoupit až po splnění současného) a končí jednouchým oknem s citátem, který se vztahuje k názvu kapitoly.

Úkoly lze v zásadě rozdělit do několika funkčních kategorií, popsaných v následujících sekcích této kapitoly.

1.1.1 Sledování videa

Obrazovka obsahuje jedno nebo více videí, ke kterým se přistupuje přes HTTP protokol. Tento typ úkolu používá *Video Fragment* z kapitoly 2.2.1. Aplikace nepracuje s lokálními videi.



Obr. 1.2: Snímek obrazovky úkolu *Sledování videa*.

1.1.2 Obrázky s přidruženou informací

Na obrazovce je několik obrázků, které je možno na kliknutí zvětšit na celou obrazovku. Ke každému obrázku lze zobrazit doprovodný text. Buď je napsaný přes malý obrázek nebo je, když je obrázek zobrazen v plné velikosti, v levém dolním rámečku.

Úkol využívá *InfoClick Fragment* z kapitoly 2.2.2.

Obr. 1.3: Snímek obrazovky úkolu *Obrázky s přidruženou informací*.

1.1.3 Zobrazení informace na kliknutí

V tomto úkolu musí uživatel kliknout na určitá místa (obvykle text nebo obrázek). Po kliknutí se zobrazí doplňující informace, která může být v libovolné podobě (text, obrázek, video, animace apod.). Úkol se považuje za splněný, když uživatel klikne na všechna místa.

Úkol využívá *InfoClick Fragment* z kapitoly 2.2.2

Obr. 1.4: Snímek obrazovky úkolu *Zobrazení informace na kliknutí*.

1.1.4 Přetahování objektů na správné místo

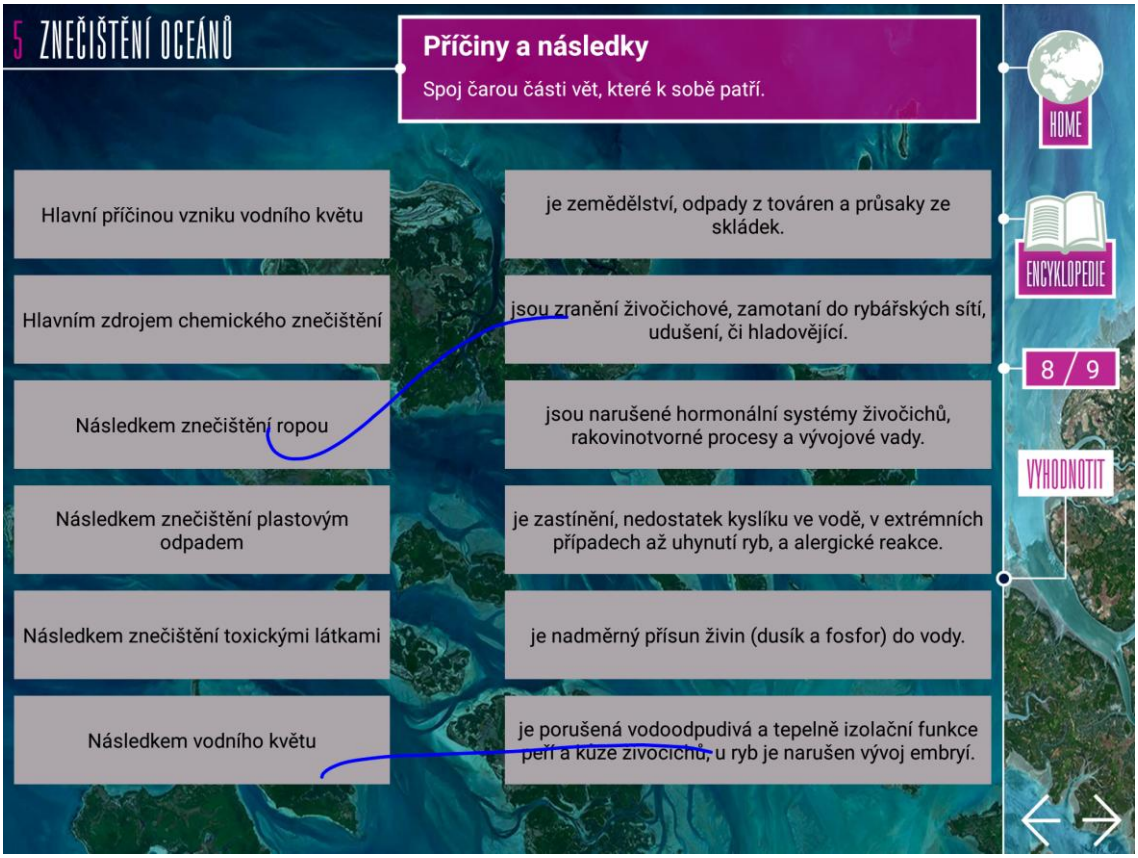
Úkol se skládá z množiny objektů (text, obrázek, kombinace), které lze po obrazovce prstem přetahovat. Dále je na obrazovce několik míst (typicky mezera v textu nebo obrázek), kam se objekty mají přetáhnout. Každý objekt může mít své vlastní místo nebo může být jedno místo pro více objektů. Špatně umístěné objekty se po vyhodnocení úkolu vrátí na své původní místo.

Úkoly jsou implementovány *DragDropQuestion Fragmentem* (kapitola 2.2.4).

Obr. 1.5: Snímek obrazovky úkolu *Přetahování objektů na správné místo*.

1.1.5 Spojení informací čárou

Na obrazovce jsou dvě skupiny objektů. Úkolem uživatele je spojit čarou (kreslí se pohybem prstu) objekty z jednotlivých skupin. Úkol je implementací *LinkPairQuestion Fragmentu* (kap. 2.2.4).

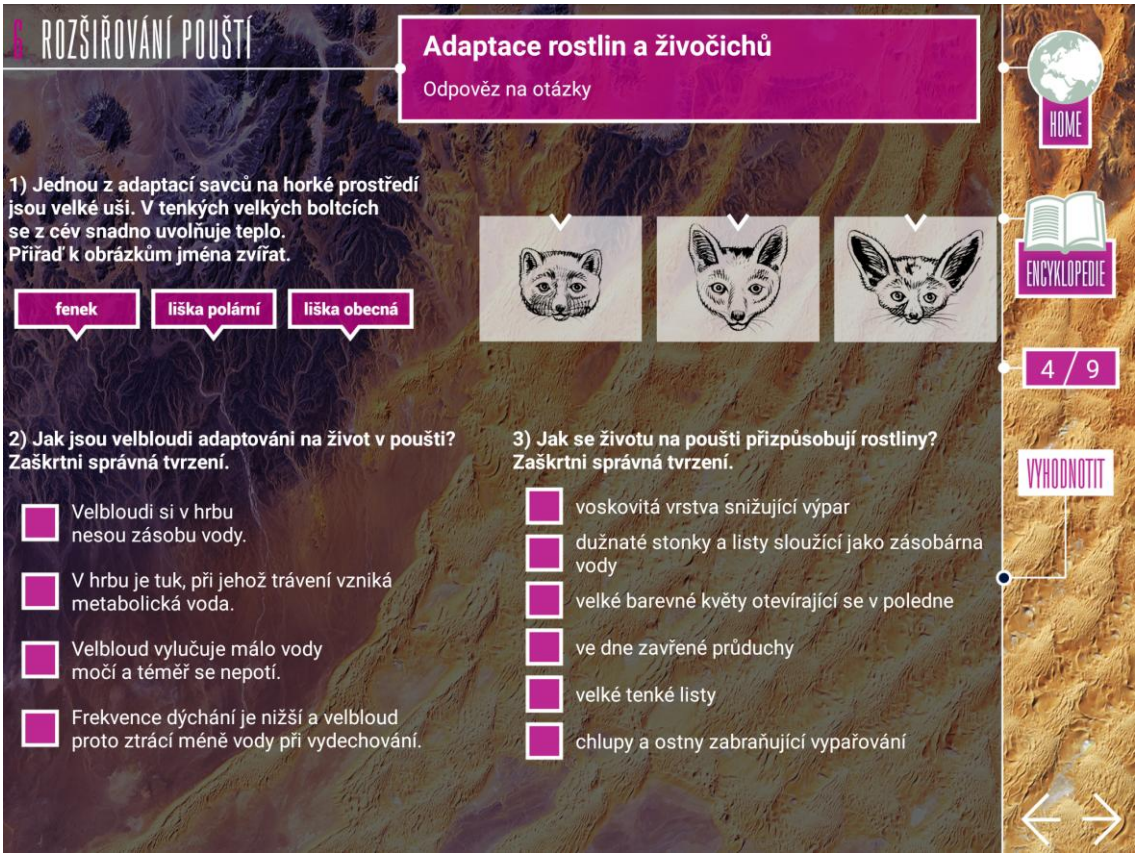


Obr. 1.6: Snímek obrazovky úkolu *Spojení informací čárou*.

1.1.6 Výběr správné odpovědi

Úkol obsahuje sadu otázek a odpovědí k nim. Správná je vždy alespoň jedna odpověď.

Jedná se o úkoly *CheckBoxQuestion* (kapitola 2.2.5).



Obr. 1.7: Snímek obrazovky úkolu *Výběr správné odpovědi*.

1.1.7 Pasivní zobrazení informace

V tomto případě je na obrazovce pouze nějaký text, který může být případně doplněn obrázkem. Od uživatele se nečeká žádná interakce. Úkoly jsou implementací *Information* Fragmentu (kapitola 2.2.6).

Obr. 1.8: Snímek obrazovky úkolu *Pasivní zobrazení informace*.

1.1.8 Obrázkové hádanky

V tomto úkolu je na pozadí obrazovky nějaký obrázek a uživatelův úkol je ze tří možností vybrat tu, která obrázek představuje. Když na první pokus vybere špatnou, má ještě jednu opravnou možnost. Po druhé špatné odpovědi se křížkem označí uživatelem vybraná odpověď a zároveň se zvýrazní odpověď správná.

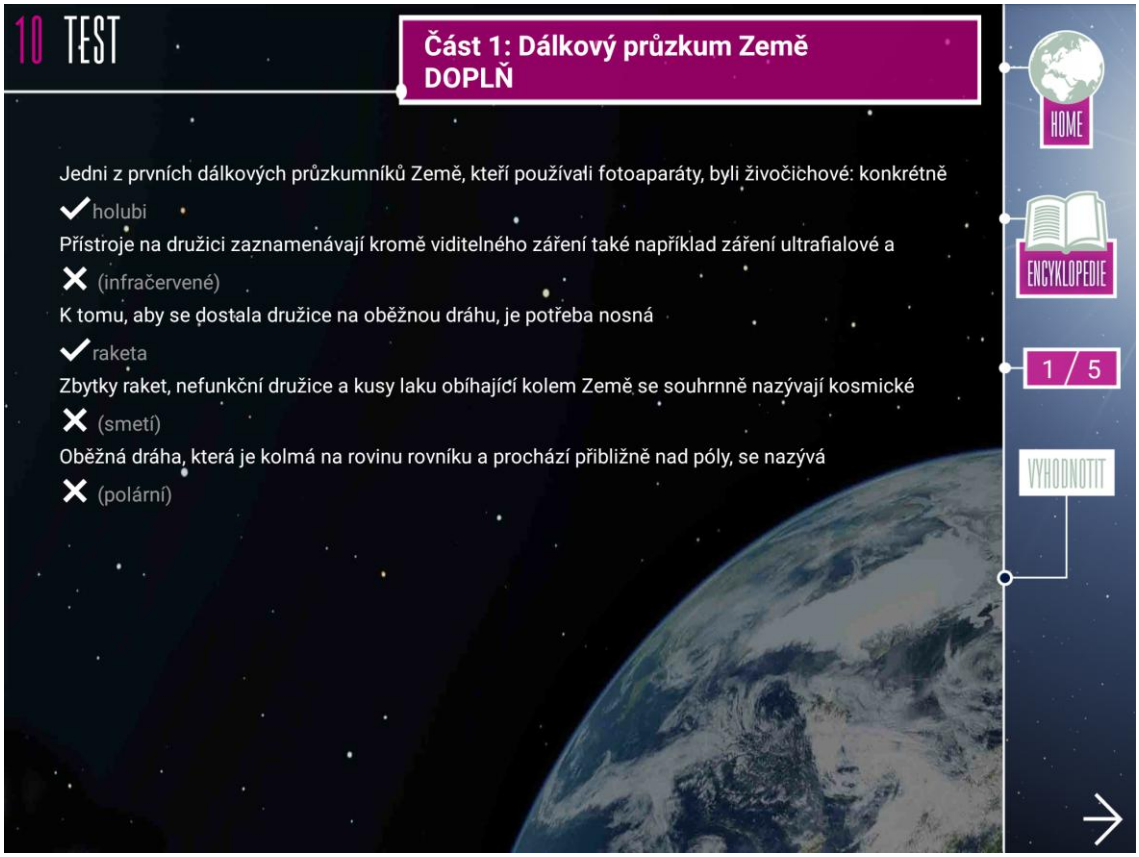
Tento typ úkolu je zástupcem automaticky generovaných testů (kapitola 2.2.7).

Obr. 1.9: Snímek obrazovky úkolu *Obrázkové hádanky*.

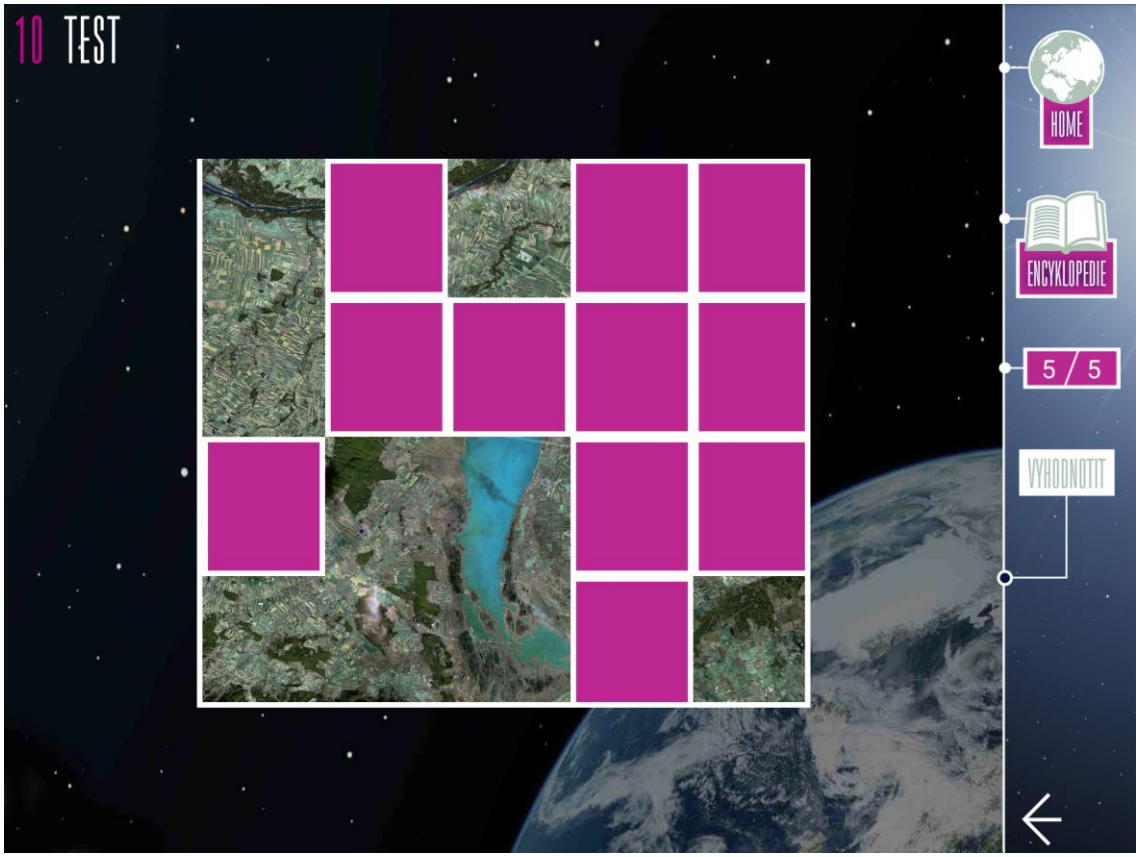
1.1.9 Závěrečný test

Závěrečný test je stejně jako Obrázkové hádanky zástupcem automaticky generovaných testů z kapitoly 2.2.7. V tomto případě se však generují otázky a *EditText View* objekty, do kterých uživatel na otázky píše odpovědi. Stejně jako v předchozí kapitole, tak i zde má dva pokusy, přičemž se po druhém špatném vyhodnocení do závorky za uživatelovu odpověď napíše očekávaná správná odpověď.

Za poslední otázkou je speciální obrazovka, na které je obrázek překrytý tolika fialovými dlaždicemi, kolik bylo otázek. Uživatel má možnost kliknout na tlačítko vyhodnotit. V takovém případě se odkryjí dlaždice, které představují otázky, na něž uživatel odpověděl správně. Obr. 1.10 a Obr. 1.11 zachycují obrazovky právě popsaných úkolů.



Obr. 1.10: Snímek obrazovky úkolu *Závěrečný test*.



Obr. 1.11: Snímek obrazovky vyhodnocení testu.

1.1.10 Kombinace

Některé úkoly se mohou skládat z kombinace dvou a více výše zmíněných typů úkolů, například *Sledování videa* a *Otázka ano / ne*, *Přetahování objektů na správné místo* a *Výběr správné odpovědi* a tak dále.

Příklad takové obrazovky lze vidět na Obr. 1.7, který spojuje úkoly typu *Výběr správné odpovědi* a *Přetahování objektů* na správné místo.

1.2 Paměť aplikace, lokalizace

Aplikace využívá lokální úložiště formou klíč-hodnota (tzv. *Preferences*) pro ukládání stavu aplikace. Ukládá se stav splněnosti úkolů, pozice přetahovatelných objektů (instance třídy *DraggableTextView*) po vyhodnocení úkolu a uživatelem zaškrtnuté odpovědi v otázkách ano / ne, výběr správných odpovědí v kvízech, v obrázkových hádankách a v závěrečném testu.

Uživatel má možnost v hlavním menu aplikace uložená data smazat.

Aplikace je vícejazyčná - konkrétně má verzi anglickou a českou. Nastavení jazyka se řídí nastavením jazyka v zařízení, na kterém je aplikace spuštěná a uživatel nemá možnost jazyk v aplikaci změnit (pouze změnou jazyka zařízení).

1.3 Závěr

V této kapitole byly popsány typy úkolů tak, jak jsou zpracovány v rámci aplikace *Globální problémy z nadhledu* a okrajově byly popsány další funkce aplikace, například držení stavu nebo lokalizace. Kromě výše popsaných typových úkolů však aplikace obsahuje i úkoly specializované, které do žádného z popsaných typů nespadají. Tato diplomová práce se věnuje implementaci i těchto úkolů ve vybraném multiplatformním *frameworku*. Následuje výčet takových úkolů s popisem funkčnosti.

- Úvodní okno: obrazovka s informací o aplikaci, která po několika vteřinách automaticky spustí další obrazovku.
- Menu: obrazovka s výběrem kapitol. Na pozadí je interaktivní animace zeměkoule využívající knihovnu Rajawali [1].
- Vertikální *SeekBar View*: úkolem uživatele je posunovat bod po vertikální ose, popř. posunout bod do určené pozice (jako příklad nechť poslouží obrázek 1.1).
- Obrázek s roletou: jedná se o obrazovku, na které je obrázek, po němž může uživatel přetahovat prstem do strany a tím odkrývat jiný obrázek, který je pod ním schovaný.
- Klikání na prostředí: v tomto typu úkolu nejprve uživatel klikáním na tlačítka postupně zobrazí dané prostředí (například prales nebo oceán) a poté z možností vybírá živočichy nebo rostliny, které dané prostředí obývají.
- Pokácej prales: úkol, kde se uživateli po kliknutí na text zobrazí obrázek motorové pily (přetahovatelné *View* - instance třídy *DraggableTextView*), kterou přetáhne přes obrázek pralesa a tím splní úkol (zároveň se obrázek změní z pralesa na pokácený prales).
- Odpověď na otázku: úkol, kde uživatel odpovídá na otázku. K získání uživatelovi odpovědi je použito *View EditText*.
- Otázky ano / ne: úkol se skládá z pole otázek, odpovědí a objektů *RadioButton*. Odpovědi se při načtení úkolů skryjí a při výběru správných *RadioButton* objektů zpětně zobrazí. Každé otázce náleží dva *RadioButton* objekty, přičemž pouze jeden se počítá jako správná odpověď.

2 KNIHOVNA ÚKOLŮ PRO VYTVÁŘENÍ EDUKATIVNÍCH APLIKACÍ

Tato kapitola popisuje původní knihovnu úkolů pro vytváření edukativních aplikací, jejíž re-implementace v některém z multiplatformních *frameworků* pro vývoj mobilních aplikací byla součástí zadání předložené diplomové práce a jež bude popsána v následujících kapitolách. V úvodu již bylo zmíněno, že původní knihovna je implementována pro operační systém Android. Znamená to, že používá knihovnu Android SDK. Protože budou v dalším textu používány termíny jako *Fragment* a *View,* které jsou s Android SDK spjaté, následující text se nejprve stručně věnuje knihovně Android SDK a teprve poté je popsána samotná knihovna úkolů.

2.1 Android SDK

Android SDK je vývojové prostředí a knihovna pro vývoj aplikací pro OS Android - tedy především pro chytré telefony a tablety. Základní prvek uživatelského rozhraní je třída jménem *View*. *View* má svou *onDraw* funkci, která jej vykreslí na obrazovku. Ze třídy *View* dědí celá řada dalších specializovanějších tříd, například *ImageView* pro vykreslování obrázků, *Text* pro zobrazení textu a podobně. Speciální třída *ViewGroup*, která také dědí od *View*, je významná tím, že slouží jako kontejner pro ostatní *View*, a tím určují rozložení prvků na obrazovce. Stejně jako *View*, rovněž *ViewGroup* má celou řadu potomků. Například *LinearLayout* je třída, která zobrazuje své vnitřní komponenty v horizontálním nebo vertikálním pořadí.

Obrazovka, kterou uživatel aplikace vidí, je tedy složena z objektů typu *View* a *ViewGroup*. Vizuální rozložení těchto objektů se nazývá *layout*. Zobrazení objektů typu *View* a obsluhu jimi generovaných událostí řídí třída jménem *Activity*. Tato třída obsluhuje *layout* obrazovky (nebo její části), vytváří instance objektů typu *View,* přiřazuje obslužné funkce jednotlivým událostem a podobně. Instance třídy *Activity* má svůj životní cyklus jdoucí od načtení do paměti, přes zobrazení uživateli, až po vymazání z paměti. Pro jednotlivé fáze životního cyklu existují metody (*onResume, onPause, onLoa*d atd.), ve kterých programátor specifikuje, co se má vykonávat.

Když tedy uživatel vyvolá kliknutím na nějaké tlačítko přechod na jinou obrazovku, vyvolá tím událost, již zpracuje instance *Activity* přiřazená původní obrazovce. Součástí této obsluhy je vytvoření a odstartování životního cyklu nové instance třídy *Activity*, která je přiřazena obrazovce, jež se zobrazí jako výsledek tohoto přechodu.

V původním návrhu operačního systému Android existovala pro každou obrazovku právě jedna správa *layoutu a* událostí, reprezentovaná instancí třídy *Activity*. Verze 3.0 operačního systému Android zavedla možnost rozdělit obrazovku na několik částí, z nichž každá má svou vlastní správu *layoutu* a událostí. To umožňuje třída *Fragment*. *Fragment* má, podobně jako *Activity,* svůj layout a svůj životní cyklus (podobný jako třída *Activity*). Nemůže však existovat samostatně. Vždy musí existovat v rámci nějaké *Activity*, se kterou vzniká i zaniká. Výhodou je, že instance třídy *Activity* může

obsahovat více Fragmentů a mít tedy několik nezávislých *layoutů* na jedné obrazovce.

Layout může být dodán instanci třídy *Activity* nebo *Fragment* v podobě odkazu k XML souboru nebo může být programově vytvořen. Aby mohl programátor referencovat objekty *View* definované v XML souboru, musí mít každý objekt nastavené své ID, které se uloží jako veřejná konstanta do třídy jménem *R*. Ve třídě *R* jsou také uloženy reference na XML layouty a další informace (například odkazy k obrázkům a textům).

2.2 Knihovna úkolů pro vytváření edukativních aplikací

Původní knihovna, implementovaná v prostředí *Android SDK*, je složena ze tří balíčků - *cz.scientica.tasks*, *cz.scientica.views* a *cz.scientica.inner.questionset*. Balíček *tasks* obsahuje třídy typu *Fragment*. Každá třída z tohoto balíčku reprezentuje nějaký druh úkolu - úkoly budou rozebrány v následujících odstavcích. Žádná z těchto tříd nemá definován vlastní *layout*, takže ho programátor musí dodat jako parametr konstruktoru ve formě reference k XML souboru. Nicméně velice často se předpokládá existence nějakých *View*, které by měl *layout* obsahovat (např. seznam tlačítek). Reference na tyto *View* musí být také předány konstruktoru. Jinak může být *layout* naprosto libovolný. Dále lze každé třídě nastavit 3 posluchače na události:

- *onResumeViewListener* při zobrazení Fragmentu,
- *onPauseViewListener* při skrytí Fragmentu a
- *onStateChangeListener* při změně stavu splněnosti Fragmentu.

Poslední posluchač je volán s parametrem typu *boolean*. Ten vyjadřuje, zda je úkol považovaný za splněný nebo nikoliv.

Balíček *views* obsahuje speciální třídy typu *View*, které byly nutné k vytvoření úkolů, ale nejsou součástí sady *View* z Android SDK.

Balíček *questionset* bude popsán v kapitole 2.2.7 - Automaticky generované testy.

Nyní budou popsány jednotlivé typy úkolů.

2.2.1 VideoFragment

Třída *VideoFragment* představuje jednoduchý fragment pro zobrazení videa. K zobrazení videa se používá třída *VideoView* z knihovny Android SDK. Zdroj videa může být jak lokální úložiště, tak úložiště dostupné přes HTTP protokol. Video Fragment může obsahovat libovolný počet videí. Při spuštění všech videí je notifikován *onStateChangeListener* s parametrem *true*.

2.2.2 *InfoClick* Fragment

Tento fragment obsahuje pole textů, pole referencí tlačítek a dvě pole referencí na obrázky. Všechna čtyři pole musí být stejně dlouhá. Při kliknutí na obrázek se zobrazí ten tentýž obrázek zvětšený na celou obrazovku. Jedno ze zmíněných polí referencí na obrázky obsahuje odkazy na miniatury obrázků a druhé pak na obrázky plné velikosti. Když uživatel klikne na tlačítko, zobrazí se přes miniaturu obrázku text,. Stejný text se

zobrazí také v levém dolním rohu. Pokud uživatel klikne na všechna tlačítka, je notifikován *onStateChangeListener* s parametrem *true*.

2.2.3 *DragDropQuestion* Fragment

Cílem tohoto úkolu je přetažení nějakých objektů (popisek, obrázek apod.) prstem na správné místo na obrazovce. Android SDK neobsahuje *View*, které by bylo možno po obrazovce libovolně prstem přetahovat. Balíček *views* proto obsahuje třídu jménem *DraggableTextView*, která právě toto umožňuje. Navíc notifikuje své posluchače při událostech začátek přetahování, posun po obrazovce, konec přetahování a může tak být použita k mnoho jiným účelům.

Fragment vyžaduje uvést jako parametr konstruktoru referenci na *layout*, referenci na tzv. *sample bitmap*, pole referencí na *DraggableTextView* objekty a pole barev. Obě pole musí být stejně dlouhá. *Sample bitmap* je obrázek, který má jako pozadí černou barvu a v popředí má několik tvarů různých barev. Tyto barvy musí odpovídat barvám v konstruktoru *Fragmentu*. *Sample* obrázek je součástí *layoutu*, nicméně je neviditelný. Podle pozice přetahovatelného *View* lze poznat, zda se nachází nad nějakou barvou *sample* obrázku a podle pořadí barev v poli se zároveň pozná, zda se jedná o „správnou" barvu.

Třída obsahuje posluchač, který lze navěsit na nějaké tlačítko, a který slouží k vyhodnocení úkolu.

2.2.4 *LinkPairQuestion* Fragment

V tomto úkolu jsou na obrazovce dvě skupiny *View* objektů a úkolem uživatele je pro každý objekt z jedné skupiny najít související z druhé skupiny a tyto spojit přetažením prstem. Při přetažení se na obrazovce kreslí čára.

Android SDK neobsahuje *View* s požadovanou funkcionalitou, proto je v balíčku *views* naimplementováno *View* jménem *CanvasView*. To obsahuje *OnTouchListener* pro sledování pohybu uživatelova prstu a zároveň je přes celou plochu *View canvas*, na který se čára vykresluje. Na *CanvasView* je možno navěsit posluchač, který je notifikován z metod *OnTouchListener* třídy.

Zmíněný posluchač využívá třída *LinkPairQuestion* k rozpoznání, zda uživatel spojil související *View* objekty.

2.2.5 *CheckBoxQuestion* Fragment

Této třídě musí být v konstruktoru poskytnuto pole referencí na *CheckBox* objekty, které musí uživatel při plnění úkolu zaškrtnout, a reference na *layou*t úkolu. Třída tento *layout* analyzuje a uloží si seznam všech *CheckBox* objektů, které v něm jsou. Při vyhodnocení jednoduše projede všechny objekty dodané v konstruktoru a zjistí, zda jsou zaškrtnuty. Zároveň prověří, zda není zaškrtnut nějaký *CheckBox*, který v seznamu z konstruktoru chybí.

2.2.6 *Information* Fragment

Information Fragment je nejjednodušší úkol ze všech. Účelem tohoto úkolu je předat

uživateli nějaké znalosti pasivní cestou, například formou textu nebo obrázku. Nepožaduje tak od uživatele žádnou interakci a nemá ani žádnou funkci pro vyhodnocení úkolu.

Třída nicméně, jak bylo popsáno v úvodu kapitoly, může mít naimplementovány posluchače *onResume* a *onPause* a tím být rozšířena. Posluchač *onStateChange* třída nepodporuje.

2.2.7 Automaticky generované testy

Tento typ úkolu nevyžaduje od programátora poskytnutí *layoutu* jako předchozí typy úkolů. Vyžaduje však poskytnutí XML souboru, jehož validita je ověřována proti DTD souboru *questions.dtd* ve složce *assets* knihovny úkolů.

Rodičovským elementem je *task* s povinným atributem *name*. Tento element obsahuje libovolný počet *questionslide* elementů. Element q*uestionslide* je základní element určující typ úkolu, neboť musí mít nastaven atribut *layout* (DTD soubor specifikuje povolené hodnoty atributu) definující *layout* úkolu. Jeho potomky jsou element *header* (nadpis úkolu) a element *questions*.

Element *questions* obsahuje alespoň jeden element *question*. Element *question* reprezentuje konkrétní úkol a obsahuje tři povinné atributy:

- *type*: typ úkolu je v DTD souboru určen seznamem možností (*fillText, singleChoice*).
- *shuffle*: očekává hodnotu typu *boolean*. V závislosti na uvedeném typu úkolu říká, zda promíchat seznam odpovědí (*type* je *singleChoice*) nebo zda promíchat jednotlivé otázky (*type* je *fillText*).
- *validate*: očekává hodnotu typu *boolean*. Pokud programátor nastaví *false*, úkol se nebude vyhodnocovat (tzn. automaticky je ve stavu splněný).

Element *question* má dále dva nepovinné potomky - *description* s popisem úkolu a *images* se seznamem obrázků (resp. referencí na ně) doplňujících úkol. A poté má alespoň jeden element *variant*.

Element *variant* obsahuje atribut *valid* očekávající *true* nebo *false*. Obsahem elementu je text s odpovědí. Při vyhodnocování úkolu program kontroluje, zda uživatel napsal/zaškrtnul hodnotu, která je uvedena v seznamu *variant* elementů s atributem *valid* nastaveným na *true*.

Podle nastaveného *layoutu* program určí, jakým způsobem vykreslí *header, images* a *question* elementy. Je programátorovou zodpovědností, aby použil takovou kombinaci elementů a atributů, aby ji mohl program vykreslit. Například použití elementu *images* v *layoutu*, který žádný obrázek neočekává je chybou (v tomto konkrétním případě se chyba neprojeví selháním aplikace, pouze nebude žádný obrázek vykreslen).

Pro větší názornost následuje ukázka XML souboru pro jeden z úkolů v aplikaci *Globální problémy z nadhledu*.

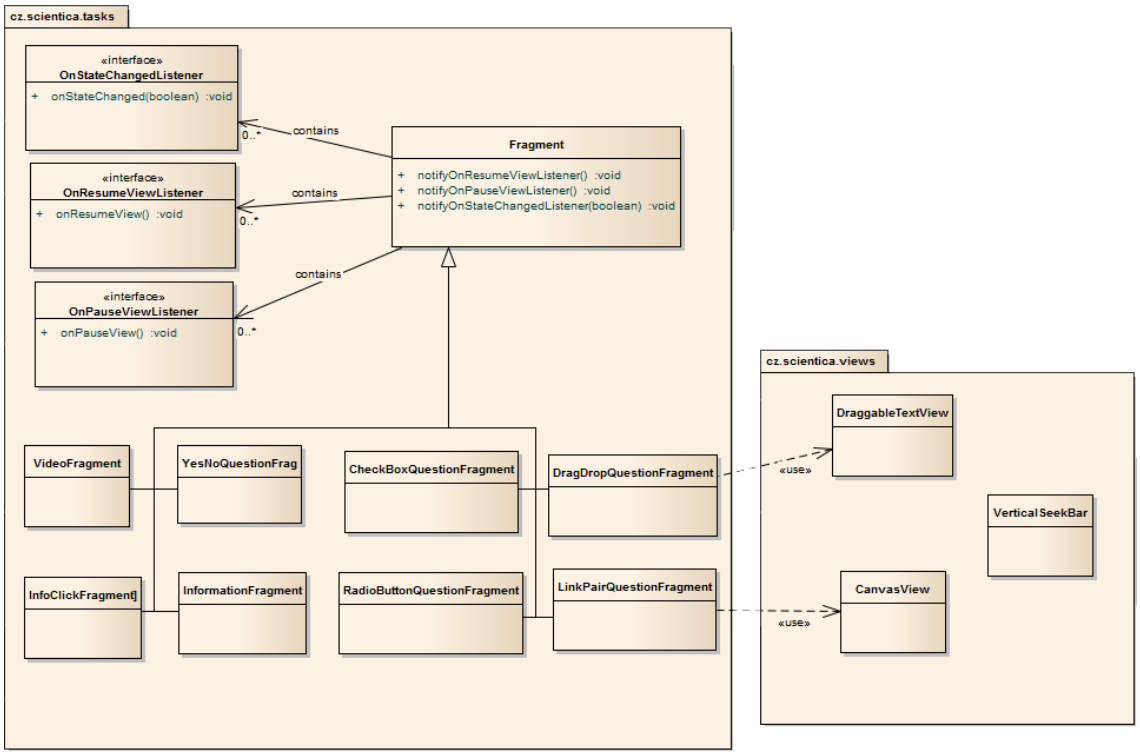
```
<questionslide name="klasika" layout="v_quest_img">
  <header><![CDATA[Part 1: Remote sensing<br>COMPLETE]]></header>
  <questions>
```

```
    <question type="fillText" shuffle="true" validate="true">
      <description>
Among the first Earth observers to use cameras were animals, to be
precise
      </description>
      <variant valid="true">pigeons</variant>
    </question>
    <question type="fillText" shuffle="true" validate="true">
      <description>
Besides visible radiation, instruments installed on a satellite also
record other types of radiation: ultraviolet and
      </description>
      <variant valid="true">infrared</variant>
    </question>
    <question type="fillText" shuffle="true" validate="true">
      <description>
To put a satellite into orbit, we need a
      </description>
      <variant valid="true">rocket</variant>
      <variant valid="true">rocket carrier</variant>
    </question>
    <question type="fillText" shuffle="true" validate="true">
      <description>
Rocket parts, defunct satellites and flakes of paint orbiting the
Earth are called space
      </description>
      <variant valid="true">junk</variant>
      <variant valid="true">debris</variant>
    </question>
    <question type="fillText" shuffle="true" validate="true">
      <description>
An orbit perpendicular to the equatorial plane and passing
approximately above the poles is called
      </description>
      <variant valid="true">polar</variant>
    </question>
  </questions>
</questionslide>
```

Z předchozího XML souboru program vyrobí *layout* se seznamem otázek a *EditText View* elementů, do kterých uživatel napíše správnou odpověď na otázku v elementu *description*. Otázky se nezobrazí v pořadí, ve kterém jsou zapsány v XML souboru, ale díky atributu *shuffle* se před vykreslením náhodně promíchají. Všechny obsahují atribut *validate* nastaven na *true*, takže uživatel musí na všechny odpovědět správně.

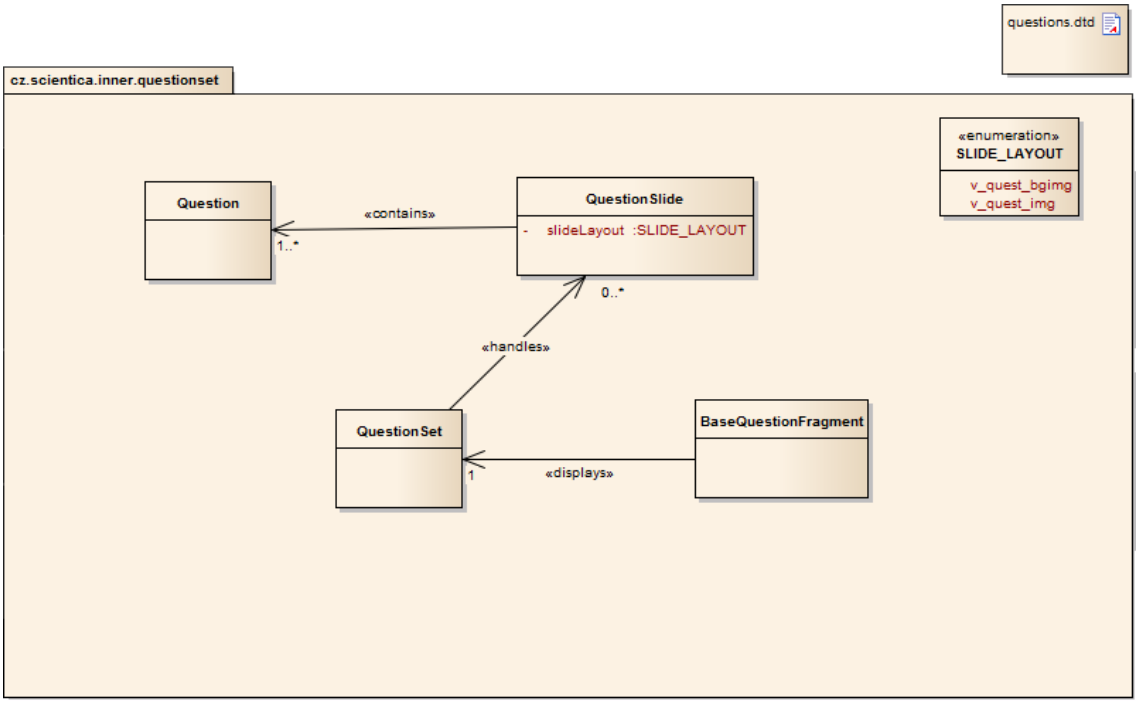
2.3 Závěr

V předchozím textu byly popsány jednotlivé třídy z knihovny úkolů. Pro přehlednost jsou v této části dva UML diagramy (obr. 2.1, obr 2.2) shrnující popsané třídy.



Obr. 2.1: Diagram tříd knihovny úkolů.

Na obrázku 2.1 je *View* jménem *VerticalSeekBar*, které v předchozím textu nebylo zmíněno. Toto *View* není využíváno žádnou výše popsanou třídou. Android SDK třídu *SeekBar* obsahuje, jedná se však pouze o horizontální variantu. Její využití bude popsáno v kapitole věnující se aplikaci *Globální problémy z nadhledu*.



Obr. 2.2: Diagram tříd reprezentující automaticky generované úkoly.

Obrázek 2.2 zobrazuje podstatné třídy spravující automaticky generované úkoly. Třída *QuestionSet* se stará o syntaktickou analýzu XML souboru a inicializuje potřebné třídy. Třída *BaseQuestionFragment* je potomkem třídy *Fragment* z knihovny Android SDK a vykresluje *layout* dodaný třídou *QuestionSet*. nicméně aplikace *Globální problémy z nadhledu* tuto třídu nevyužívá a místo ní definuje vlastní třídu.

3 ANALÝZA – MULTIPLATFORMNÍ MOBILNÍ FRAMEWORKY

Aplikace je multiplatformní pokud funguje na různých operačních systémech a je založena na jednom společném zdrojovém kódu [2]. Ve světě mobilních aplikací lze rozlišit v zásadě dva druhy multiplatformních aplikací - nativní a hybridní [3]. Tato kapitola se na oba dva druhy zaměřuje. Nejprve budou popsány aplikace nativní.

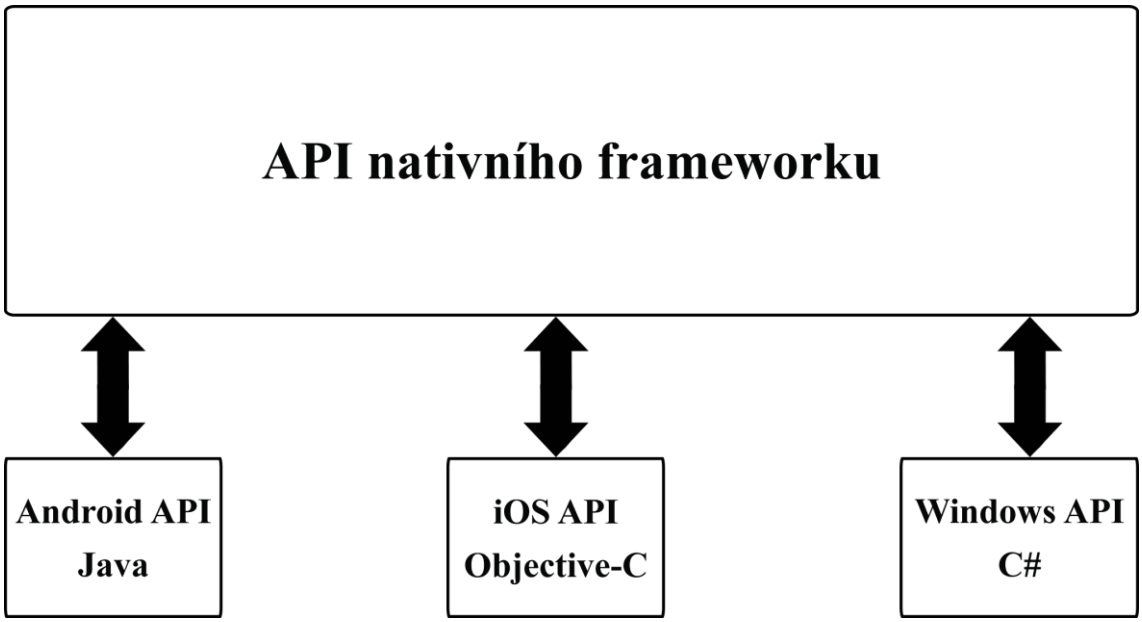
3.1 Nativní aplikace

Nativní aplikace je vyvíjena pomocí SDK, které poskytuje výrobce operačního systému a za použití preferovaného programovacího jazyka. To znamená, že pokud má nějaká aplikace fungovat na více OS, musí být několikrát implementována pro různé OS, což s sebou přináší komplikace při její údržbě, následném rozvoji a dodržení jednotné funkcionality a vzhledu na všech platformách.

3.2 Nativní multiplatformní aplikace

OS Android, iOS a další poskytují k vývoji aplikací své vlastní SDK, které preferují různé programovací jazyky (např. Java pro Android a Objective-C nebo Swift pro iOS). Zároveň poskytují API ke svým SDK, které lze použít libovolným programovacím jazykem.

Nativní multiplatformní aplikace využívá jedno API napsané v nějakém programovacím jazyce, který výrobce OS nemusí podporovat. Toto společné API je postaveno nad API jednotlivých výrobců operačních systémů. Výsledná aplikace je pak tvořena jedním zdrojovým kódem společným pro všechny operační systémy. Jelikož využívá nativní API výrobce, může dosahovat výkonu blížícímu se nativní aplikaci.



Obr. 3.1: Schéma nativní multiplatformní aplikace.

V následujících odstavcích jsou popsány některé populární nativní multiplatformní frameworky.

3.2.1 Xamarin

Xamarin byl vznikl v roce 2011 a je vlastněn společností Microsoft. Pro vývoj využívá programovací jazyk C#. Lze použít vývojové prostředí Xamarin Studio pro Mac nebo Visual Studio pro Windows. V těchto nástrojích lze jednoduše kreslit uživatelské rozhraní pomocí předpřipravených prvků nebo je psát ručně.

Xamarin nevyužívá programovací jazyk *JavaScript*, proto se mu tato práce podrobněji věnovat nebude.

3.2.2 React Native

React Native je *open source* framework od společnosti Facebook přímo vycházející z *frameworku React*, který slouží k vývoji internetových aplikací v jazyce *JavaScript*. Vzniknul v roce 2015, takže je zatím poměrně mladý a stále v intenzivním vývoji (v době psaní textu ve verzi 0.44). Každý měsíc vychází nová stabilní verze.

Stavebním kamenem *frameworku* jsou Komponenty (*Components*) a jejich zanořování. Komponenta má své vlastnosti (*props*), svůj stav (*state*) a *render* funkci. *Props* jsou vlastnosti, které jsou komponentě nastaveny rodičovskou komponentou a daná komponenta je může pouze číst, nikoliv měnit. *State* se naproti tomu definuje v konstruktoru komponenty a jeho vlastnosti lze měnit. *State* se mění voláním metody *setState* a změna tohoto stavu vynutí volání *render* funkce. Důležité je poznamenat, že *setState* je asynchronní funkce. Funkce *render* vykresluje *layout* komponenty. Tento *layout* je definován pomocí JSX - vlastní značkovací jazyk *frameworku*. Elementy JSX zhruba odpovídají nativním elementům (např. *View, Text, Button*) - každý element odpovídá jedné komponentě, jedná se tedy o strukturovaný zápis komponent. Rozložení

layoutu a vzhled jednotlivých komponent se nastavuje pomocí stylů. Jde v podstatě o *camelCase* variantu CSS 3.

Každá komponenta obsahuje instanční proměnnou *refs*. *Refs* je pole referencí na komponenty, s nimiž pracuje její *render* funkce. Aby bylo možné pomocí tohoto pole na komponenty odkazovat, musí mít odkazovaná komponenta v *render* funkci nastaven atribut *ref* na nějakou hodnotu (*ref* je typu text).

Stejně jako v Android SDK má *Activity* i *Fragment* svůj životní cyklus, tak i komponenta v *React Native* prochází životním cyklem [4].

Framework rozlišuje *index.android.js* a *index.ios.js* jako vstupní body aplikace pro jednotlivé OS. Stejně tak obsahuje složky *android* a *ios*, do kterých *React Native* generuje nativní kód.

Následující kroky jsou vykonány při každém startu *React Native* aplikace [5].

- Inicializace *JavaScript* VM a nativních modulů OS (*cache*, síť, *UI manager* atd.).
- Načtení minimalizovaného *JavaScript* balíčku do *JavaScript* VM, který provede jeho syntaktickou analýzu a vygeneruje bajt kód.
- Načtení aplikace z *cache* paměti.
- Inicializace *React* komponent a jejich poslání do UI manažeru daného OS.
- Výpočet rozměrů objektů v tzv. *shadow* vlákně a jejich následná inicializace a vykreslení ve vlákně hlavním.

V předchozím seznamu byly použity termíny jako *shadow* vlákno a hlavní vlákno. Ve skutečnosti *React Native* využívá tři vlákna [6]:

- *Shadow* vlákno starající se o výpočet rozměrů *layoutů*.
- Hlavní vlákno starající se o vykreslování aplikace.
- *JavaScript* vlákno, ve kterém běží *JavaScript* kód.

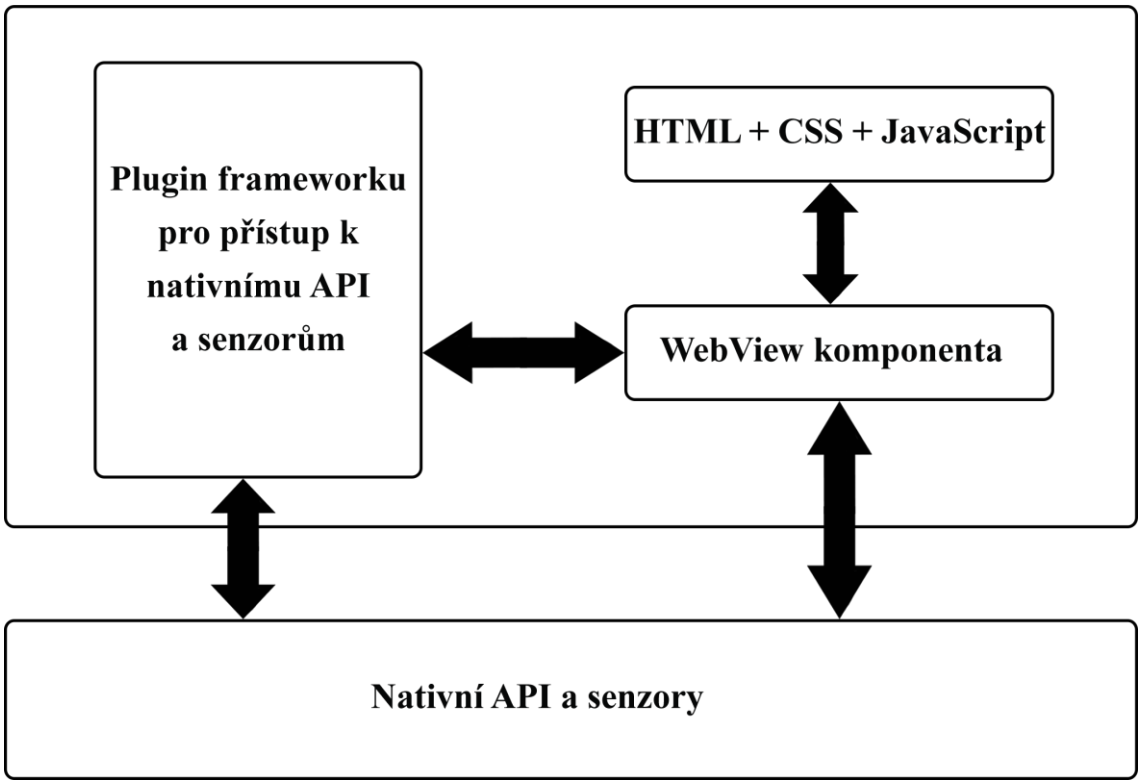
Pro rychlý vývoj aplikací disponuje *framework* NodeJS serverem jménem *React Packager*, který se stará o překlad kódu, a který dokáže detekovat, jakou část kódu programátor modifikoval a podle toho znovu přeložit jen onu změněnou část kódu. Programátor tak během několika vteřin vidí změny přímo v aplikaci běžící na simulátoru nebo v cílovém zařízení.

3.3 Hybridní multiplatformní aplikace

SDK jednotlivých mobilních operačních systémů obsahují komponentu *WebView* (např. v iOS je pojmenována *UIWebView*), což je v jednoduchosti komponenta pro vykreslování webového obsahu (HTML, CSS, JavaScript).

Hybridní multiplatformní framework využívá tuto nativní komponentu k vykreslování aplikace, která je implementována v podstatě jako internetová aplikace. K přístupu například k fotoaparátu, internímu úložišti zařízení a dalším funkcím se používají funkce *frameworku*.

Vzhledem k tomu, že celá aplikace funguje v rámci *WebView* nativního API, nemůže se výkonem srovnávat s aplikací nativní a jen velice těžko dosáhne UX nativní aplikace.

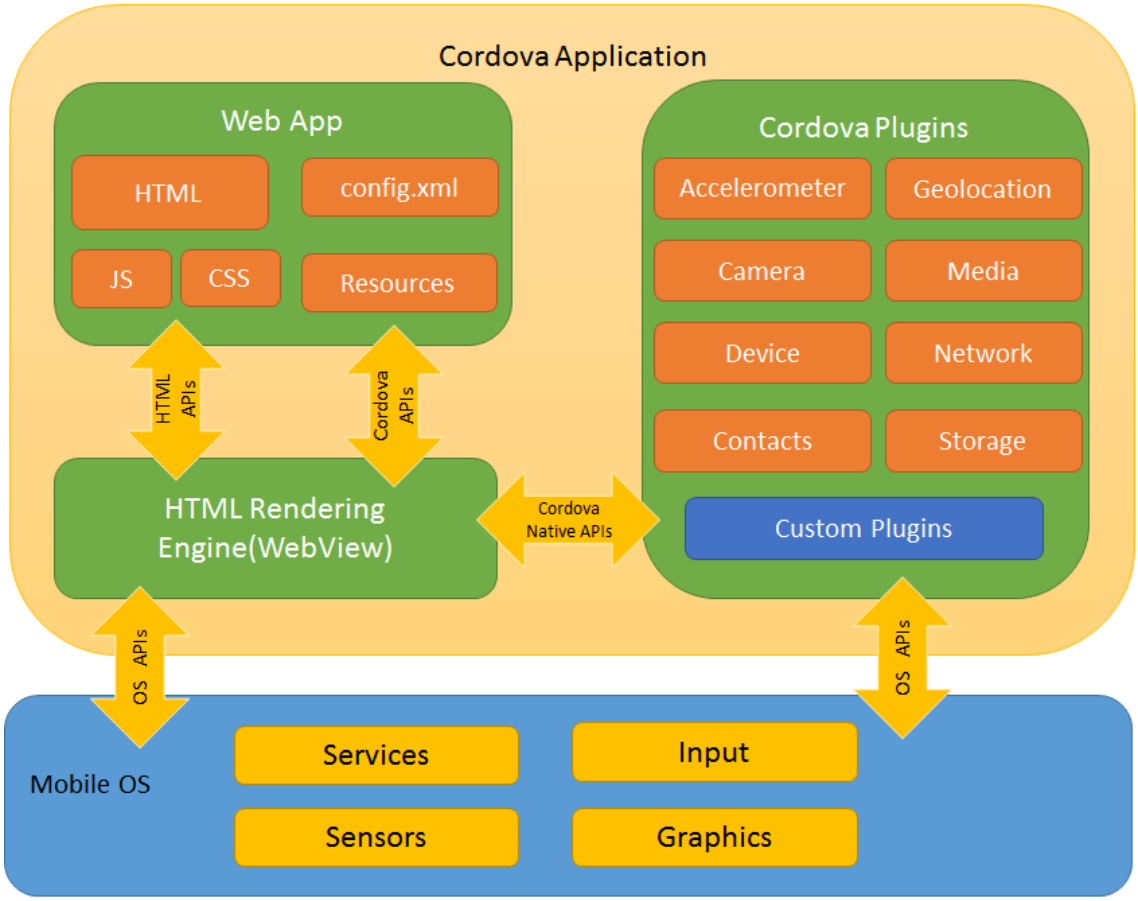


Obr. 3.2: Schéma hybridní multiplatformní aplikace.

V následujících odstavcích jsou rozebrány nejpopulárnější hybridní multiplatformní frameworky.

3.3.1 Apache Cordova

Cordova od společnosti Apache je *open source* zástupce hybridních multiplatformních *frameworků*. Aplikace se píše jako klasická internetová stránka za pomoci HTML, CSS a *JavaScript*. Vše se vykresluje ve *WebView* a pro komunikaci s nativními funkcemi se využívají jednotlivé *Cordova* doplňky. Obrázek 3.3 zobrazuje architekturu *frameworku* - jsou na něm vidět doplňky (*Cordova Plugins*) *frameworku*, *WebView* i komunikace mezi *Cordovou* a nativní API.



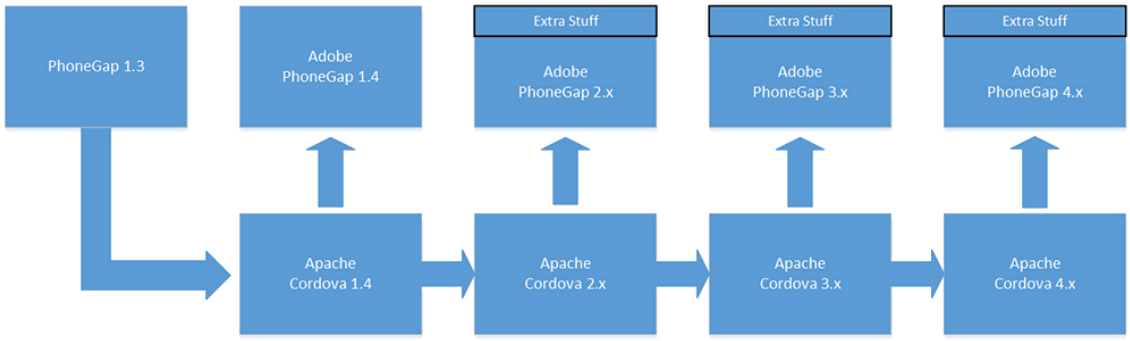
Obr. 3.3: Schéma *frameworku* Cordova z oficiálních stránek výrobce [10].

Funkce *frameworku* se dají jednoduše používat pomocí CLI (*command-line interface*). Pomocí něj se také doinstalují doplňky k používání nativních API.

3.3.2 Adobe PhoneGap

Adobe *PhoneGap* je hybridní *framework*, jenž je de facto distribucí Apache *Cordova frameworku*. Vztah těchto dvou *frameworků* je nicméně složitější [11]. Oba projekty vycházejí z původního *open source* projektu jménem *PhoneGap* od společnosti Nitobi. V roce 2011 přešla společnost Nitobi pod Adobe, avšak krátce předtím Nitobi projekt *PhoneGap* věnovala společnosti Apache Software Foundation. Společnost Apache projekt po čase přejmenovala na *Cordova*. Nitobi, již pod společností Adobe, vydala distribuci *Cordovi* jménem Adobe *PhoneGap* v době, kdy byl projekt *Cordova* ve verzi 1.4.

Když byl projekt *Cordova* ve verzi 2.0, byla z něj opět vydána distribuce Adobe *PhoneGap*, která zahrnovala jistá rozšíření. Tímto způsobem proces pokračuje i nadále. Celý výše popsaný vývoj projektů shrnuje obrázek 3.4.



Obr. 3.4: Vývoj projektů Adobe *PhoneGap* a Apache *Cordova* [11].

3.3.3 Ionic

Ionic je *open source* HTML 5 *framework* implementovaný v *AngularJS*. Poskytuje uživatelské prvky nativních aplikací, což je výhodné v případě implementace klasické aplikace, která dodržuje nativní *look & feel* jednotlivých výrobců OS. Velkou výhodou při implementaci pomocí tohoto *frameworku* je také fakt, že lze aplikaci zdarma nahrát na *cloud* výrobce. V obchodech s aplikacemi u Android i iOS je ke stažení aplikace *Ionic View*, pomocí které je možné aplikace z *cloudu* prohlížet a jednoduše tak testovat na cílových zařízeních.

Nicméně jde pouze o HTML / *JavaScript framework*, takže ke spuštění potřebuje nějaký tzv. *native wrapper*, jako například výše zmíněnou *Apache Cordovu*. V nestandardním uživatelském rozhraní, jako v případě aplikace *Globální problémy z nadhledu,* by implementace s *Ionic* byla jen o málo snazší, než pouze v *Apache Corvoda* kombinaci s *AngularJS*.

3.3.4 OnsenUI

OnsenUI je stejně jako *Ionic* HTML 5 *framework,* má *cloud*, do kterého lze nahrávat aplikace, ale na rozdíl od *Ionic* si zde programátor může vybrat, zda použije *AngularJS* nebo *React* coby JS *framework*.

Je také poměrně mladý, verze 1.1 byla vydána v říjnu 2015.

3.4 Závěr

V této kapitole byly popsány nativní a hybridní mobilní multiplatformní *frameworky*. Velká výhoda hybridních spočívá v možnosti využití znalostí z vývoje internetových stránek. Další výhodou je fakt, že hybridních *frameworků* je nepřeberné množství. Nicméně fungují coby nadstavba *frameworků Apache Cordova* / *Adobe PhoneGap*. Konkrétně byly zmíněny dva - *Ionic* a *OnsenUI*. Pro úplnost je zde ještě uveden již pouze výčet dalších, které mohou být použity jako ekvivalent například k populárnímu *Ionic frameworku*.

- Intel XDK
- Sencha Touch

- Kendo UI
- Framework 7
- JQuery Mobile
- Mobile Angular UI
- Famo.us
- Monaca

Nevýhodou naprosto všech hybridních *frameworků* je jejich fungování v rámci *WebView*, díky čemuž klesá výkon aplikace a jen velice těžko lze dosáhnout UX nativní aplikace. Na druhou stranu OS iOS i Android svá *WebView* neustále vylepšují, čímž prakticky zvyšují výkon hybridních aplikací [12].

Operační systém iOS používal do verze 7 komponentu *UIWebView*, která nepoužívala výkonnostní vylepšení, které měl v té době prohlížeč *Safari*. Ve verzi 8 přešla na komponentu *WKWebView*, která tato vylepšení obsahovala a tím výrazně zvýšila výkon hybridních aplikací [13].

Zařízení Android výkonnostně dohnalo prohlížeč Google *Chrome* až ve verzi 4.4 tím, že přešlo na *Chromium* JS *engine* [14]. Od verze 5.0 může uživatel automaticky aktualizovat svou *WebView* komponentu přes obchod s aplikacemi *Google Play*, a tím využívat nejnovější úpravy této komponenty [15].

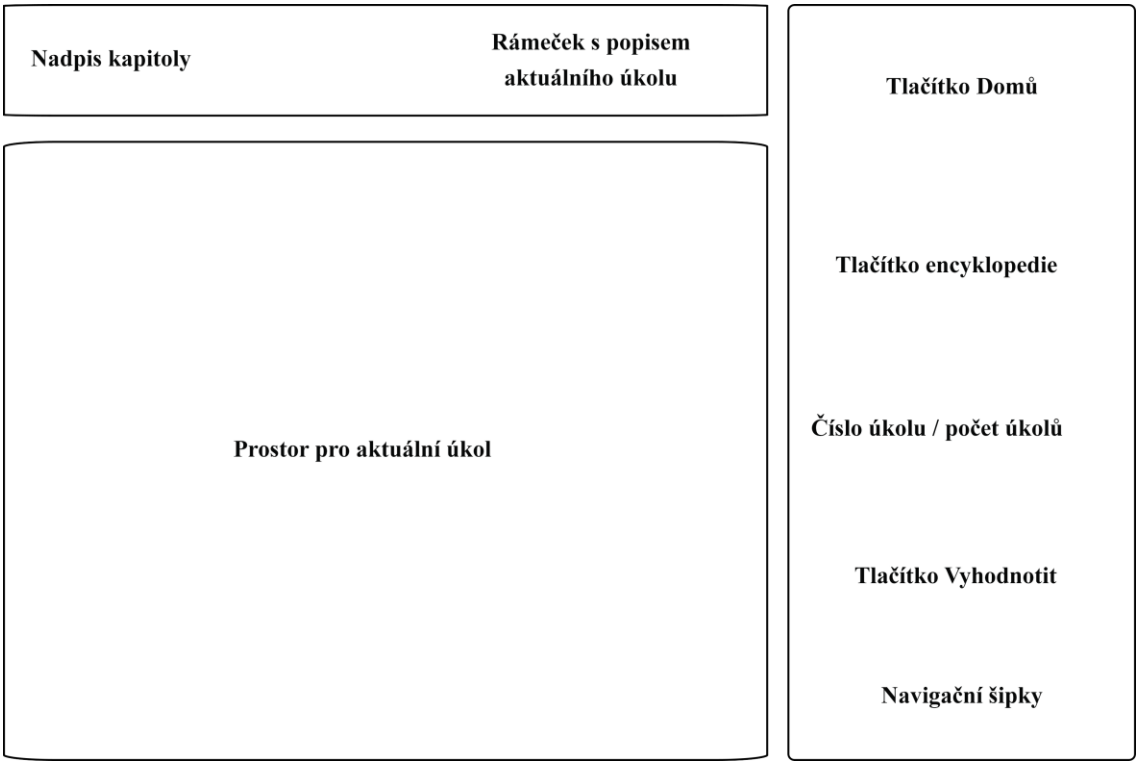
V nativních multiplatformních *frameworcích* je nevýhodou nutnost naučit se novému API *frameworku*. *React Native* je moderní *open source framework* postavený na jazyce *JavaScript*. Používá značkovací jazyk JSX, který obsahuje nativní uživatelské prvky pro Android i iOS. Stejně tak i výkonem by se aplikace měla blížit nativní [16]. Nicméně jeho nevýhodou je právě onen dynamický rozvoj spojený s jeho nevyzrálostí a faktem, že se verze *frameworku* mění každý měsíc.

Na základě analýzy výše popsaných *frameworků* by bylo dle autora textu nejoptimálnější zvolit jako nástroj k implementaci klasické multiplatformní mobilní aplikace *Ionic* v kombinaci například s *Apache Cordova* nebo v případě jednoduché aplikace, či nové služby do stávající nativní aplikace *React Native*. Aplikace *Globální problémy z nadhledu* nedodržuje *look & feel* standardy jednotlivých OS a má často velice specifické rozložení prvků na stránce. Multiplatformní knihovna úkolů pro vývoj výukových aplikací svými vlastnosti předpokládá, že aplikace, které ji budou využívat budou moci mít vzhled podobný hrám.

V následující kapitole budou proto představeny dva prototypy - jeden postaven na *React Native* a druhý na *Apache Cordova*. Teprve na základě vzniklých prototypů bude vybrán *framework* pro výslednou implementaci.

4 *GLOBÁLNÍ PROBLÉMY Z NADHLEDU* – REACT NATIVE VS. APACHE CORDOVA

Kapitola je zaměřena na základní implementaci rozložení obrazovky aplikace *Globální problémy* z nadhledu tak, jak je vidět na obrázku 1.1 v *React Native* a v *Apache Cordova* a na některé další možné implementační problémy aplikace. Podle jednoduchosti vývoje se rozhodne, zda použít *framework React Native* nebo *Apache Cordova*.



Obr. 4.1: Rozložení obrazovky aplikace *Globální problémy z nadhledu*.

Veškerá tlačítka v pravém rámečku obrázku 4.1 jsou realizována obrázkem. Aplikace by měla fungovat na tabletech od úhlopříčky 7" a různých rozlišení obrazovky (počet pixelů na palec čtverečný).

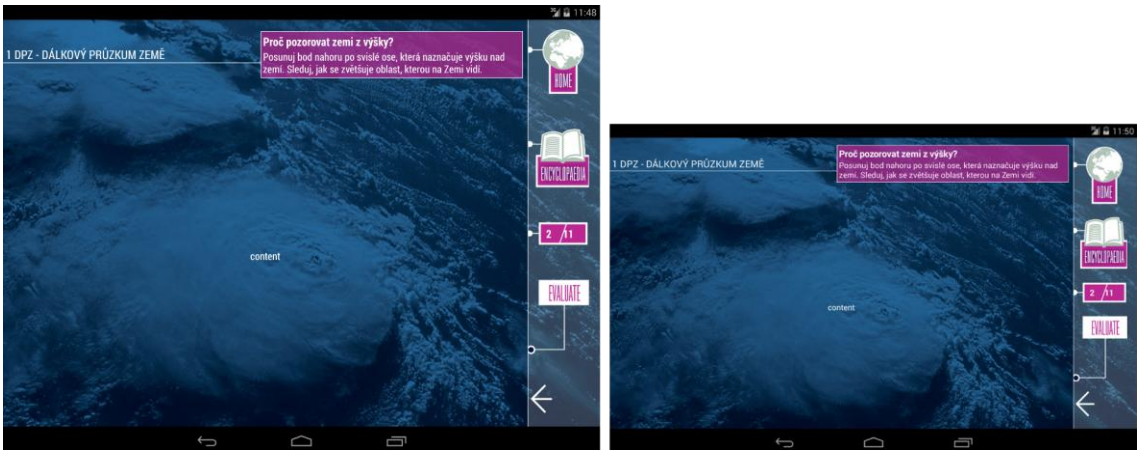
4.1 React Native

React Native poskytuje ve svých stylech podporu pro *Flexbox*, což je v jednoduchosti způsob rozložení prvků na obrazovce do řádků a sloupců, přičemž rozdělení prostoru pro jednotlivé prvky je možno určit relativně dle procent (kromě jiných způsobů). Navíc lze určit, jak se jednotlivé objekty v rámci *Flexbox* objektu rozmístí. Například v prototypu, který je vidět na obrázcích této kapitoly, je použita volba *space-between*, což

znamená, že mezery mezi objekty budou všude stejně velké.

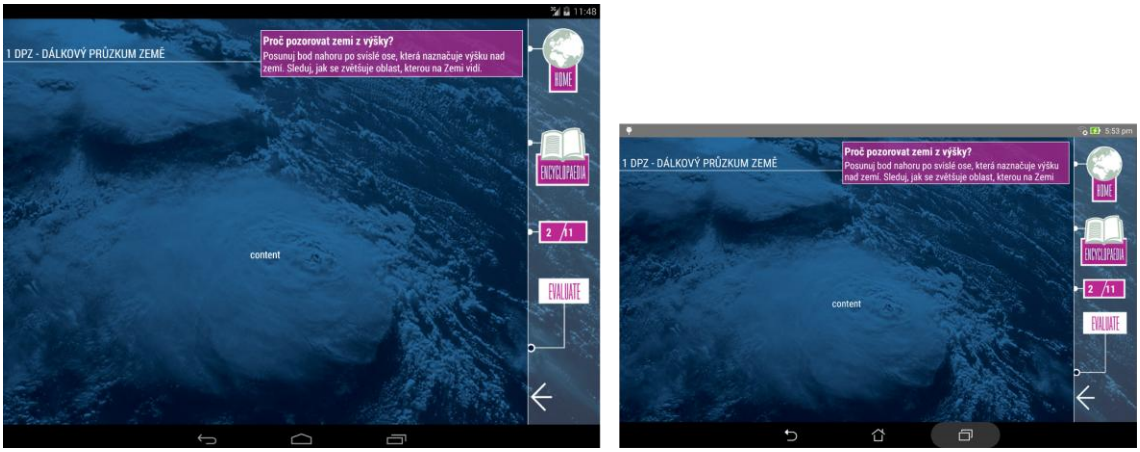
Další velkou výhodou je možnost přidávat do zdrojových obrázků informaci o tom, pro jaké rozlišení jsou - stačí do názvu obrázku před jeho koncovku přidat *@2x* , *@1.5x* apod. Nicméně v implementaci je nutno obrázky ještě škálovat kvůli různým rozměrům obrazovky. Tohoto efektu je dosaženo implementací vlastní komponenty *ResizableImage*, která při načtení obrazovky zjistí velikost obrazovky, vydělí ji referenční hodnotou a tímto poměrem vynásobí velikost obrázku. Tento přístup přináší velice dobré výsledky jak na Android zařízeních, tak i na iOS.

Stejně tak se musí velikosti obrazovky přizpůsobit i velikost textu. Toho je v prototypu dosaženo velice podobným způsobem, ale výsledek se rozdělí do několika intervalů a v nich se velikost textu násobí určitou hodnotou. Implementace je inspirována odpovědí na serveru *Stack Overflow* [19]. Na obrázcích Obr. 4.2 až Obr. 4.4 je vidět, že tento přístup se zdá být správný, nicméně pro úplně správný výsledek se ještě musí mírně upravit konstanty.



Obr. 4.2: Zleva prototyp aplikace na emulátorech Nexus 9 a Nexus 7 (*React Native*).

Na obrázku 4.3 je stejná aplikace jako na obr. 4.2 na skutečných zařízeních Galaxy Tab S2 a Asus MeMO Pad 7, která jsou svými parametry velice podobná emulovaným zařízením z obrázku 4.2.



Obr. 4.3: Zleva prototyp aplikace na Galaxy Tab S2 a Asus MeMO Pad 7 (*React Native*).

Na obrázku 4.4 je prototyp pro zařízení iOS, konkrétně iPad Pro 9.7" a iPad Pro 12.9". Mezi prototypy pro Android a iOS jsou různé drobné implementační rozdíly, které jsou popsány v následujících odstavcích.

Obr. 4.4: Zleva prototyp aplikace na iPad Pro 12.9" a iPad Pro 9.7" (*React Native*).

Mezi snímky obrazovky z Android a iOS je patrno několik rozdílů. Především ikony v pravém sloupci obsahují vlevo kolečko, které má překrývat svislý pruh. Toho je dosaženo použitím stylu obrázku *overflow* nastaveného na *visible*. V OS Android však tato metoda nefunguje, neboť prostor pro vykreslení *View* je zcela určen prostorem svého rodiče. Viz například *issue* [20].

Další vcelku očekávaný rozdíl je použití nekompatibilních typů písma u jednotlivých OS. V Androidu je použit *sans-serif-condensed*, zatímco iOS používá typ písma *HelveticaNeue*. Tato písma jsou v aplikaci také nastavena, nicméně je pouze na vůli programátora, jaký typ písma použije.

Zajímavým rozdílem je také vykreslení pozadí komponenty *Text*. Ten podle specifikace dědí barvu pozadí své rodičovské komponenty. Když je v OS Android rodičem obrázek a *Text* nemá pevně nastavenou barvu pozadí, výsledná barva bude průhledná. V operačním systému iOS to ovšem bude barva předka, která, není-li nastavena, je bílá.

Aplikace je vícejazyčná, takže potřebujeme podporu pro tuto funkci. *React Native* jí přímo nedisponuje, nicméně lze jí přidat, přičemž je možné se inspirovat projektem *ReactNativeLocalization* na GitHub [17].

V aplikaci se také poměrně často vyskytuje typ úkolu, kde se má nějaký objekt prstem přetáhnout na určené místo. *React Native* k tomuto účelu poskytuje poměrně složitý systém *Gesture Responder System*. Implementovaný prototyp má komponentu jménem *DraggableView*, která přetahování podporuje. Nicméně prototyp ještě nemá implementovanou funkci přetahování na konkrétní předem určená místa a vyhodnocování správnosti přetažení.

4.2 Apache Cordova

Stejně jako v *React Native*, i v *Apache Cordova* lze použít *Flexbox* pro implementaci rozložení obrazovky. Nicméně její funkčnost záleží na tom, zda je *Flexbox* podporován *WebView* cílového zařízení.

V případě implementace za pomoci *WebView* je nutno (vhodné) zvážit použití nějakého *JavaScript frameworku*. Zde se opět nabízí *React* nebo také *AngularJS*, což je velice pokročilý *framework* pro vývoj dynamických webových aplikací. S oběma přístupy nicméně přicházejí podobné problémy, které byly zmíněny v předchozí kapitole (implementace přetahovatelných objektů je jednodušší, neboť lze využít klasické *JavaScript* metody *mousemove, mousedown* [18]).

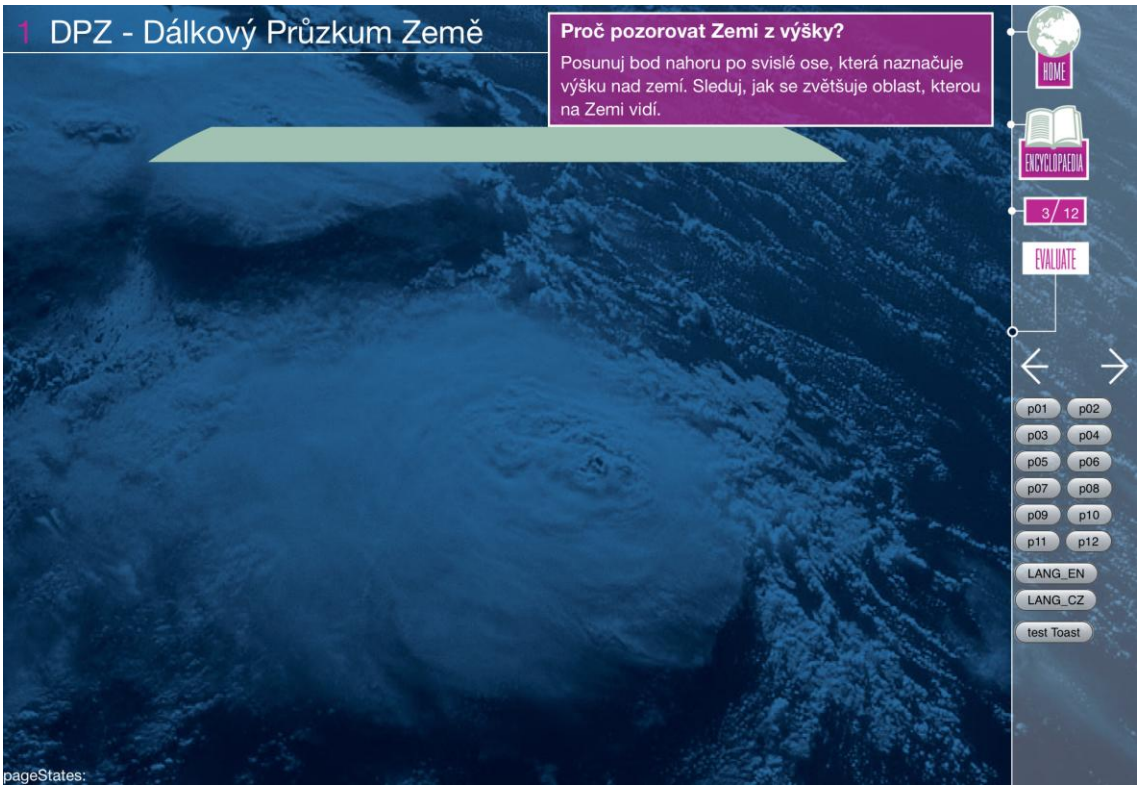
Hlavní nesnáze související s použitím *Apache Cordova* jsou však spjaté s faktem, že podobně jako každý internetový prohlížeč, tak i každé *WebView* může různě interpretovat HTML a CSS.

Obr. 4.5: Zleva prototyp aplikace na Galaxy Tab S2 a Asus MeMO Pad 7 (*Apache Cordova*).

Obr. 4.5 zobrazuje prototyp na 10" a 7" obrazovce zařízení Galaxy Tab S2 a Asus MeMO Pad 7. Prototyp je implementován s použitím *frameworku* AngularJS 1.0. Škálování obrázků se řeší za použití vlastní *directive Resizable*, která se používá u obrázků. Velikost obrázku se spočítá jako součin původní velikosti s konstantou (rozměr zařízení / 2200). Číslo 2200 zhruba představuje referenční šířku obrazovky, podle které se ostatní přizpůsobují. Z obrázku je patrné, že tento přístup funguje jen

částečně - také kvůli různým poměrům stran obrazovek.

Na Obr. 4.6 je snímek obrazovky stejné implementace jako pro Android na zařízení iPad 2. Jde o názornou ukázku různého vykreslování jednotlivých *WebView*, neboť výsledek měl být stejný jako na obrázku Obr. 4.5. Stejně tak, jako v případě implementace v *React Native*, se i zde nevyhneme rozdílným částem kódu pro oba operační systémy.



Obr. 4.6: Prototyp aplikace na zařízení iPad 2 (*Apache Cordova*).

4.3 Závěr

Z předchozí analýzy a z implementace prototypů jak v *React Native*, tak i v *Apache Cordova* se autor textu přiklání k použití *React Native*. Hlavním důvodem pro toto rozhodnutí se jeví fakt, že chování výsledné aplikace není určeno verzí *WebView* komponenty, ale *hardwarem* zařízení a verzí OS stejně tak, jako je tomu při nativním vývoji. Dalším důvodem je větší plynulost interakce při použití *ScrollView* elementů, přetahovatelných elementů a *SeekBar* elementů.

5 NÁVRH

V této kapitole budou nejprve navrženy *React Native* komponenty reprezentující typy úkolů z kapitoly 2 a poté bude navržena samotná aplikace *Globální problémy z nadhledu*.

První části kapitoly rozebírají funkční a nefunkční požadavky. Funkční požadavky vycházejí z požadavků na původní aplikaci *Globální problémy z nadhledu*. První čtyři nefunkční požadavky jsou stanovené vedoucím práce, zbylé si definoval autor práce na základě analýzy multiplatformních *frameworků*.

5.1 Nefunkční (obecné) požadavky

Následuje výčet nefunkčních požadavků.

- Operační systém Android bude verze alespoň 4.1 (API 16).
- Operační systém iOS bude verze alespoň 8.
- Velikost zařízení bude minimálně 7".
- Aplikace *Globální problémy z nadhledu* bude navržena pro tzv. *landscape* mód (netýká se knihovny úkolů).
- Implementace bude využívat *framework React Native* verze 0.44.
- Implementace bude ve skriptovacím jazyce *EcmaScript 2016* [22].

5.2 Funkční požadavky

Následuje výčet funkčních požadavků.

- Aplikace *Globální problémy z nadhledu* bude implementována v českém a anglickém jazyce.
- Aplikace *Globální problémy z nadhledu* bude ukládat stav splněnosti úkolů do paměti zařízení a při znovuotevření aplikace se stav z paměti načte.
- Uživatel bude mít možnost uložená data z paměti vymazat prostřednictvím aplikace *Globální problémy z nadhledu*.
- Aplikace *Globální problémy z nadhledu* bude obsahovat 9 kapitol, v každé kapitole bude několik úkolů, které bude uživatel postupně plnit.
- Na další úkol bude uživatel moci přejít pouze po splnění stávajícího úkolu.
- Dále bude v každém úkolu možnost aktivovat tzv. *Debug Mode*, který odemkne průchod úkolů v celé aplikaci a uživatel tak nebude muset úkoly plnit.
- Z každého úkolu se uživatel může vrátit do úvodní obrazovky, která

obsahuje seznam kapitol.

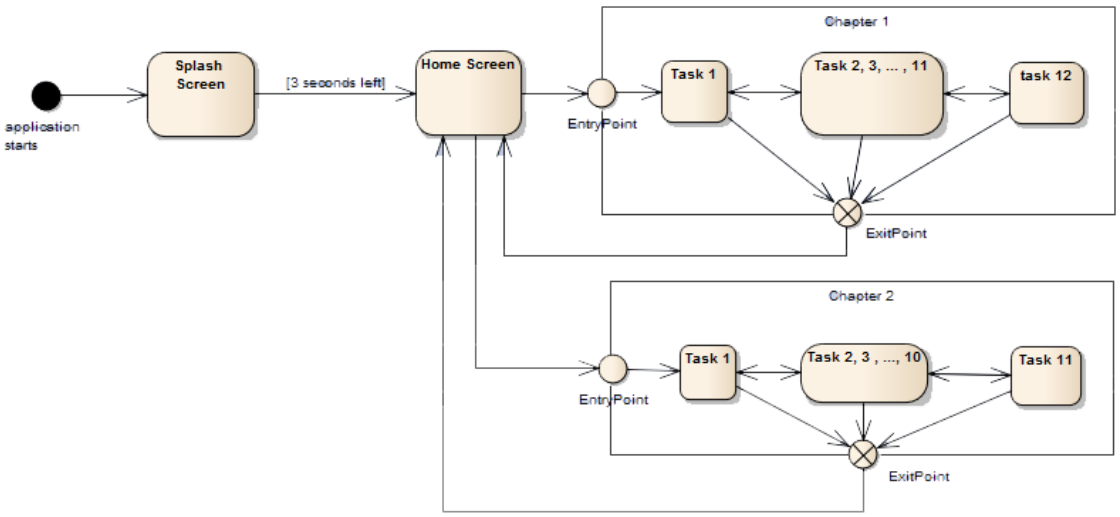
5.3 Knihovna úkolů pro tvorbu výukových aplikací

V kapitole 2 je již požadovaná funkcionalita popsána, stejně tak jsou tam popsány i názvy tříd v jazyce *Java*, které požadovanou ji splňují. Knihovna implementovaná v rámci této práce bude požadovanou funkcionalitu také splňovat. Její implementační detaily budou podrobně řešeny v kapitole implementace.

5.4 Aplikace *Globální problémy z nadhledu*

V kapitole 4 je obrázek znázorňující navržení *layoutu* (Obr. 4.1) pro původní aplikaci implementovanou pro OS Android. Nově vzniklá aplikace bude samozřejmě toto rozvržení splňovat. Vedoucí práce poskytl autorovi této diplomové práce veškeré potřebné texty a obrázky, původní kód v jazyce *Java* i fungující aplikaci na zařízeních Samsung Galaxy Tab 10" a Asus Zen Pad 7". Podrobný návrh aplikace byl záměrně popsán v kapitole 2, neboť autor považoval za nutné, aby byl čtenář s aplikací seznámen ještě předtím, než se přikročí k implementaci prototypů (kapitola 4).

V kapitole 1 je napsáno, že se aplikace dělí do jednotlivých kapitol a každá z nich obsahuje sadu úkolů.



Obr. 5.1: Obrazovky aplikace *Globální problémy z nadhledu* a možný průchod mezi nimi.

Na obrázku Obr. 5.1 jsou zaznamenány obrazovky aplikace a jsou spojeny šipkami ve směru, ve kterém se uživatel může navigovat. Obr. 5.1 nepokrývá celou aplikaci, neboť základních kapitol je celkem 6. Navíc aplikace obsahuje kapitoly *Test* a *Obrázkové hádanky* a jednu obrazovku, na které jsou HTTP odkazy na další studijní

materiály. Zanesení těchto obrazovek do obrázku Obr. 5.1 by dle názoru autora nepřineslo žádnou přidanou hodnotu, pouze by výsledný obrázek byl nepřehledný. Z toho důvodu obsahuje obrázek jen první dvě kapitoly.

6 IMPLEMENTACE

Kapitola v úvodní části popisuje technologie potřebné k vývoji *softwaru* ve *frameworku React Native*. Poté popisuje implementaci knihovny úkolů pro tvorbu výukových aplikací, implementaci aplikace *Globální problémy z nadhledu* a integraci knihovny úkolů do této aplikace.

6.1 Použité technologie

React Native ke svému chodu potřebuje Node.js, Python2, *React Native* CLI, JDK, Android Studio a Xcode (pochopitelně JDK a Android Studio je nutnost při vývoji Android aplikací a Xcode pro vývoj iOS aplikací) [9].

Za zmínku stojí fakt, že vyvíjet aplikace pro iOS není možno na počítači s operačním systémem jiným než iOS.

React Native je *open source* projekt, existuje tedy celá řada modulů implementovaných komunitou vývojářů. V případě nutnosti lze tyto moduly do projektu přidat buď manuálně nebo automaticky za použití balíčkovacího manažeru NPM.

6.1.1 Framework pro správu stavu aplikace

V popisu *frameworku React Native* bylo zmíněno, že každá komponenta má svůj stav (tzv. *state*), a že změna tohoto stavu vynutí volání *render* funkce. Pro komponentu je jednoduché změnit stav nějaké komponenty, jež je jejím potomkem (např. této komponentě nastavit *ref* a přistupovat k ní přes tuto hodnotu). Ale co když aplikace potřebuje změnit stav své rodičovské komponenty nebo si pamatovat tento stav i při vymazání komponenty z DOM aplikace a při znovunačtení ho obnovit?

Přesně pro tuto funkcionalitu existují tzv. *State Management frameworky*. Aplikace postavené na *React* nebo *React Native* obvykle využívají *Redux* nebo *MobX framework*. Oba *frameworky* umí v podstatě stejné věci, navíc je možno v rámci jedné aplikace oba zkombinovat.

V obou *frameworcích* existují tzv. *store* třídy. Tyto třídy obsahují proměnné, které reprezentují stav. Komponenta může *store* třídu využít tak, že její instanční proměnné používá v *render* funkci místo lokálního *state*. Změna stavu proměnné ve *store* vynutí volání *render* funkce stejně, jako kdyby byl změněn *state* komponenty. Výhoda spočívá v tom, že stav je oddělen od komponenty a lze k němu přistupovat z celé aplikace. Je tedy důležité identifikovat vlastnosti komponenty, které pro stav aplikace nemají význam a ty ponechat jako *state* dané komponenty a dále vlastnosti, které ovlivňují celou aplikace a ty vyextrahovat do *store* tříd použitého *State Management frameworku* (jako příklad lze uvést stav splněnosti úkolu v aplikaci *Globální problémy z nadhledu*, který určitě patří do *store*. Na druhou stranu například stav, který reprezentuje viditelnost / neviditelnost nějakého tlačítka na obrazovce pravděpodobně patří spíše do *state*).

Autor práce si k implementaci vybral *framework MobX*.

6.2 Knihovna úkolů pro tvorbu výukových aplikací

Ke tvorbě knihovny lze použít *React* Native CLI příkaz *react-native new-library*. Po tomto příkazu musí programátor udělat ještě mnoho práce, než může komponenty knihovny skutečně začít vyrábět. Veškou tuto práci udělá automaticky *open source* projekt *react-native-create-library* [23], který lze nainstalovat pomocí NPM.

Po vykonání konzolového příkazu `react-native-create-library EduLibrary` se automaticky vytvoří všechny potřebné složky a soubory a vývoj knihovny jménem *EduLibrary* může začít.

Dále jsou popsány implementační detaily jednotlivých typů úkolů z kapitoly 1. Závěrem je celá knihovna přehledně shrnuta diagramem tříd.

6.2.1 ResizableTextView, ResizableImage, ImageButton

Jak již bylo popsáno v kapitole 4, komponenty *ResizableTextView* a *ResizableImage* jsou téměř klasické *React Native* komponenty *Text* a *Image*, pouze přizpůsobují svou velikost velikosti obrazovky. Škálování obrázků dle rozlišení obrazovky dělá *ReactNative* sám, čímž usnadňuje práci.

Komponenta *ImageButton* z nějakého důvodu ve *frameworku* chybí, proto je vytvořena v rámci této knihovny. Obsahuje následující *props* vlastnosti.

- src: reference k obrázku,
- srcPressed: reference k obrázku v případě, že uživatel na komponentu klikl,
- srcDisabled: reference obrázku, pokud je ve stavu *disabled*,
- pressFunction: funkce, která je volána při kliknutí,
- disabled: příznak znamenající, zda je komponenta aktivní či nikoliv.

6.2.2 Video

Tento typ úkolu je reprezentován komponentou *VideoComponent*, jejíž *render* funkce vrací *Video* komponentu z *open source* projektu *react-native-video* [24] a *progress bar*. Zdrojem videa může být cesta k lokálnímu souboru nebo URL. Podporovaný formát videa je mp4. Důležité je při události komponenty *componentWillUnmount* (tzn. když se komponenta maže z DOM *layoutu*) vypnout přehrávání, neboť v projektu [24] je na toto téma otevřené *issue*.

6.2.3 InfoClick

Tento typ úkolu je složen ze dvou komponent. První komponentou je *ClickInfoView*, jejíž *render* funkce vykreslí obrázek a tlačítko v jednom z rohů obrázku. Komponenta má celou řadu *props*, kterými lze její vzhled a funkčnost upravit. Následuje jejich výčet.

- *image*: reference k obrázku,

- *text*: text, který se přes obrázek zobrazí,
- *imagePressed*: funkce volaná při kliknutí na obrázek,
- *btnInfoPressed*: funkce volaná při kliknutí na tlačítko,
- *textAlign*: horizontální zarovnání textu,
- *textPadding*: vnitřní odsazení textu vzhledem k obrázku,
- *hasOverlayText*: zda se má text zobrazovat či nikoliv,
- *btnInfoSrc*: reference k obrázku tlačítka,
- *btnInfoSrcPressed*: reference k obrázku tlačítka při stisknutí,
- *btnInfoSrcDisabled*: reference k obrázku tlačítka při stavu *disabled*,
- *btnInfoPosition*: ve kterém rohu obrázku se tlačítko zobrazí,
- *overlayColor*: barva podkladu textu,
- *hasImage*: pokud je nastaveno na *false*, místo obrázku je na pozadí *React Native* komponenta *View*, která může mít nastaven libovolný *style*.

Druhá komponenta je *ClickInfoTask*. Jedná se o rodičovskou komponentu, která vykreslí to, co se jí předloží v JSX *layoutu* programátor mezi otvírací a zavírací *tag*. Nicméně ještě před volání *render* funkce musí být zavolána její metoda *initViews*. Parametry funkce *initViews* jsou pole referencí na *ClickInfoView* komponenty, pole textů, pole obrázků zobrazených přes celou obrazovku a příznak, zda je úkol splněn. Kromě prvního parametru jsou všechny nepovinné.

Komponenta *ClickInfoTask* obsahuje 3 *props*.

- *taskDoneFunction*: funkce, která se zavolá po splnění úkolu,
- *setCoverImage*: funkce zobrazující velký obrázek,
- *isProgressive*: příznak znamenající, zda má uživatel na *ClickInfoView* komponenty klikat v pořadí v jakém jsou uvedeny v poli nebo libovolně.

6.2.4 DragDropQuestion

Typ úkolu vyžaduje tři komponenty - *DroppableView*, *DraggableView*, *DragDropTask*.

Komponentě *DragDropTask* jsou při inicializaci dodány pole referencí na *DroppableView* a *DraggableView* komponenty, které jsou v jejím *layoutu*. Obsahuje funkci *evaluate*, která kontroluje, zda jsou všechny přetahovatelné objekty (*DraggableView* komponenty) na správném místě (reprezentované *DroppableView* komponentami) a zda zároveň nezůstalo žádné místo neobsazené.

DroppableView je komponenta, která může mít v *render* funkci libovolné komponenty. To znamená, že místo, na které se objekty přetahují může být obrázek, text, video nebo cokoliv jiného. Dále má metodu *getPosition* vracející rozměry a umístění komponenty a následující *props* vlastnosti:

- *autocenter*: příznak, zda se mají komponenty přetažené do oblasti komponenty *DroppableView* automaticky vycentrovat nebo ne.

- *isDeceptive*: pokud je vlastnost nastavena na *true*, nebude se při vyhodnocování úkolu kontrolovat, zda je obsazena přetahovatelným objektem nebo nikoliv.

Komponenta *DraggableView* reprezentuje přetahovatelný objekt. Uživatel může tuto komponentu přetahovat po obrazovce díky tomu, že obsahuje *React Native* třídu *PanResponder* [25] a rodičovská komponenta *render* funkce je *Animated.View*, která naslouchá změnám v *PanResponderu* a mapuje je na pozice komponenty na obrazovce. Stejně jako komponenta *DroppableView*, tak i tato má metodu *getPosition*, nicméně vyhodnocení, zda je komponenta správně umístěna, probíhá v této komponentě při události *onPanResponderRelease*. Stav komponenty je přístupný přes metodu *getState*.

Před vykreslením musí programátor zavolat metodu *initOKDroppables* a *initWrongDroppables* a v parametru poskytnou reference na *DroppableView* komponenty.

V metodě *onPanResponderRelease* kromě vyhodnocení dochází k centrování komponenty vzhledem k *DroppableView* komponentě, na kterou je umístěna. Způsob centrování určuje *props* vlastnost *centerTo*, jež může být nastavena na jednu z hodnot *left*, *right*, *center*, *top* nebo *bottom*.

6.2.5 LinkPairQuestion

V tomto úkolu jsou dvě kategorie objektů, které lze spojit čarou. Kreslit čáru je možné jen z jedné kategorie do druhé. Úkol je úspěšně splněn v případě, že objekty z první kategorie jsou spojeny s odpovídajícími objekty ze druhé kategorie. Každému objektu ze druhé kategorie musí být přiřazen alespoň jeden z kategorie první.

Pokud čára nekončí na nějakém prvku kategorie druhé, tak hned zmizí. Úkoly této kategorie se vytváří jako potomci třídy *LinkPairTask*. Ve volání konstruktoru předka (tj. třídy *LinkPairTask*) musí být kromě *props* ještě pole referencí na objekty *Line* (tato třída úkolů využívá *open source* knihovnu *react-native link react-native-svg* [27]). Objekt *Line* vyžaduje souřadnice *x, y* počátečního a koncového bodu a jeho rodič musí být komponenta *Svg* ze stejné knihovny.

Podobně jako u *DraggableView*, tak i zde je využita *React Native* třída *PanResponder*, která při změně souřadnic mění *state* třídy *LinkPairTask*. Ve *state* je pole *x, y* koncových bodů objektů *Line*. Počáteční body jsou drženy v instanční proměnné (také pole *x, y* pozic). Důvod tohoto rozdělení je prostý, kdyby koncové body byly ukládány do instanční proměnné místo do stavu, nebylo by změnou *state* vynuceno volání *render* metody a objekty *Line* by se nemohly měnit dle pohybu prstu po obrazovce.

Programátor musí třídě, jež je potomkem *LinkPairTask,* v konstruktoru nastavit následující instanční proměnné.

- *leftRefs*: pole referencí na objekty z první kategorie,
- *rightRefs*: pole referencí na objekty ze druhé kategorie,
- *lineRefs*: pole referencí na objeky *Line* (musí být jako parametr metody *super*),

- *order*: pole přiřazení (viz dále),
- *orderCorrect*: pole požadovaných přiřazení (viz dále),
- *taskStore*: MobX *store*, do kterého se ukládají souřadnice objektů *Line*.

Pole přiřazení má stejnou délku jako pole objektů první kategorie. Na jednotlivých pozicích jsou indexy znamenající, na jaký prvek z druhé kategorie má být objekt přiřazen. Pole *order* reprezentuje současné přiřazení, pole *orderCorrect* reprezentuje požadované přiřazení. Úkol je splněn, pokud mají obě pole shodné indexy.

6.2.6 CheckBoxQuestion

Tento typ úkolu používá komponentu *CheckBox* [26] upravenou autorem textu tak, aby mohla mít libovolný vzhled. Má mnoho *props*, kterými může být upravena.

- *label*: text vedle *check boxu*,
- *labelBefore*: zda-li je popisek zleva nebo zprava od *check boxu*,
- *labelStyle*: styl popisku,
- *labelLines*: počet řádek popisku,
- *checkboxStyle*: styl *check boxu*,
- *containerStyle*: styl komponenty jako celku,
- *checked*: zda-li je komponenta zaškrtnutá,
- *checkedImage*: obrázek zaškrtnutého *check boxu*,
- *uncheckedImage*: obrázek nezaškrtnutého *check boxu*,
- *underlayColor*: podbarvení při zaškrtnutí komponenty,
- *onChange*: funkce volaná při změně zaškrtnutí,
- *disabled*: zda-li má uživatel možnost změnit zaškrtnutí komponenty.

Komponenta *CheckBoxTask* vygeneruje v *render* funkci vertikální seznam *CheckBox* komponent. Obsahuje funkci *evaluate* pro vyhodnocení úkolu. Má následující *props* vlastnosti.

- *values*: pole *boolean* hodnot, značících, které *CheckBox* komponenty mají být při vykreslení zaškrtnuté, a které nikoliv,
- *labels*: pole popisků pro *CheckBox* komponenty,
- *btnCheckbox*: obrázek nezaškrtnutého *check boxu*,
- *btnCheckboxChecked*: obrázek zaškrtnutého *check boxu*,
- *taskDone*: příznak říkající, zda-li je úkol splněn.

6.2.7 Information

Tento úkol nemá v prostředí *React Native* žádný smysl implementovat, neboť takový typ úkolu vznikne skládáním libovolných komponent do sebe.

6.2.8 Automaticky generované testy

V kapitole 2 se úkoly automaticky generovaly z XML souborů.V případě *JavaScriptu* je však jednodušší použít JSON soubor. Každý JSON soubor reprezentující automaticky generovaný test musí mít elementy *type* reprezentující typ úkolu a elementy *task* specifikující potřebná data.

V případě obrázkových hádanek je v elementu *task* pole s elementy *bck, questions, okIndex*. Element *bck* obsahuje referenci na obrázek, *questions* obsahuje pole referencí na jednotlivé otázky a *okIndex* obsahuje index správné odpovědi v poli *questions*.

Pokud se jedná o doplňování textu, tak je v elementu *task* pole. Každý prvek pole má alespoň jeden záznam s elementy *question* a *anwers*. Element *question* je reference na text otázky a element *answers* obsahuje referenci na pole textů - správné odpovědi na danou otázku.

Komponenty *ImageRiddlesTask* a *TestTask* obě obsahují metody *renderView* a *evaluate*. Metoda *renderView* generuje instance tříd *ImageRiddlesView*, resp. *TaskView* a metodu *evaluate* tato vygenerovaná třída volá kvůli vyhodnocení.

Nicméně tyto komponenty mohou generovat jakoukoliv jinou instanci, neboť ta je jim dodána zvnějšku. Třídy *ImageRiddlesView* a *TaskView* nepatří do knihovny úkolů.

6.2.9 Závěr

Veškeré komponenty knihovny jsou ve složce *app/components* a jsou exportovány v souboru *index.js*. Aplikace, která tuto knihovnu bude používat tak importuje nějakou komponentu například tímto způsobem:

```
import RNEduLibrary, { ResizableText } from 'react-native-edu-library';
```

Knihovnu by bylo možno nainstalovat přes balíčkovací manažer NPM (tam však není dostupná) nebo manuálně.

Pro Android jsou kroky k nasazení následující.

- Otevřít soubor *android/app/src/main/java/[...]/MainApplication.java*.
- Přidat řádek `import com.reactlibrary.RNEduLibraryPackage;` do seznamu importovaných souborů.
- Vložit kód `new RNEduLibraryPackage()` do seznamu návratových hodnot metody *getPackages()*.
- Vložit následující řádky do souboru *android/settings.gradle*:

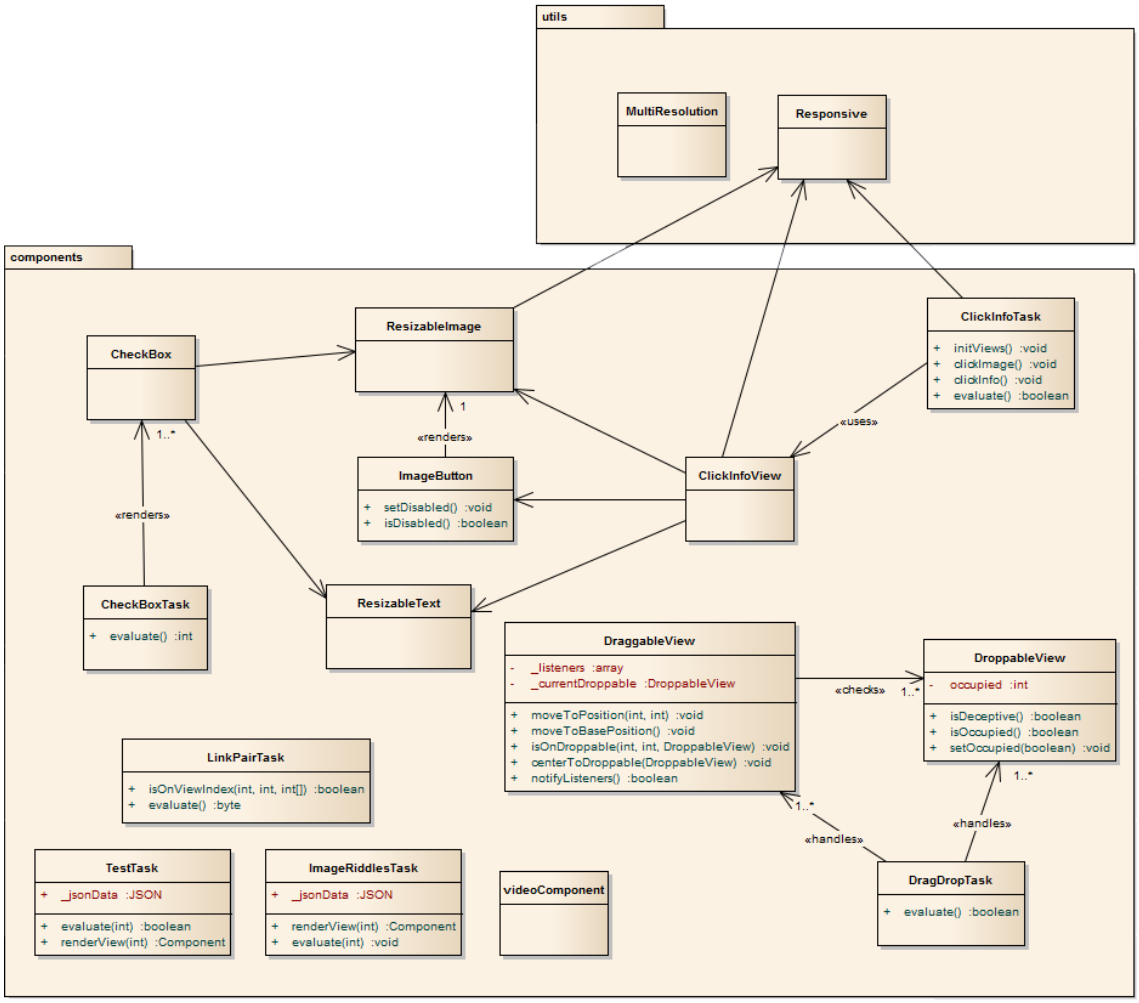
```
include ':react-native-edu-library'
project(':react-native-edu-library').projectDir = new File(rootProject.projectDir, '../cesta-ke-knihovně/android').
```

- Do souboru *android/app/build.gradle* vložit řádek `compile project(':react-native-edu-library')`.

Pro iOS je nutno vykonat následující kroky.

- V navigačním panelu *XCode* kliknout pravým tlačítkem myši na složku *Libraries* a vybrat možnost *Add Files to [jméno projektu]*.
- Vybrat soubor *RNEduLibrary.xcodeproj* v adresáři *ios* knihovny úkolů.
- Vybrat projekt v navigačním panelu *XCode*. Přidat *libRNEduLibrary.a* do *Build Phases / Link Binary With Libraries*.

Pro shrnutí kapitoly je ještě přiložen diagram tříd knihovny úkolů pro vývoj edukativních aplikací.



Obr. 6.1: Diagram tříd knihovny úkolů pro vývoj edukativních aplikací.

6.3 Aplikace *Globální problémy z nadhledu*

V této kapitole jsou popsány implementační detaily aplikace *Globální problémy z nadhledu*. Kapitola se v první části věnuje propojení aplikace s knihovnou úkolů pro tvorbu edukativních aplikací, poté detailně popisuje strukturu aplikace a její funkcionalitu včetně popisu jednotlivých komponent, které musely být v rámci aplikace implementovány, neboť požadovaná funkcionalita nebyla součástí knihovny úkolů.

Aplikace *Globální problémy z nadhledu* využívá knihovnu úkolů pro vývoj edukativních aplikací. V kapitole 6.2 bylo pospáno, jakým způsobem je knihovna do projektu přidána. V případě, že komponenta aplikace *Globální problémy z nadhledu* vyžaduje komponentu z knihovny, importuje si ji následujícím příkazem.

```
import RNEduLibrary, {JménoKomponenty} from 'react-native-edu-library';
```

V první části kapitoly bude popsána struktura projektu a význam jednotlivých složek.

6.3.1 Struktura projektu

Strukturu projektu popisuje následující seznam.

- kořenový adresář
  - android
  - app
  - img
  - ios
  - libraries
  - node_modules

V kořenovém adresáři jsou kromě výše vyjmenovaných složek hlavně soubory *index.android.js* a *index.ios.js*. Oba dva soubory odkazují do jednoho, společného, souboru *index.js* ve složce *app*, neboť aplikace pro iOS i Android má být totožná.

Ve složkách *android* a *iOS* je kód specifický pro danou platformu a zároveň jsou to složky, do kterých *React Native* generuje kód aplikace pro jednotlivé operační systémy.

Složka *img* slouží pro uchování obrázků. Teoreticky by obrázky mohly být i v jiné složce, ale pouze v této složce *React Native* provádí jejich škálování na základě rozlišení zařízení, na kterém aplikace běží.

Ve složce *libraries* je knihovna pro tvorbu edukativních aplikací. Do této složky patří všechny moduly, které nejsou nainstalovány přes balíčkovací manažer NPM.

Ve složce *node_modules* jsou všechny moduly nainstalované pomocí balíčkovacího manažeru NPM. Knihovna pro tvorbu edukativních aplikací by v této složce byla v případě, kdyby byla instalovaná pomocí NPM. Moduly, které nejsou nainstalované pomocí NPM se do složky *node_modules* nedávají, neboť při čištění *cache* paměti se obsah složky automaticky maže.

Za hlavní složku aplikace lze považovat složku *app*. V té je, kromě podsložek, také soubor *index.js*. Funkce tohoto souboru bude popsána níže.

Složka *app* má následující podsložky.

- components
- config
- navigation
- screens
- stores
- styles

Do složky *components* patří veškeré vlastní komponenty. Její obsah bude detailně rozebrán v další části kapitoly.

Ve složce *config* jsou věci, které by v klasické Android aplikaci byly ve složce *values*. Jedná se o texty, barvy a rozměry. Dále je zde soubor *images.js* obsahující

reference na obrázky (význam tohoto souboru je popsán v kapitole 6.3.2) a složka *data*, ve které je pro každou kapitolu aplikace soubor obsahující pole obrazovek dané kapitoly (prvkem pole je komponenta reprezentující daný úkol).

Ve složce *navigation* je komponenta, jejíž *render* funkce vrací *React Native* komponentu *Navigator* [28]. Jedná se o zcela zásadní soubor, neboť Navigator zodpovídá za možnost přecházet z jedné obrazovky na druhou. K mapování obrazovek využívá pole obrazovek ze složky *data*. Výše zmíněný soubor *index.js* dělá hlavně to, že komponentě *Navigator* nastaví vlastnost *intialRoute* a tím spustí první obrazovku.

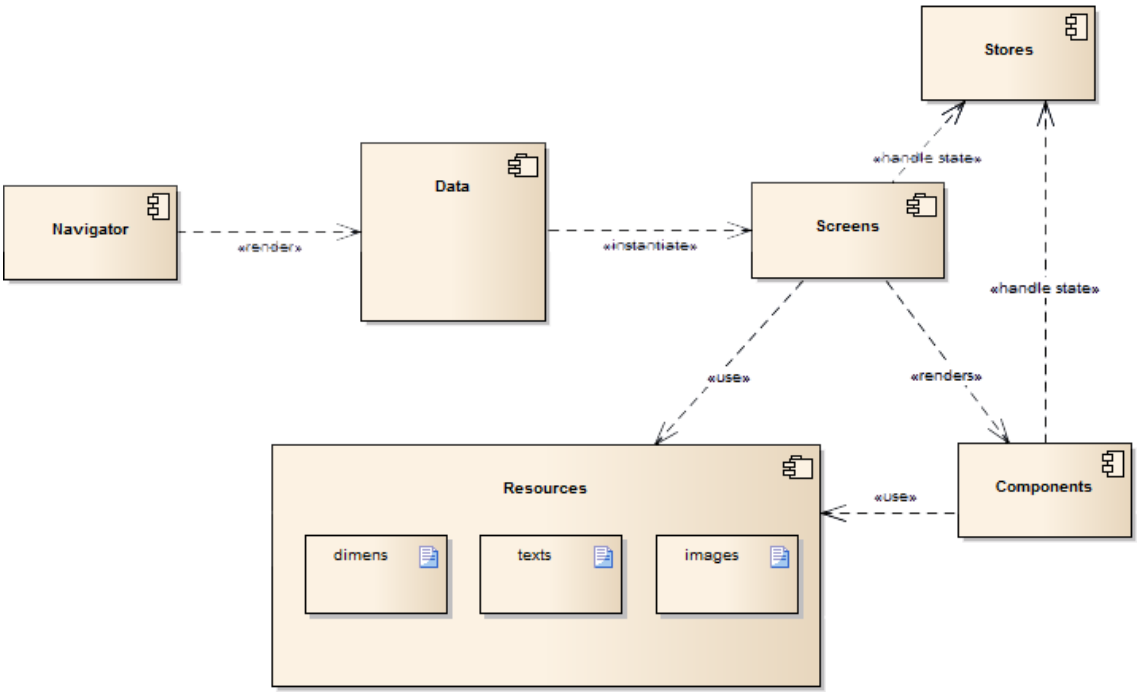
Navigator má *props* jménem *taskInfo*. Do této vlastnosti je ukládán index kapitoly, index obrazovky v rámci kapitoly, nadpis a popis úkolu a *taskStates MobX store*, ve kterém jsou pole stavu splněnosti jednotlivých úkolů. Vlastnosti *taskInfo* má tedy každá obrazovka aplikace.

Složka *screens* obsahuje komponenty reprezentující dané obrazovky (jsou to právě komponenty z této složky, jež jsou obsahem polí ve složce *data*).

Složka *stores* obsahuje *Store* objekty pro držení stavu úkolů pomocí *MobX*.

Složka *styles* obsahuje soubory se styly společnými pro různé komponenty aplikace.

Předchozí popis složek *navigation*, *screens*, *data* a *stores* by mohl být pro čtenáře matoucí. Proto následuje schematický obrázek shrnující vztah mezi jednotlivými komponentami aplikace.



Obr. 6.2: Diagram komponent aplikace *Globální problémy z nadhledu*.

6.3.2 Lokalizace

Pro lokalizaci využívá aplikace *open source* knihovnu *react-native-localization* [31]. Příklad použití ze stránek projektu:

```
import LocalizedStrings from 'react-native-localization';

let strings = new LocalizedStrings({
  en:{
    how:"How do you want your egg today?",
    boiledEgg:"Boiled egg"
  },
  it: {
    how:"Come vuoi il tuo uovo oggi?",
    boiledEgg:"Uovo sodo"
  }
});
```

Tímto způsobem jsou definovány texty a obrázky ve složkách *images* a *texts* pro jazyky čeština a angličtina.

6.3.3 Paměť stavu aplikace

Jak již bylo popsáno v kapitole 6.1.1, aplikace používá pro správu stavu *framework MobX*. Ve složce *stores* tedy definuje *store* třídy, ve kterých jsou stavové proměnné. Při prvním přístupu ke *store* třídě jsou hodnoty proměnných inicializovány z paměti zařízení. *React Native* pro tento účel disponuje třídou *AsyncStorage*, která do paměti zařízení ukládá data v podobě klíč-hodnota. V iOS je reprezentována tzv. serializovaným slovníkem a v OS Android buď *RocksDB* nebo *SQLite* dle toho, co je v

zařízení dostupné [32].

Při změně stavu jsou hodnoty pomocí *AsyncStorage* třídy zapsány do aplikace.

TaskStateStore je třída, která uchovává pole splněnosti úkolů. Je to tedy pole, které má tolik prvků, kolik je kapitol aplikace. Každý prvek pole je opět pole, tentokrát *boolean* hodnot, značících stav úkolu v dané kapitole. Dále třída obsahuje proměnnou *debugMode* typu *boolean*, pokud je její hodnota *true*, tak uživatel může úkoly volně procházet. Je zde i proměnná *scrollEnabled* znamenající, zda je komponenta *ScrollView* aktivní či nikoliv (více v následující kapitole - komponenta *TouchableScrollView*). Proměnné *debugMode* a *scrollEnabled* se do paměti zařízení neukládají.

Třída **DragDropStore** slouží pro držení stavu *DraggableView* komponent. Obsahuje pole souřadnic *x, y* jednotlivých komponent.

Další *store* třídy jsou **HoldIndexStore**, **HoldStringStore**, **LinkPairStore** a **TaskStore**. Poslední zmíněná třída nevyužívá *AsyncStorage* a komponenty, které ji využívají nemohou svůj stav ukládat do paměti.

6.3.4 Komponenty aplikace

Tato kapitola obsahuje popis komponent ze složky *app/components*, jež jsou nezbytné pro fungování aplikace.

BaseLayout je komponenta, která v *render* funkci vykresluje *layout* aplikace tak, jak je schematicky znázorněno na Obr. 4.1. Zároveň poskytuje metodu *setCoverImage*, která zobrazí obrázek přes celou obrazovku. Tuto zodpovědnost nemůže mít žádná jiná komponenta, neboť *View*, do kterého se obrázek vykreslí musí být ve struktuře komponent na nejvyšší úrovni.

Další metodou je *evaluate*. Tato metoda uloží stav úkolu do MobX *Stores* objektu podle toho, z jaké kapitoly a jakého úkolu je volána. Také podle výsledku vyhodnocení úlohy nastaví stav tlačítek v pravém menu.

Komponenta obsahuje následující vlastnosti *props*.

- *title*: obsahuje titulek úkolu zobrazený v horním rámečku (viz Obr. 1.1),
- *description*: obsahuje popis úkolu zobrazený v horním rámečku,
- *evaluate*: vyhodnocující funkce, funkce *evaluate* komponenty *BaseLayout* volá tuto funkci kvůli zjištění stavu splněnosti,
- *navigateLeftPressed*: funkce volaná při stisknutí tlačítka vlevo (volá se předtím, než dojde k přechodu na předchozí obrazovku),
- *navigateRightPressed*: funkce volaná při stisknutí tlačítka vpravo (volá se předtím, než dojde k přechodu na následující obrazovku),
- *btnHomePressed*: funkce volaná při stisknutí tlačítka *Home* (volá se těsně předtím, než dojde k přechodu na další obrazovku),
- *evaluateDisabled*: hodnota typu *boolean* značící, zda uživatel může kliknout na tlačítko Vyhodnotit,
- *hideTopPanel*: hodnota typu *boolean* značící, zda je horní panel s nadpisem

a popisem úkolu vykreslen nebo ne.

ClickSatelliteTask reprezentuje tři úkoly v první kapitole, ve kterých má uživatel kliknout postupně na satelit obrázek a tlačítko Info. Obrázek s tlačítkem Info je implementován komponentou *ClickInfoView* z knihovny úkolů. Obrazovka, která tento typ úkolu používá je potomkem komponenty. Musí mít nastavené následující instanční proměnné.

- *_title*: nadpis úkolu,
- *_description*: popis úkolu,
- *_text*: text, který se zobrazí přes obrázek,
- *_imageLeft*: obrázek vlevo,
- *_imageRight*: obrázek vpravo.

Obr. 6.3: Snímek obrazovky, která je instancí komponenty *ClickSatelliteTask*.

Obr. 6.3 ukazuje příklad komponenty, která je potomkem *ClickSatelliteTask*. Úkol je v počátečním stavu. Po kliknutí na satelit se vpravo zobrazí obrázek s tlačítkem Info. Po kliknutí na tlačítko Info se přes pravý obrázek zobrazí text a úkol se nastaví jako splněný.

CreateEnvironmentTask je komponenta pro úkoly, ve kterých je úkolem uživatele kliknout na všechna tlačítka reprezentující dané prostředí a poté na všechna tlačítka reprezentující živočichy a zvířata obývající dané prostředí. Obrazovka s tímto úkolem je potomkem komponenty *CreateEnvironmentTask*. Komponenta má následující instanční proměnné, které musí její potomek inicializovat.

- *_animalBtns*: reference tlačítek se zvířaty nebo rostlinami,
- *_partBtns*: reference tlačítek s typy prostředí,
- *_wrongIndices*: indexy tlačítek se zvířaty / rostlinami, na které uživatel nemá kliknout.

Obr. 6.4: Snímek obrazovky, která je instancí komponenty *CreateEnvironmentTask*.

Na Obr. 6.4 jsou v horním pruhu patrná tlačítka reprezentující typ prostředí (v tomto případě moře) a v pravém panelu jsou tlačítka rostlin a živočichů.

DraggableView je potomkem třídy *DraggableView*. Rozšiřuje jeho funkčnost o *MobX Store*.

ImageRiddlesView je *View* komponenta pro *ImageRiddlesTask* úkol z knihovny úkolů. Obsahuje *render* funkci, která vykreslí obrázek na pozadí a rámeček s možnostmi. Možnosti také promíchá. Jako ilustrační obrázek může posloužit Obr. 1.9.

QuoteTask je jednoduchý úkol pro zobrazování citátů (k vidění na Obr. 1.8). Má tři *props*:

- *quote*: text citace,
- *author*: autor citace,
- *desc*: popisek v dolní části obrazovky.

RadioButton je jedno ze základních *View* v OS Android, nicméně iOS toto tlačítko nepoužívá. *React Native* ho ve své sadě *View* také nemá, takže muselo být pro tuto aplikaci vytvořeno. Pomocí *props* lze nastavit rozměry i barvu tlačítka.

ResizableText je potomek třídy *ResizableText* z knihovny úkolů. Rozšiřuje komponentu o styly specifické pro aplikaci - barvu, velikost a typ písma. Typ písma závisí na operačním systému. V OS Android byl zvolen typ písma *sans-serif-condensed* a v iOS *HelveticaNeue-Light*.

Slider je třída, která není dílem autora této práce. Její kód je dostupný přes GitHub [29]. Používá se v úkolech, které vyžadují komponentu *Slider*. Autor této práce třídu upravil tak, aby bylo možno *Slider* vykreslit vertikálně, neboť to původní komponenta neumí. Náhled použití komponenty je například v obrázku Obr. 1.1.

Komponenta **TestResults** je použitá v poslední obrazovce kapitoly Test. Obsahuje metodu *renderCards*, která vykreslí síť dlaždic *n * m* podle počtu úkolů a otázek v nich. V *props* vlastnostech je pole *MobX store* objektů pro jednotlivé testy. Z těchto objektů komponenta pozná počet dlažic a také to, jaké mají být odkryty při stisknu tlačítka Vyhodnotit.

TestView poskytuje *layout* komponentně *TestTask*, která je obsažena v knihovně úkolů. Obsahuje metodu na promíchání otázek a *render* funkci, která vykreslí seznam otázek a *TextInput* komponent, do kterých uživatel zadává odpovědi. Její *props* vlastnosti jsou shrnuty v následujícím seznamu:

- *underlineColorAndroid*: barva podtržení textu v komponentě *TextInput*. Podtržení textu v této komponentě je záležitostí pouze OS Android, v iOS nemá žádný efekt,
- *textColor*: barva textu komponenty *TextInput*,
- *bordercolor*: barva dolního rámečku pro komponentu *TextInput*,
- *selectionColor*: barva zvýraznění textu v komponentě *TextInput*,
- *icoAnswerOk1*: reference na obrázek, která se zobrazí při správné odpovědi,
- *icoAnswerWrong*: reference na obrázek, který se zobrazí při špatné odpovědi,
- *textOk*: text zobrazený při správné odpovědi,
- *textTryAgain*: text zobrazený při první špatné odpovědi,
- *textWrong*: text zobrazený při druhé špatné odpovědi,
- *taskInfo.data*: obsahuje pole otázek a správných odpovědí.

Toast je třída, která vykresluje komponentu *react-native-root-toast* [30]. Tato komponenta zobrazuje přes obrazovku nějaký text. Po určené době text zmizí. Třída *Toast* této komponentě nastaví *props* a vykreslí ji. *Toast* je použit ve většině obrazovek pro zobrazení stavu splněnosti po kliknutí na tlačítko Vyhodnotit. Vzhledem k tomu, že má mít tento text na všech obrazovkách shodný vzhled, bylo lepší vytvořit novou komponentu, než nastavovat komponentě *react-native-root-toast* vlastnosti zvlášť v každém okně. Vlastnosti *props* komponenty:

- *textColor*: barva textu (nastavena ze souboru *colors* na hodnotu *text*),
- *backgroundColor*: barva pozadí textu (ze souboru *colors* na *purple*),

- *shadow*: zda-li je kolem rámečku s textem stín (na staveno na *true*),
- *shadowColor*: barva stínu rámečku (shodná s *backgroundColor*).

React Native má vlastní komponentu *ScrollView*. Když je však uvnitř této komponenty nějaká komponenta využívající *PanHandler* (například *DraggableView*), tak události ze *ScrollView* nejsou propagovány, neboť *TouchEvent* je zpracován komponentou *ScrollView* a není propagován dále (konkrétně není propagována pouze událost *onPanResponderMove* v tom směru, ve kterém by mohlo probíhat *scrollování*. Událost *onPanResponderGrant* funguje). Toto se děje pouze v iOS, v Android je funkčnost správná. **TouchableScrollView** je komponenta, která tuto chybu řeší. Předpoklad je, že *MobX store* na komponentě s třídou *PanHandler* obsahuje metody *setScrollEnabled* a *isScrollEnabled* a při události *onPanResponderGrant* zavolá metodu *setScrollEnabled* s parametrem *false*. *TouchableScrollView* má v *render* funkci obyčejné *ScrollView*, kterému nastavuje vlastnost *scrollEnabled* voláním funkce *isScrollEnabled*. Tím dočasně *scrollování* vypne a uživatel tak může s vnitřní komponentou interagovat.

TransformableImage je komponenta, která vykreslí dva obrázky přes sebe. Horní obrázek funguje jako roleta, kterou uživatel může pohybem prstu rolovat do strany a skrývat nebo odkrývat tak obrázek pod ní. Komponentě lze nastavit tyto *props*:

- *srcBackground*: reference na podkladový obrázek,
- *srcForeground*: reference na obrázek - roletu,
- *defaultOffset*: počáteční odkrytí rolety v procentech, pokud není nastaveno použije se 0.

První dvě *props* vlastnosti musí být nastaveny.

YesNoTask je komponenta, která vykresluje seznam tvrzení a *RadioButton* komponent pro odpovědi ano / ne na tato tvrzení. Pokud uživatel odpoví správně, zobrazí se vedle každého tvrzení ještě text s doplňujícím komentářem. Komponenta má tři *props*:

- *correctValues*: pole správných odpovědí (každý prvek má hodnotu 0 nebo 1),
- *taskDone*: zda-li je úkol splněn. Pokud ano, automaticky se vyberou správné odpovědi, zobrazí se doplňující text a *RadiButton* se zneaktivní,
- *taskStore*: *MobX store*, ve kterém jsou uloženy odpovědi uživatele.

Komponenta **YesNoTaskSpecial** je potomkem třídy *YesNoTask*. Přepisuje její *render* funkci tím způsobem, že místo *RadioButton* komponent používá pro odpovědi *ImageButton* komponenty. Má stejné *props* jako předchozí třída.

7 TESTOVÁNÍ

Kapitola se věnuje testování aplikace *Globální problémy z nadhledu* a komponentám z knihovny úkolů pro tvorbu edukativních aplikací. Uživatelské testování aplikace neprobíhalo, neboť tímto způsobem již byla testována původní aplikace pro OS Android. Cílem testování je tedy ověřit shodu vzhledu a funkcionality původní aplikace a obou verzí aplikace nové (iOS a Android verze).

V prostředí Android probíhalo testování na dvou zařízeních, Samsung Galaxy Tab S2 o úhlopříčce obrazovky 10" a verzi OS 6.0 a Asus MeMO Pad 7 o úhlopříčce 7" a verzi OS 5.0. Pro testování s operačním systémem iOS bylo použito zařízení iPad 2 s verzí operačního systému iOS 9.

7.1 Průběh testování

Průběh testování byl pro oba operační systémy shodný. Nejprve se vytvořila *Release* verze aplikace, tzn. spustitelná aplikace, která ke svému běhu nepotřebuje *React Packager* sever, a která je teoreticky připravena k nahrání na obchod s aplikacemi.

Poté byla aplikace spuštěna a každá obrazovka byla porovnána s originální aplikací. Porovnáván byl jak vzhled, tak požadovaná funkčnost. Také se testovalo, zda si aplikace drží svůj stav tam, kde si ho drží aplikace původní.

Testování prováděl autor této diplomové práce, neboť má detailní znalosti původní aplikace.

7.2 Testování Android

Aplikace pro testování se z projektu vytvoří jednoduchými konzolovými příkazy z domovského adresáře projektu: `cd android && gradlew assembleRelease` a `react-native run-android --variant=release`. Android projekt musí být předtím podepsán, což je možné z Android Studia.

Tabulka 7.1 zobrazuje nálezy z testování a přiděluje jim identifikátor. V tabulce je ve sloupci Testovací zařízení buď písmeno S (Samsung) nebo písmeno A (Asus).

Tabulka 7.2 dle identifikátoru z tabulky 7.1 popisuje příčinu chyby a nápravu.

Tab. 7.1: Seznam nálezů z testování aplikace *Globální problémy z nadhledu* (Android).

ID	Testovací zařízení	Kapitola, úkol, komponenta	Nález
1	S, A	BaseLayout	Tečky na rámečku v pravém menu nejsou viditelné celé.
2	S, A	VideoComponent	Video se nedaří vždy přehrát.
3	A	YesNoTask	Popisek po splnění úkolu přesahuje do pravého menu.
4	A	Kap. 1, úkol 7	Text v pravém menu přesahuje přes obrazovku.
5	S, A	Kap. 2, úkol 5	Nelze otevřít aplikaci kalkulačka.
6	S, A	Kap. 5, úkol 5	Obrázky po splnění úkolu nejsou zarovnané v jedné linii.
7	S, A	globální nález	Po určité době se obrázky přestanou vykreslovat a aplikace se zasekává, animace jsou trhané, po dalším používání je systémem vypnuta pro nedostatek paměti.

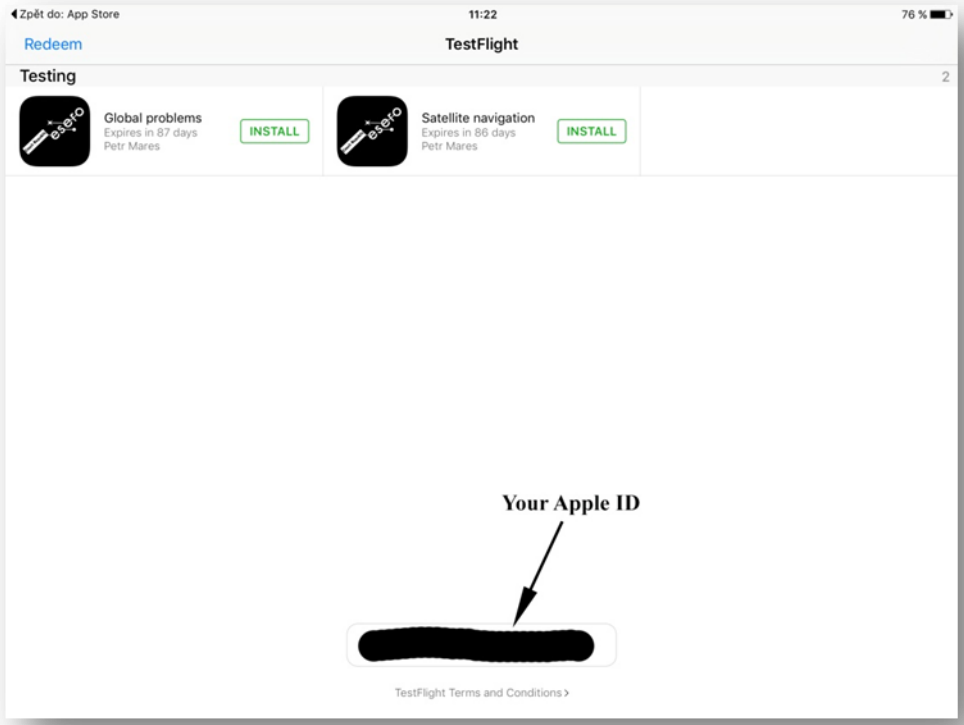
Tab. 7.2: Tabulka příčin nálezů z tabulky 7.1 a jejich řešení.

ID	Příčina	Řešení
1	Android nepodporuje CSS vlastnost *overflow: hidden*.	Rámeček je implementován obrázkem s průhledným pruhem vlevo do kterého se vykreslí zbytek tečky.
2	Neznámá chyba v komponentě *react-native-video*.	Vyměnit komponentu.
3	Bug ve *frameworku* - text uvnitř komponenty se stylem *flexDirection: row* nerespektuje jeho šířku.	Nastavit šířku boxu na pevnou hodnotu.
4	Málo prostoru pro text.	Přidána komponenta *TouchableScrollView*.
5	Kalkulačka na rozdíl od fotoaparátu, webového prohlížeče a další aplikací není specifikována pomocí URL, ale na různých zařízení je implementována jinak.	Nelze opravit. Bude fungovat jen na některých zařízení.
6	Obrázky zobrazené po splnění nemají stejné rozměry.	Opravit obrázky.
7	Chyba *frameworku* specifická pro Android. Aplikace s velkým množstvím obrázků touto chybou trpí [33].	V *React Native* je na toto téma několik otevřených *issue* [34, 35]. Podrobnosti k této chybě jsou v kapitole 8 Závěr.

7.3 Testování iOS

Vygenerovat aplikaci pro iOS a spustit ji v připojeném zařízení lze příkazem `react-native run-ios --configuration Release --device="název"`. Nicméně v tomto případě bylo nutno vygenerovat aplikaci, kterou si může pustit téměř kdokoliv, neboť to vyžadoval vedoucí práce. Lidé, kteří aplikaci chtějí spustit musí mít tablet s iOS a poskytnou své *appleID*. Aplikace musí být vygenerována přes placený účet a nahrána do *iTunnes Connect*. Uživatel, který chce takovou aplikaci testovat musí vykonat tyto kroky:

1. Přihlásit se do svého iOS zařízení pod účtem, který poskytli pro testování.
2. Stáhnout si z *App Store* aplikace *TestFlight*.
3. Aplikace *TestFlight* má na své úvodní obrazovce aplikace, které jsou uživateli přístupné. Mezi těmito aplikacemi je i aplikace *Globální problémy z nadhledu*.
4. Kliknutím na aplikaci začne stahování aplikace.
5. Po stažení a nainstalování je na ploše vytvořen zástupce. Aplikace *Globální problémy z nadhledu* poté může být spuštěna z plochy zařízení nebo z aplikace *TestFlight*.



Obr. 7.1: Snímek aplikace *TestFlight* s možností stažení aplikace *Globální problémy z nadhledu*.

Tabulka 7.3 zobrazuje nálezy z testování a přiděluje jim identifikátor a tabulka 7.4 dle identifikátoru z tabulky 7.3 popisuje příčinu chyby a nápravu. Nálezy z tabulky 7.1 s ID 3, 5 a 6 se v testování na iOS vyskytly také, nicméně do tabulek 7.3 a 7.4 zahrnuty nebyly.

Tab. 7.3: Seznam nálezů z testování aplikace *Globální problémy z nadhledu* (iOS).

ID	Kapitola, úkol, komponenta	Nález
1	TouchableScrollView	Pokud je v komponentě *TouchableScrollView* komponenta využívající *PanResponder*, tak její události nefungují (např. *DraggableView*).
2	DragDropTask	Některé *DraggableView* komponenty se při přetahování skryjí za jinou komponentu.
3	LinkPairTask	Čára, kterou uživatel prstem kreslí je pod komponentou místo nad ní.
4	BaseLayout, TestView	Rámeček pod nadpisem kapitoly a rámeček pod komponentou *TextInput* není vidět.

Tab. 7.4: Tabulka příčin nálezů z tabulky 7.3 a jejich řešení.

ID	Příčina	Řešení
1	v iOS není propagován *TouchEvent*.	Tento problém je podrobně popsán v kapitole 6.3.4, komponenta *TouchableScrollView*.
2	iOS nepodporuje CSS vlastnost *zIndex*.	Náprava spočívá v tom, aby *DraggableView* komponenty byly v *layoutu* vždy uvedeny nakonec. Toho lze jednoduše dosáhnout tím, že se v *layoutu* použije místo *flexDirection: row/column* vlastnost *row-reverse* nebo *column-reverse*. Pokud by toto nebylo z nějakého důvodu možné, stačí nastavit *position: absolute*.
3	iOS nepodporuje CSS vlastnost *zIndex*.	Stejné jako v předchozím řádku.
4	iOS nepodporuje CSS vlastnost *borderBottom*.	Místo CSS *borderBottom* se pod komponentu vloží prázdná *View* komponenta se stylem *border*, ten podporován je.

7.4 Závěr

Tabulky 7.1 a 7.3 obsahují seznamy chyb, které se v aplikaci našly. Nález 2 z tabulky 7.1 se před odevzdáním práce nepodařilo opravit. *Open Source* knihoven pro přehrávání videa je v *React Native* komunitě celá řada, je tedy pravděpodobné, že se mezi nimi vyskytuje nějaká, která nebude dělat problémy.

8 ZÁVĚR

Hlavním cílem předložené diplomové práce bylo vytvořit implementaci existující knihovny úkolů pro tvorbu edukativních aplikací a na ní postavené aplikace *Globální problémy z nadhledu*, jejichž dosavadní implementace lze provozovat pouze v prostředí Android, v multiplatformním prostředím. Tento úkol v sobě rovněž zahrnoval výběr multiplatformního frameworku, který by byl pro tento typ aplikací vhodný. Součástí práce proto byla rovněž poměrně podrobná analýza původní implementace předmětné knihovny i jí využívající aplikace *Globální problémy z nadhledu*.

Následným krokem byla analýza dostupných *frameworků* pro vývoj multiplatformních mobilních aplikací v jazyce *JavaScript*. V jejím rámci vznikly dva prototypy uvedené aplikace - jeden implementovaný ve *frameworku React Native* a druhý ve *frameworku Apache Cordova*.

Na základě vyhodnocení těchto prototypů byl pro následnou implementaci jako nejvhodnější řešení zvolen *framework React Native*. Prostředky tohoto frameworku byla naprogramována výsledná verze knihovny úkolů pro tvorbu edukativních aplikací i jí využívající výukové aplikace *Globální problémy z nadhledu*. Práce popisuje způsob, jak spojit knihovnu s aplikací, a to buď spojit přímo nebo pomocí balíčkovacího manažeru NPM.

Aplikace *Globální problémy z nadhledu* byla otestována oproti původní aplikaci na skutečných zařízeních s operačními systémy Android a iOS. V průběhu těchto testů byly identifikovány a opraveny jednak chyby způsobené autorem diplomové práce, ale zejména ty, které vznikly při implementaci v důsledku rozdílného *chování frameworku React Native* v různých operačních systémech. Pozitivním výsledkem práce je zjištění, že takových rozdílů je velmi málo, a že použitý *framework* umožňuje tvorbu skutečně *multiplatformních* aplikací. Zjištěné rozdíly jsou v předložené práci uvedeny společně s popisem způsobu jak je ošetřit.

Byla však objevena i jedna zásadnější chyba, která se týká *frameworku React Native* v operačního systému Android, a která způsobuje mizení obrázků v případě, že jich je v aplikaci mnoho (hranice mezního počtu obrázků je dána pouze velikostí paměti zařízení, na němž aplikace běží).

V kapitole Testování jsou uvedeny dva odkazy na *issues*, které se danému tématu věnují. Zajímavé je, že 20. 5. 2017, tedy nedlouho před odevzdáním práce, bylo jedno z nich [34] uzavřeno jako vyřešené. Byl uveden důvod, který tuto chybu způsobuje, a popsána jeho oprava (týká se komponenty *ScrollView*). Autor diplomové práce ještě stihl zareagovat a ověřil toto zveřejněné řešení v aplikaci *Globální problémy z nadhledu*. Jediným zatím nevyřešeným problémem je tak pouze stabilita komponenty *VideoView* pod oběma operačními systémy - Android i iOS.

Úspěšně se podařilo převést Java knihovnu a poměrně složitou (co se týče komplexity uživatelského rozhraní) Android aplikaci do multiplatformního prostředí. Kód pro operační systém Android a iOS se tak liší v jediném řádku kódu:

```
fontFamily: (Platform.OS === 'ios') ? 'HelveticaNeue-Light' : 'sans-serif-condensed'.
```

Tento řádek nastavuje typ písma pro každý z uvažovaných operačních systémů Android a iOS.

Závěrem lze konstatovat, že zvolený *framework React Native* umožňuje tvorbu skutečně multiplatformních aplikací. A vzhledem k tomu, že se jedná o nativní multiplatformní *framework*, tedy výsledná aplikace je nativní aplikací příslušného zařízení, není tato multiplatformnost vykoupena degradací výkonu aplikace.

LITERATURA

[1] GitHub - Rajawali/Rajawali: Android OpenGL ES 2.0/3.0 Engine. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 06.05.2017]. Dostupné z: *https://github.com/Rajawali/Rajawali*

[2] Platform definition by The Linux Information Project. *The Linux Information Project (LINFO) Home Page* [online]. Dostupné z: *http://www.linfo.org/platform.html*

[3] Ritesh Patil. Pros and Cons of Cross-Platform Mobile App Development. *InfoQ*. [online]. 13.9.2016 [cit. 2016-12-28]. Dostupné z: *https://www.infoq.com/articles/mobile-cross-platform-app-development*

[4] React.Component - React. [online]. Copyright © 2017 Facebook Inc. [cit. 06.05.2017]. Dostupné z: *https://facebook.github.io/react/docs/react-component.html*

[5] Dive into React Native performance. *Facebook Code* [online]. Copyright © 2016 [cit. 2017-05-06]. Dostupné z*: https://code.facebook.com/posts/895897210527114/dive-into-react-native-performance/*

[6] Bridging in React Native. *Tadeu Zagallo* [online]. Copyright © 2015 [cit. 2017-05-06]. Dostupné z: *https://tadeuzagallo.com/blog/react-native-bridge/*

[7] Mobile App Development & App Creation Software – Xamarin. *Xamarin.* [online]. © 2016 [cit. 2016-12-29]. Dostupné z: *https://www.xamarin.com/*

[8] React Native: Bringing modern web techniques to mobile. *F code*. [online]. 26.3.2015 [cit. 2016-12-29]. Dostupné z: *https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/*

[9] React Native | A framework for building native apps using React. *React Native.* [online]. © 2016 [cit. 2016-12-31]. Dostupné z: *https://facebook.github.io/react-native/*

[10] Architectural overview of Cordova platform – Apache Cordova. *Apache Cordova.* [online]. © 2012, 2013, 2015 [cit. 2016-12-31]. Dostupné z: *https://cordova.apache.org/docs/en/latest/guide/overview/*

[11] Which to Use: Cordova or PhoneGap? | | InformIT. *InformIT: The Trusted Technology Source for IT Pros and Developers* [online]. Copyright © 2017 Pearson Education, [cit. 07.05.2017]. Dostupné z: *http://www.informit.com/articles/article.aspx?p=2478076*

[12] do u webview? | High Performance Web Sites. *Steve Souders - High Performance Web Sites* [online]. Copyright © Steve Souders [cit. 07.05.2017]. Dostupné z: *https://www.stevesouders.com/blog/2014/10/09/do-u-webview/*

[13] Apple Shows Love for HTML5 with iOS 8 | Sencha. *Enterprise Web Apps - Design, Develop, and Test | Sencha.com | Sencha* [online]. Dostupné z: *https://www.sencha.com/blog/apple-shows-love-for-html5-with-ios-8/*

[14] The Updatable WebView on Android 5.0 Lollipop - what is it and why should you care? | Infinum. *Infinum | App design & development* [online]. Copyright © 2017 Infinum Inc. [cit. 07.05.2017]. Dostupné z: *https://infinum.co/the-capsized-eight/the-updateable-webview-on-android-5-lollipop-what-is-it-and-why-should-you-care*

[15] Android Lollipop. *Android Developers* [online]. Copyright © 2016 [cit. 2017-05-07]. Dostupné z: *https://developer.android.com/about/versions/lollipop.html#WebView*

[16] Performance. *React Native* [online]. Copyright © 2017 [cit. 2017-05-07]. Dostupné z:

https://facebook.github.io/react-native/docs/performance.html

[17] Stefano Falda. ReactNativeLocalization. *GitHub.* [online]. © 2017 [cit. 2017-01-02]. Dostupné z: *https://github.com/stefalda/ReactNativeLocalization*

[18] Creating Custom Directives. *AngularJS.* [online]. ©2010-2016 [cit. 2017-01-02]. Dostupné z: *https://docs.angularjs.org/guide/directive*

[19] Ioannis Kokkinidis. How to set font size for different IOS devices in react native. *Stack Overflow.* [online]. 16.12.2016 [cit. 2017-01-10]. Dostupné z: *http://stackoverflow.com/questions/32947036/how-to-set-font-size-for-different-ios-devices-in-react-native*

[20] [Android] Overflow:hidden not work properly. *GitHub: facebook / react-native.* [online]. 2016 [cit. 2017-01-10]. Dostupné z: *https://github.com/facebook/react-native/issues/3198*

[21] Ionic. *Ionic.* [online]. © 2013-2017 [cit. 2017-01-16]. Dostupné z: *https://ionicframework.com/*

[22] ECMAScript® 2016 Language Specification. [online]. Copyright © 2016 Ecma International [cit. 07.05.2017]. Dostupné z: *https://www.ecma-international.org/ecma-262/7.0/*

[23] GitHub - frostney/react-native-create-library: Command line tool to create a React Native library with a single command. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 08.05.2017]. Dostupné z: *https://github.com/frostney/react-native-create-library*

[24] GitHub - react-native-community/react-native-video: A <Video /> component for react-native. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 08.05.2017]. Dostupné z: *https://github.com/react-native-community/react-native-video*

[25] PanResponder. [online]. Copyright © 2017 Facebook Inc. [cit. 08.05.2017]. Dostupné z: *https://facebook.github.io/react-native/docs/panresponder.html*

[26] GitHub - sconxu/react-native-checkbox: Checkbox component for React native. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 08.05.2017]. Dostupné z: *https://github.com/sconxu/react-native-checkbox*

[27] GitHub - react-native-community/react-native-svg: SVG library for React Native.. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 13.05.2017]. Dostupné z: *https://github.com/react-native-community/react-native-svg*

[28] Navigation. [online]. Copyright © 2017 Facebook Inc. [cit. 13.05.2017]. Dostupné z: *https://facebook.github.io/react-native/docs/navigation.html*

[29] GitHub - jeanregisser/react-native-slider: A pure JavaScript <Slider> component for react-native. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 13.05.2017]. Dostupné z: *https://github.com/jeanregisser/react-native-slider*

[30] GitHub - magicismight/react-native-root-toast: react native toast like component, pure javascript solution. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 14.05.2017]. Dostupné z: *https://github.com/magicismight/react-native-root-toast*

[31] GitHub - stefalda/ReactNativeLocalization: Class to localize the ReactNative interface. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 14.05.2017]. Dostupné z: *https://github.com/stefalda/ReactNativeLocalization*

[32] AsyncStorage. [online]. Copyright © 2017 Facebook Inc. [cit. 14.05.2017]. Dostupné

z: *https://facebook.github.io/react-native/docs/asyncstorage.html*

[33] Known Issues – React Native | A framework for building native apps using React. [online]. Copyright © 2016 Facebook Inc. [cit. 14.05.2017]. Dostupné z: *https://facebook.github.io/react-native/releases/0.26/docs/known-issues.html*

[34] Images fail to render after a certain number are added to the page [Android, RN 0.36] · Issue #10569 · facebook/react-native · GitHub. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 14.05.2017]. Dostupné z: *https://github.com/facebook/react-native/issues/10569*

[35] [Android] not all images are shown · Issue #13600 · facebook/react-native · GitHub. *The world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 14.05.2017]. Dostupné z: *https://github.com/facebook/react-native/issues/13600*

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

API	Application Programming Interface, rozhraní pro programování aplikací
CLI	Command-line interface, rozhraní příkazové řádky
CSS	Cascading Style Sheets, kaskádové styly
DOM	Document Object Model, objektový model dokumentu
DTD	Document Type Definition, definice typu dokumentu
HTML	HyperText Markup Language, hypertextový značkovací jazyk
HTTP	Hypertext Transfer Protocol, hypertextový přenosový protokol
ID	Identifier, identifikátor
JDK	Java Development Kit
JSX	JavaScript XML
OS	Operating System, operační systém
SDK	Software Development Kit, soubor nástrojů pro vývoj software
UI	User Interface, uživatelské rozhraní
UML	Unified Modeling Language
URL	Uniform Resource Locator, jednotná adresa zdroje
UX	User Experience, uživatelský prožitek
VM	Virtual Machine, Virtuální stroj
XML	Extensible Markup Language, rozšiřitelný značkovací jazyk