FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Věnná města českých královen - Jádro mobilního klienta
Student:	Bc. Ondřej Slabý
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Věnná města Českých Královen je platforma zaměřující se na zobrazení historického prostředí v mobilních aplikacích pomocí prvků rozšířené reality.

1. Analyzujte a popište možný způsob automatické aktualizace aplikace pod OS Android pomocí vybraného systému CI.
2. Analyzujte uživatelské rozhraní alespoň 5 mobilních aplikací, které využívají technologií rozšířené reality za účelem interakce s virtuálními objekty umístěnými v reálném světě.
3. Pomocí metod softwarového inženýrství navrhněte modulární architekturu pro výslednou aplikaci se zaměřením na funkce jádra a samotné jádro mobilní aplikace.
4. V návrhu počítejte s potřebou měřit využití systémových prostředků (CPU, RAM, disk, datová propustnost, latence) modulů rozpoznání obrazu a vybrat nejvhodnější modul pro konkrétní konfiguraci hardwaru.
5. Implementujte prototyp jádra a propojte jej se souběžně vyvíjenými moduly.
6. Hotový prototyp podrobte vhodným testům.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 23. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Věnná města českých královen – Jádro mobilního klienta

Bc. Ondřej Slabý

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Chludil

7. května 2019

Poděkování

Chtěl bych poděkovat svému vedoucímu diplomové práce Ing. Jiřímu Chludilovi za odborné vedení, za pomoc a rady při zpracování této práce. Dále bych rád poděkoval kolegovi Bc. Jaroslavu Štěpánovi za výbornou spolupráci na návrhu a implementaci platformy.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 7. května 2019 .....

Odkaz na tuto práci

Slabý, Ondřej. *Věnná města českých královen – Jádro mobilního klienta.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce je součástí projektu Věnná města českých královen. Cílem tohoto projektu je mimo jiné tvorba mobilní aplikace pro zařízení s operačním systémem Android. Účelem této aplikace je zobrazování historických verzí budov na jejich skutečných pozicích využitím prvků rozšířené reality. Práce se věnuje analýze uživatelského rozhraní podobných aplikací a návrhu modulární architektury, jejímž cílem je oddělit složité výpočty, které jsou k realizaci rozšířené reality nutné, od specifik systému Android, čímž je usnadněna jejich implementace. Navržená architektura je v práci realizována a otestována, včetně uživatelského rozhraní na základě poznatků z analýzy. Výsledná aplikace je automaticky nahrávána na obchod Google Play Store a tento postup v práci popsán.

Klíčová slova historické budovy, rozšířená realita, modulární architektura, dependency injection, kontinuální integrace, Android

Abstract

This thesis is part of the project Dowry Towns of the Queens of Bohemia. The aim of the project is, among other things, to create a mobile application for the Android operating system. The goal of the application is to display historical versions of buildings at their real locations using augmented reality. The thesis deals with the analysis of the user interface of similar applications and design of a modular architecture which aims to separate algorithms used to make augmented reality possible from the specifics of the Android platform to make their implementation easier. The designed architecture is realized as part of the thesis, including user interface. It is tested and the resulting application uploaded to Google Play Store automatically. The process of the automatic upload is also covered by the thesis.

Keywords historical buildings, augmented reality, modular architecture, dependency injection, continuous integration, Android

Obsah

Seznam obrázků

Úvod

V současné době se stále více šíří zájem o technologie virtuální reality a zároveň také technologie rozšířené reality. Technologie virtuální reality obvykle spočívají v umístění uživatele do virtuálního prostoru použitím speciální soupravy. Taková souprava následně sleduje pohyb uživatele a jeho pohyb promítá do virtuálního prostoru, na základě čehož je uživateli pomocí soupravy zobrazena virtuální scéna. Na rozdíl od virtuální reality, rozšířená realita většinou nevyžaduje využití speciální soupravy, je možné jí provozovat například na chytrém mobilním telefonu. Podobně jako u virtuální reality je uživatel, resp. jeho zařízení, umístěno do virtuálního prostoru a jeho pozice může být založena na kameře a dalších senzorech zařízení. Místo konstrukce celé virtuální scény je využito obrazu kamery zařízení, na který je virtuální scéna superponována. Tímto se technologie rozšířené reality snaží spojit reálný svět s virtuálním.

Aplikace Věnná města čeckých královen využívá prvků rozšířené reality za účelem zobrazení historických verzí skutečných budov v různých časech dne a za různého počasí. Aplikace je stavěná modulárně, jako jednotlivé moduly jsou oddělené algoritmy zjišťující umístění zařízení z různých zdrojů informací. Práce je týmového charakteru a je tvořena ve spolupráci s prací Jaroslava Štěpána [7], jejíž tématem je právě tvorba zmíněných modulů.

Kapitola 1

Cíl práce

Cílem práce je tvorba prototypu aplikace, která se bude zabývat zobrazením virtuálních modelů budov na místě jejich reálného umístění ve skutečném světě za použití prvků rozšířené reality. Prototyp bude sloužit jako základ pro tvorbu komplexní aplikace, který bude možné využít i k jiným účelům. Pro usnadnění implementace nových algoritmů zpracování obrazu pro účely rozšířené reality bude abstrahovat a zjednodušovat přístup k senzorům a dalším částem zařízení s operačním systémem Android.

Zároveň se práce zabývá automatickým nahráváním výsledné aplikace na obchod Google Play Store prostřednictvím systému Gitlab CI a nastavením systémů, které s tím souvisí.

Kapitola 2

Analýza

Tato kapitola se zaměřuje na analýzu existujících aplikací, zejména jejich uživatelského rozhraní, za účelem zlepšení výsledného návrhu uživatelského rozhraní prototypu. Dále obsahuje výběr knihovny, která bude použita pro realizaci vzoru dependency injection. Nakonec jsou uvedeny funkční a nefunkční požadavky na aplikaci.

2.1 Existující mobilní aplikace využívající rozšířenou realitu

Za účelem vytvoření dobrého návrhu uživatelského rozhraní prototypu aplikace, která je předmětem této práce, bylo zvoleno několik již existujících mobilních aplikací, které využívají prvků rozšířené reality k umístění virtuálních objektů do reálné scény. Pro účely analýzy byly zahrnuty pouze aplikace, které jsou dostupné bezplatně v úplné nebo omezené formě na portálu Google Play Store.

Jednotlivá řešení jsou hodnocena na základě Nielsenovo heuristik [8] a sekce Google Augmented Reality Design Guidelines, která se zabývá interakcí s uživatelem. Hodnocena je pouze část aplikace, která využívá prvků rozšířené reality. Z Nielsenovo heuristik jsou brány v potaz zejména na následující části:

- **Viditelnost stavu systému** Uživatel by vždy měl být dostatečně informován o tom, co se v aplikaci děje.
- **Shoda systému s reálným světem** Aplikace by měla využívat konvencí reálného světa, což je u aplikací, které využívají rozšířenou realitu, obzvláště důležité.
- **Uživatelská kontrola a svoboda** Aplikace by měla uživateli umožnit snadno se dostat z nežádoucího stavu, ideálně s podporou kroku zpět.

- **Konzistence se standardy platformy** Rozhraní aplikace by mělo využívat standardních prvků uživatelského rozhraní platformy, na které aplikace běží.
- **Prevence chyb** Uživatel by ideálně neměl mít možnost aplikaci dostat do chybového stavu, popřípadě by měla aplikace včas varovat před provedení akce, která by k chybovému stavu mohla směřovat.

Za každou adekvátně splněnou heuristiku je aplikaci udělen jeden bod, celkem aplikace může za splnění všech zkoumaných Nielsenovo heuristik získat 5 bodů. Následující Nielsenovy heuristiky nejsou brány v potaz, jelikož již jsou pro účely rozšířené reality zahrnuty v Google Augmented Reality Design Guidelines a příslušné části bodování:

- **Rozpoznání místo vzpomínání** Uživatel by neměl být nucen zapamatovat si složité postupy. Ovládání by mělo být intuitivní.
- **Estetický a minimalistický design** Aplikace by uživatele neměla zatěžovat informacemí, které nejsou aktuálně relevantní.
- **Smysluplnost chybových hlášení** Chybové hlášky mají být podány v jednoduchém jazyce.
- **Nápověda a návody** Jakákoliv dokumentace a návody by měly být snadno přístupné a prohledávatelné.
- **Flexibilní a efektivní použití** Aplikace by měla být jednak snadno použitelná pro nového uživatele, ale také by měla umožňovat rychlejší provedení akcí pro pokročilé uživatele, například zkratkami. Tuto heuristiku považuji za nedůležitou pro využití rozšířené reality, jelikož jde jednak o zařízení s omezenými možnostmi zkratek, ale také protože se většinou jedná o jednoduché akce, pro které možnost zkratky nemá smysl.

Z Google Augmented Reality Design Guidelines jsou k analytickému hodnocení vybrány následující části sekce zabývající se interakcí s uživatelem - UI [9]:

- **Věrohodný svět a jednoduchost použití** Zážitek uživatele by neměl být přerušen 2D vyskakovacími okny, dialogy a oznámeními. Takové elementy narušují věrohodnost virtuálního světa.
- **Intuitivní ovládání** Pokud aplikace nutně potřebuje ovládání pomocí 2D tlačítek, mělo by jich být minimálně. Většina interakce by se měla odehrávat ve virtuálním světě.
- **Postupné vysvětlení ovládání** Spíše, než učit uživatele všechny prvky ovládání najednou, je vhodné ovládání vysvětlovat vždy, když na něj uživatel poprvé narazí.

- **Snadné zotavení z chyb** Aplikace by měla jasně indikovat způsob, jakým je daný problém možné vyřešit a za chyby by nikdy neměla obviňovat uživatele.

S ohledem na sekci interakce s uživatelem - UX [10] pak bylo hodnoceno následující:

- **Jasný přechod z/do rozšířené reality** Z klasického 2D uživatelského rozhraní do rozšířené reality a zpět by měl uživatele provázet jasný vizuální přechod, například plynulým přechodem do černé a následně na obraz kamery.
- **Vnitřek virtuálního objektu** Pokud uživatel umístí kameru uvnitř virtuálního objektu, je důležité mu dát najevo, že by zde nemělo zařízení být, například rozmazáním obrazu.
- **Reset** Uživatel by se měl mít možnost jednoduše dostat do původního stavu.

Za splnění každého z těchto bodů byl aplikaci přidělen jeden bod, celkem tedy za splnění všech sledovaných částí Google Augmented Reality Design Guidelines může aplikace získat až 7 bodů.

Kromě výše zmíněných heuristik je navíc na aplikaci pozorováno, zda umožňuje, ať již manuálně, nebo automaticky, zpozdit obraz z kamery zařízení na základě rychlosti zpracování daného obrazu pro správné umístění prvků rozšířené reality. Zpožděné zobrazení by mělo za cíl párovat zobrazovat objekt přímo se snímkem, podle kterého byl do světa umístěn. Tím by bylo zamezeno opožděnému posouvání objektu na cílovou pozici, které je nejvíce viditelné při rychlém natočení nebo posunu kamery.

2.1.1 Pokémon GO

Pokémon GO je hra pro mobilní zařízení se systémem Android. Úkolem hráče je mimo jiné sbírat virtuální monstra do své sbírky. Hra je založena na umístění a pohybu hráče za použití GPS a dalších polohovacích mechanismů. Prvky rozšířené reality se ve hře Pokémon GO objevují při chytání virtuálních monster.

Stav aplikace je v analyzované části dostatečně vidět. Je indikováno, kde se hledané monstrum schovává, kolik zbývá času, než opět zmizí, další detaily jsou hráči sděleny hláškami v době, kdy jsou relevantní. Načítání aplikace bylo jasně indikováno a po načtení nebyl pozorováno žádné výrazné zaseknutí aplikace. Shodu s realitou nelze snadno hodnotit, jelikož jde o smyšlený způsob chytání monster, nicméně objekt tvaru koule, který je na chytání použitý, se ve hře chová podle očekávání. Z analyzované části aplikace se lze snadno dostat pryč například stiskem tlačítka zpět na zařízení, hra tedy splňuje uživatelskou

Obrázek 2.1: Snímek aplikace Pokemon GO [1]

kontrolu a svobodu. Rozhraní analyzované části obsahuje minimální počet ovládacích prvků, jelikož je styl hry jednoduchý, nicméně prvky, které nejsou hře výhradně specifické, jako je například ikona fotoaparátu k pořízení snímku monstra, splňují standardy platformy. Z povahy aplikace nelze předejít tomu, aby se uživatel dostal do chybového stavu, jedním z chybových stavů totiž můžou být špatné podmínky prostředí pro zpracování obrazu a nalezení pro hru nutného plochého prostoru, například za špatných světelných podmínek.

Některé ze sledovaných Google Augmented Reality Design Guidelines hra dobře nesplňuje. Hra obsahuje v analyzované části 2D vyskakovací okna. Jde hlavně o chybová hlášení, nicméně prezentace těchto hlášení narušují věrohodnost virtuálního světa. Ovládání hry je intuitivní a dodatečných 2D tlačítek je minimální množství, tudíž tento bod hra splňuje. Při prvním spuštění hry jsou důležité prvky ovládání vysvětlovány, když na ně hráč narazí, čímž je bod postupného vysvětlení ovládání splněn. Díky chybovým hláškám, které obsahují stručné instrukce, jak se z chybových stavů dostat (např. nutnost viditelného plochého povrchu), hra splňuje snadné zotavení z chyb. Přechod do rozšířené reality je také jasně indikován. Dovnitř virtuálních objektů se ve hře není možné dostat, jelikož chytaná monstra utečou dříve, než se k nim kamera dostane dostatečně blízko, čímž končí část hry s prvky rozšířené reality. Posledním sledovaným bodem je možnost resetování virtuálního světa, nicméně vzhledem k povaze hry takovou funkci není možné realizovat. Celkem

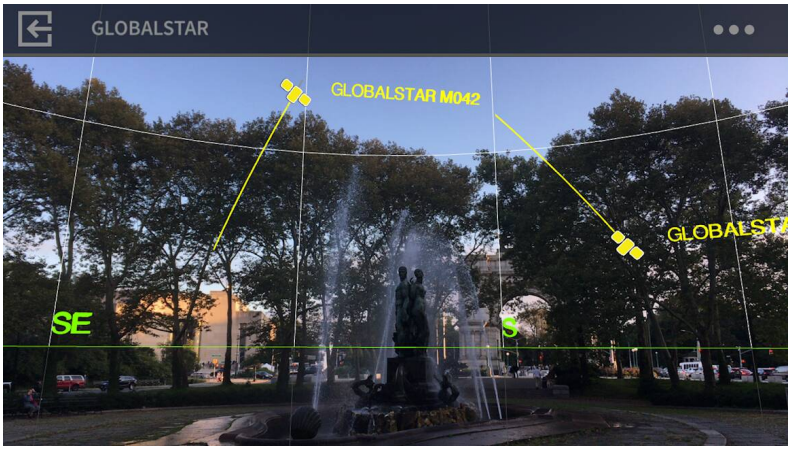
tedy hra splňuje 5 bodů.

Hra nevyužívá zpoždění obrazu kamery k zarovnání monster a dalších virtuálních elementů na správné místo relativně ke zobrazovanému snímku, dochází tedy k malému opoždění umístění virtuálních elementů.

Parametr	Hodnota
Nielsenovo heuristiky	4
Google Augmented Reality Design Guidelines	5
Zpoždění obrazu kamery	NE
Celkové hodnocení	**9**

2.1.2 Satellite-AR

Účelem aplikace Sattelite-AR je zobrazit aktuální pozice satelitů a družic nad obrazem získaným z kamery zařízení [2]. Aplikace k umístění objektu ale nevyužívá obrazu z kamery. Používá polohovací nástroje, jako je GPS, ke zjištění obecné lokace zařízení a vymezení relevantních družic. Následně pak využívá další senzory, jako např. gyroskop, k umístění virtuálních objektů.



Obrázek 2.2: Snímek aplikace Satellite-AR [2]

Aplikace svůj stav v režimu rozšířené reality občas nedává dostatečně najevo, aplikace se dostala mimo jiné do stavu, kdy se virtuální objekty vůbec nehýbaly s pohybem zařízení a nebylo jasné, proč tomu tak je. Ikony a tlačítka, které aplikace používá, se shodují s jejich reálnými významy a zároveň se shodují s příslušnými standardy platformy Android. Jelikož v některých případech nefunguje typické tlačítko zpět, dokonce celá aplikace občas nereaguje na uživatelský vstup, nesplňuje heuristiku uživatelské kontroly a svobody. Chybový stav nebyl v části rozšířené reality aplikace pozorován a vzhledem k tomu, že k umístění objektů nepoužívá obraz kamery, pravděpodobně nelze

snadno chybového stavu docílit. Za splněné Nielsenovy heuristiky tedy aplikace dostává 3 body.

Svět aplikace ztrácí na věrohodnosti hlavně kvůli tomu, že s každou volbou virtuálního objektu dojde ke zobrazení 2D okna s informacemi o daném objektu. Zobrazené ovládání aplikace v módu rozšířené reality je minimální a nestaví se do cesty uživateli, zároveň obvyklým akcím a ikonám odpovídají obvykle očekávané výsledky, jako je například odchod ze sekce rozšířené reality tlačítkem v levém horním rohu obrazovky, jak je vidět na obrázku 2.2, což ovládání činí intuitivním. Vzhledem k jednoduchosti aplikace také není třeba prvky ovládání vysvětlovat a takové vysvětlení opravdu aplikace neobsahuje. Hlášení aplikace, když je například potřeba kalibrovat nastavení senzorů, které aplikace využívá, jsou smysluplná a obsahují jasné instrukce pro uživatele, čímž je splněno snadné zotavení z chyb. Přechod do rozšířené reality je v aplikaci zprostředkován samostatnou obrazovkou načítání, nicméně vyskakovací 2D okna při volbě virtuálního objektu by se vzhledem k jejich velikosti dala považovat za výstup z rozšířené reality a v takovém případě žádný přechod aplikace neposkytne. Vzhledem k povaze aplikace nelze kameru umístit dovnitř virtuálních objektů, čímž je splněna ochrana jejich vnitřku. Aplikace neobsahuje možnost resetu virtuální scény. Celkem z této sekce aplikace dostává 4 body.

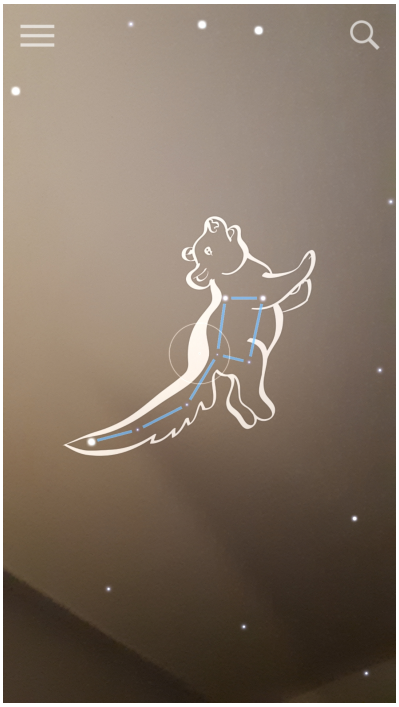
Vzhledem k tomu, že aplikace k umístění objektů nepoužívá obrazu kamery zařízení, nebylo by jednoduché obraz kamery opozdit tak, aby vždy virtuální objekty odpovídaly natočení zařízení a aplikace se vyvarovala opožděnému posouvání objektů. Aplikace tedy zpoždění obrazu kamery nepoužívá.

Parametr	Hodnota
Nielsenovo heuristiky	3
Google Augmented Reality Design Guidelines	4
Zpoždění obrazu kamery	NE
Celkové hodnocení	**7**

2.1.3 SkyView Free

SkyView Free je aplikace pro platformu Android, která svým uživatelům umožňuje sledovat vesmírná tělesa, souhvězdí a satelity. Zároveň obsahuje mód rozšířené reality, ve kterém lze sledovat pohyb těles ve vztahu k reálnému světu [3]. Aplikace však za účelem rozšířené reality nezpracovává obraz kamery, k umístění objektů používá ostatní senzory zařízení, jako je např. akcelerometr nebo gyroskop.

Stav aplikace je dobře vidět, neobsahuje znatelná zaseknutí ani dlouhé pauzy mezi akcemi. Aplikace používá ikony a prvky typické pro platformu Android a používá je tak, že se chovají podle očekávání na základě svých protějšků z reálného světa, například lupa sloužící k vyhledávání. Je tedy splněna nejen shoda se standardy platformy, ale také shoda s realitou. Vzhledem



Obrázek 2.3: Snímek aplikace SkyView Free [3]

k tomu, že v aplikací podle očekávání funguje tlačítko fyzické zpět zařízení a tomu, že aplikace neobsahuje ve sledované části dlouhotrvající akce, splňuje také uživatelskou kontrolu a svobodu. Pro umístění virtuálních objektů není použito kamery zařízení, není zde tedy prostor pro chybový stav způsobený špatným osvětlením prostředí. Za splněné Nielsenovy heuristiky aplikace SkyView Free získává 5 bodů.

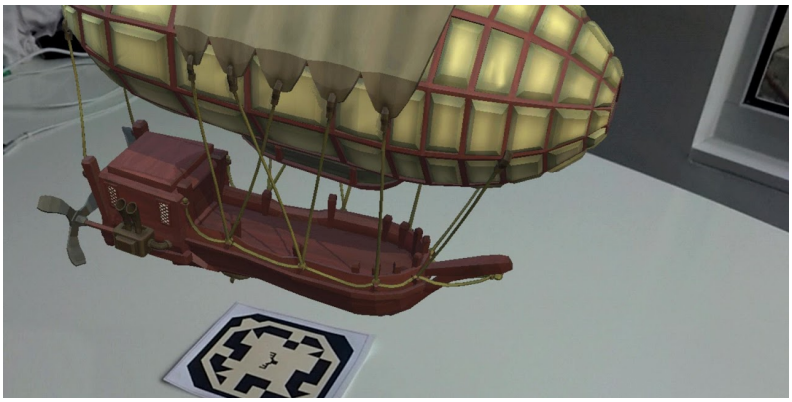
2D prvky jsou v části rozšířené reality aplikace téměř nepřítomné, jedná se hlavně o poloprůhledná tlačítka. Svět aplikace si tím zachovává věrohodnost. Jelikož tlačítka splňují standardy platformy a interakce s virtuálními objekty je jednoduchá, je ovládání aplikace intuitivní. Počet 2D tlačítek je zároveň nízký. Díky intuitivnosti ovládání není nutné poskytnout uživateli jeho bližší vysvětlení, aplikace tedy své ovládání nevysvětluje. Jelikož část aplikace využívající prvků rozšířené reality chybové stavy ohledně právě těchto prvků neobsahuje, není nutné se z takových chyb zotavovat. Část rozšířené reality aplikace je její hlavní částí, do které uživatel vstupuje při jejím spuštění. Přechodem je tedy vlastní spuštění aplikace a úvodní obrazovka. Jelikož se aplikace zabývá pouze objekty mimo fyzicky možný dosah kamery a ty zobrazuje, není možné kameru do některého z virtuálních objektů umístit, čímž je ochrana vnitřku virtuálních objektů splněna. Zároveň také aplikace neobsahuje manipulaci s virtuálními objekty, nemá tedy potřebu funkce resetu.

Jelikož aplikace k umístění virtuálních objektů nepoužívá kameru zařízení, nemůže přesně zpozdit obraz kamery tak, aby umístění objektů přesněji odpovídalo natočení zařízení. Nijak tedy obraz kamery zpožděn není.

Parametr	Hodnota
Nielsenovo heuristiky	5
Google Augmented Reality Design Guidelines	6
Zpoždění obrazu kamery	NE
Celkové hodnocení	**11**

2.1.4 Table Top AR

Aplikace Table Top AR má za účel podpořit hraní stolních RPG pomocí snadné možnosti vizualizace různých herních elementů. Obsahuje různé budovy, nepřátele a jiné herní objekty, které mohou být umístěny na vytištěné značky umístěné kamkoliv na hrací plochu [4].



Obrázek 2.4: Snímek aplikace Table Top AR [4]

Stav aplikace je v analyzované části dobře vidět a aplikace se znatelně nepozastavuje. Aplikace nepoužívá standardní tlačítka a ikony, které jsou typické pro platformu Android. Zároveň, ačkoliv žádná sledovaná část aplikace striktně neporušuje heuristiku shody systému s reálným světem, mohly by pro některé akce být zvoleny vhodné ikony místo nápisů. Aplikace v části rozšířené reality nemá mnoho distinktivních stavů, nicméně z části rozšířené reality se nelze dostat typickým tlačítkem zpět, které je obvykle na platformě Android pro tento účel používáno, ale pouze 2D tlačítkem, které je součástí rozhraní aplikace. Chybové stavy v aplikaci Table Top AR vzhledem k prvkům rozšířené reality neexistují. Pokud kamera značku vidí, aplikace na ni umístí příslušný objekt. Z Nielsenovo heuristik tedy aplikace Table Top AR získává pouze jeden bod.

Aplikace v části rozšířené reality neobsahuje až na nutná tlačítka žádný 2D obsah, čímž dobře splňuje věrohodnost světa a jednoduchost použití. Veškerá

interakce s virtuálními objekty je prováděna interakcí s příslušnými značkami v reálném světě. 2D tlačítka aplikace obsahuje pouze dvě, která jsou nutná a relevantní po celou dobu běhu aplikace. Bod intuitivního ovládání je tím splněn. Veškeré nastavení virtuálního světa je prováděno mimo rozšířenou realitu, resp. před vstupem do ní. Není tedy nutné žádné prvky ovládání vysvětlovat. Aplikace nicméně nedává najevo, když nevidí žádnou značku, že by uživatel mohl kameru na nějakou značku namířit. Snadné zotavení z chyb tedy není splněno. Přechod do rozšířené reality je v aplikaci jasně označen přechodem z bílé. Aplikace ale nebrání uživateli kameru posunout dovnitř virtuálních objektů a nijak nedává uživateli najevo, že by na takovém místě kamera být neměla. Pomocí jednoho z tlačítek je uživateli umožněno resetovat virtuální scénu.

Během používání aplikace lze snadno pozorovat zpoždění virtuálních objektů za pohybem kamery. Lze tedy usoudit, že aplikace nepoužívá zpoždění obrazu. Zpoždění oproti pohybu značek je maskováno položením virtuální značky pod objekt, která se hýbe spolu s objektem, nicméně oproti obrazu reálného světa lze stále zpoždění umístění pozorovat.

Parametr	Hodnota
Nielsenovo heuristiky	1
Google Augmented Reality Design Guidelines	5
Zpoždění obrazu kamery	NE
Celkové hodnocení	**6**

2.1.5 IKEA Place

Aplikace IKEA Place slouží hlavně k vyzkoušení umístění nábytku, který je v katalogu firmy IKEA. Nábytek je možné umístit na plochu detekovanou aplikací, následně lze nábytek libovolně posouvat a otáčet [5]. Jelikož je umisťování nábytku pomocí prvků rozšířené reality hlavní částí aplikace, neobsahuje aplikace striktně 2D obrazovky, vždy je v pozadí pohled kamery. Aplikace využívá knihovny ARCore od firmy Google za účelem realizace prvků virtuální reality.

Stav aplikace je vždy dobře vidět. Objekt, se kterým je manipulováno, je viditelně zvýrazněn a nadzvednut a v aplikaci nebyly pozorovány žádné výrazné záseky. Symbolika, která je v aplikaci použita, se shoduje jednak s obvykle používanými symboly na platformě Android, také se jejich funkce shoduje s protistranou v reálném světě. Například tlačítko se symbolem plus podle očekávání slouží k přidání nového objektu do scény a vyvolá katalog s vyhledáváním, ve kterém je možné objekt zvolit. Tímto tedy aplikace splňuje heuristiku shody systému s reálným světem a heuristiku konzistence se standardy platformy. Uživatel se ve všech případech snadno může dostat zpět. Například položení předmětu, pokud jej uživatel nechtěl zvednout, je možné jak stiskem příslušného 2D tlačítka, tak dotykem mimo zvednutý objekt. Je tedy dodržena uživatelské svoboda a kontrola. Jelikož je umístění objektu do

Obrázek 2.5: Snímek aplikace IKEA Place [5]

světa založeno na obrazu kamery zařízení, existuje zde opět chybový stav nevhodných podmínek viditelnosti, kterému nelze zabránit. Aplikace tedy až na prevenci chyb splňuje sledované Nielsenovy heuristiky.

Body Google Augmented Reality Design Guidelines aplikace IKEA Place splňuje hůře než Nielsenovy heuristiky. Vzhledem k tomu, že až na zobrazení virtuálního nábytku je veškerá komunikace s uživatelem vedena prostřednictvím rozhraní, které připomíná posílání SMS zpráv, jedná se o 2D vyskakovací okna, tudíž dochází k výraznému narušení věrohodnosti světa. Na druhou stranu 2D tlačítka použitá k ovládání aplikace jsou k dispozici vždy jen ta relevantní k aktuální situaci a jsou umístěna vždy na stejném místě. Umisťování virtuálního nábytku se drží typických konvencí pro posun a otáčení na dotykovém displeji, čímž zaručuje svoji intuitivnost. Aplikace tedy splňuje bod intuitivního ovládání. Jelikož je po prvním startu aplikace uživatel krátce a postupně proveden jednotlivými ovládacími prvky, splňuje aplikace také postupné vysvětlení ovládání. Při hledání plochého prostoru pro umístění objektů však aplikace neposkytuje textovou informaci nebo dostatečný vizuální návod, co by měl uživatel dělat, a tedy nesplňuje snadné zotavení z chyb. Jelikož je aplikace již spouštěna do části rozšířené reality, přechodem je spuštění aplikace a bod přechodu je splněn. Kamera může být umístěna dovnitř virtuálního nábytku, a to i dovnitř jeho stěn, nicméně nikdy není uživateli naznačeno, že je kamera na nežádoucím místě. Vnitřek virtuálního objektu není nijak chrá-

něn. Aplikace poskytuje možnost resetu scény pouze při opětovném spuštění, nikoliv ale během následného používání, čímž nesplňuje tento bod.

Při používání aplikace IKEA Place se občas obraz pohybuje pomaleji a všechny umístěné objekty vypadají, že se dobře drží na správném místě, z čehož usuzuji, že aplikace zpožďuje snímky kamery na základě jejich zpracování aby pohyb virtuálních objektů vypadal reálněji.

Parametr	Hodnota
Nielsenovo heuristiky	4
Google Augmented Reality Design Guidelines	3
Zpoždění obrazu kamery	ANO
Celkové hodnocení	**7**

2.1.6 Knightfall™ AR

Mobilní hra Knightfall™ AR spočívá v obraně středověkého města pomocí lučištníků a katapultů. Hráč má zároveň možnost na virtuální mapě, na které přicházejí vlny nepřátel, umístit své jednotky nepřátelům do cesty. Kromě samotné hry umožňuje aplikace umisťovat postavy ze hry libovolně do prostoru, například pokoje, a animovat je [6].

Obrázek 2.6: Snímek aplikace Knightfall™ AR [6]

V několika momentech se aplikace na znatelnou dobu zasekla bez jakékoliv indikace důvodu, například při dohrání aktuální úrovně, čímž nesplňuje viditelnost stavu. Na druhou stranu ikony, které jsou v aplikaci použity, se funkcí shodují s protistranami v reálném světě. Ačkoliv se jedná o hru s vlastním vizuálním stylem uživatelského rozhraní, splňuje hra konvence, které jsou obecně v podobných hrách na platformě Android používány a splňuje tedy konzistenci se standardy platformy. Hlavním chybovým stavem, do kterého se aplikace může dostat, je omezená schopnost aplikace nalézt vhodnou plochu

pro umístění herních objektů, což může nastat například za špatných světelných podmínek, vzhledem k závislosti na obrazu kamery. Prevence chyb z tohoto důvodu nemůže být splněna. Jelikož se jedná o hru, existuje mnoho pro uživatele nežádoucích stavů, do kterých se lze dostat. Uživatel může použít speciální možnost obrany nebo zakoupit herní jednotku omylem a v takové situaci nelze udělat krok zpět. Pokud ale tyto situace nejsou počítány a jsou brány v potaz pouze situace, ze kterých by se uživatel měl mít možnost dostat i vzhledem k pravidlům hry, splňuje aplikace heuristiku uživatelské svobody. Jedinými nesplněnými heuristikami jsou tedy prevence chyb a viditelnost stavu systému.

Hra v mnoha případech obsahuje vyskakovací 2D okna, ať již jde o dokončení úrovně, nové odemčené možnosti nebo jiné oznámení. Krom těchto oken také věrohodnost světa narušuje řada herních tlačítek a informací o průběhu hry umístěných po stranách obrazovky. Ovládání hry je intuitivní, mimo jiné díky použitým ikonám na tlačítkách. Uživatel má také řadu možností interakce s 3D objekty pomocí tlačítek umístěných přímo do prostředí rozšířené reality. Během první úrovně hra hráče provede svým ovládáním postupně a srozumitelně. Hra poskytuje uživateli srozumitelné instrukce při chybových stavech, jako je hledání plochy použitelné pro umístění prvků rozšířené reality, a umožňuje tedy snadné zotavení z chyb. Aplikace se téměř výhradně pohybuje v sekci rozšířené reality, nicméně přechody mezi plně 2D obrazovkami a sekcí rozšířené reality jsou dobře odděleny přechodem do černé. Vnitřek virtuálních objektů není v aplikaci nijak chráněn a uživatel může snadno kameru dovnitř objektu posunout. Jelikož se jedná o hru, která má daný průběh a pravidla, nelze scénu rozšířené reality snadno uvést do původního stavu. Celkem aplikace splňuje čtyři body Google Augmented Reality Design Guidelines.

Podobně jako aplikace IKEA Place, hra Knightfall™ AR používá pro realizaci rozšířené reality knihovnu ARCore od firmy Google. Za běhu aplikace je znát správné zarovnání virtuálních objektů vzhledem k pohybu kamery zpožděním snímků kamery tak, aby pro výsledné zobrazení objektů byl použit snímek kamery, ze kterého bylo umístění vypočteno. Takto zpožděný obraz silně podporuje reálnost virtuálních objektů v aplikaci.

Parametr	Hodnota
Nielsenovo heuristiky	3
Google Augmented Reality Design Guidelines	4
Zpoždění obrazu kamery	ANO
Celkové hodnocení	**7**

2.1.7 Souhrn analýzy zvolených aplikací

Z analyzovaných aplikací nejlépe dopadla aplikace SkyView Free, přestože k umístění objektů nepoužívá obrazu kamery. Lze si ale z jejího uživatelského rozhraní lecos odnést, hlavně poloprůhledná tlačítka, díky čemuž méně kradou

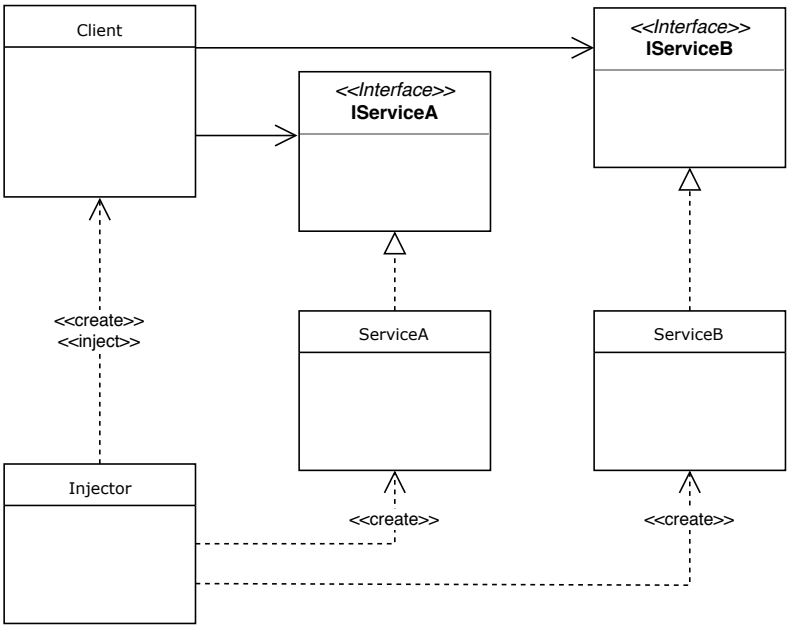
pozornost od samotných virtuálních objektů. Zároveň aplikace dobře realizuje poskytování dalších informací opět částečně průhledným zobrazením ve snaze lépe splynout s pozadím.

Aplikace IKEA Place a Knightfall^{TM} AR se sice umístily relativně nízko, jako u jediných byla u nich ale pozorována velice dobrá synchronizace umístění objektů a posunu kamery a jako u jediných aplikací zde bylo pozorováno zpomalení obrazu kamery, aby nedocházelo k dohánění kamery objektem. Tím aplikace dosáhly lepší věrohodnosti svého virtuálního světa.

Aplikace	Heur.	Google AR	Zpož. obrazu	**Celkem**
SkyView Free	5	6	NE	**11**
Pokémon GO	4	5	NE	**9**
Sattelite-AR	3	4	NE	**7**
IKEA Place	4	3	ANO	**7**
Knightfall^{TM} AR	4	3	ANO	**7**
Table Top AR	1	5	NE	**6**

2.2 Frameworky pro realizaci vzoru dependency injection

Dependency injection je architektonický vzor je v zásadě způsob externí konfigurace a propojování vytvářených objektů a služeb. Jde o alternativu případu, kdy se objekt konfiguruje sám. Jelikož jde o konfiguraci při tvorbě objektu, je typické vzor realizovat za použití jejich konstruktorů, nicméně lze také použít veřejně deklarované vlastnosti nebo setter metody [11].



Obrázek 2.7: Schema dependency injection, převzato z [7]

Na obrázku 2.7 je znázorněn jednoduchý diagram vzoru, který se skládá z následujících částí:

- **Injektor** Zvláštní služba, jejíž úkolem je konstruovat nastavené závislosti a objektům, které si o ně žádají, je jedním ze zmíněných způsobů nastavovat.

- **Služby** Závislosti nastavovaných objektů konstruované injektorem. Mohou definovat vlastní závislosti, které injektor před jejich konstrukcí musí připravit.

- **Rozhraní** Kontrakt, pro jehož splnění musí injektor poskytnout příslušnou službu. Právě rozhraní jsou použita pro definici závislostí, aby mohly závislosti být splněny více vyhovujícími službami.

- **Klient** Objekt, jehož závislosti jsou právě splňovány. Může se jednat také o jednu ze služeb, která má vlastní závislosti ke splnění.

Právě tento architektonický vzor byl pro návrh aplikace zvolen za účelem oddělení jednotlivých modulů, zejména modulů zpracování obrazu, aby byla umožněna jejich snadná výměna, například z důvodu nedostatečného výkonu zařízení. Jaroslav Štěpán, který souběžně vyvíjí moduly zpracování obrazu, ve své práci uvádí odůvodnění, proč byl vybrán právě tento vzor [7].

Aby byla usnadněna implementace vzoru, bylo rozhodnuto použít jednoho z dostupných frameworků zajišťujících dependency injection pro jazyk Java. Čtyři vybrané frameworky jsou postupně ohodnoceny na základě jejich stavu vývoje (poslední aktualizace) a kompatibility se systémem Android, jejíž kvalita se u různých frameworků liší, ačkoliv v zásadě všechny vybrané frameworky platformu Android podporují. Zároveň je pozorována velikost frameworků, jelikož je pro aplikaci určenou pro mobilní zařízení důležité udržovat celkovou velikost alespoň samotné aplikace na minimu, jelikož bude pro správný chod aplikace potřebovat další, potenciálně objemná, data. Každé ze tří kritérií je hodnoceno až dvěma body, potenciální maximální počet bodů, kterého může framework dosáhnout je tedy šest.

2.2.1 Guice

Guice je framework umožňující snadnou implementaci architektonického vzoru dependency injection spravovaný firmou Google. Použití dekorátoru `@Inject` umožňuje definovat závislosti objektů, které budou splněny za pomoci příslušných factory tříd, tedy tříd, jejichž jediným účelem je správným způsobem vytvářet nové objekty. Zároveň se framework snaží ulehčit vývoj srozumitelným kódem a smysluplnými chybovými hláškami. [12].

Ačkoliv má z analyzovaných frameworků nejširší možnosti použití, využívá k realizaci vzoru reflekce jazyka Java, tedy nahlížení do metadat objektů a tříd za běhu aplikace. Na platformě Android je reflekce sice podporována, nese s sebou ale nemalé zpomalení, které se může projevit například při startu aplikace [13]. Poslední vydaná stabilní verze frameworku Guice je 4.2.2 z 29. října 2018 a repozitář se zdrojovými kódy je stále aktivní s pravidelnými změnami. Guice tedy lze považovat za aktivní a dobře udržovaný framework.

Z analyzovaných frameworků jde také o největší z hlediska velikosti knihovny, díky čemuž ztrácí v tomto kritériu bod.

Parametr	Hodnota
Podpora OS Android	1
Aktivita vývoje	2
Velikost knihovny	1
Celkové hodnocení	**4**

2.2.2 Dagger

Framework Dagger, aktuálně již v druhé verzi, je také pod správou firmy Google. Na rozdíl od Guice silně využívá generace kódu, nikoliv reflekce, k zajištění splnění závislostí. Vyhnutí se reflekci je jeden z hlavních důvodů jeho vzniku [14].

Díky tomu, že využívá generovaného kódu k implementaci dependency injection, lépe podporuje platformu Android. Oficiální dokumentace dokonce obsahuje pro platformu Android doporučená použití a příklady, jak pracovat s Android aktivitami v rámci frameworku Dagger. Framework sice neřeší cyklické závislosti automaticky, to ale pro účely této práce není problém.

S aktuální stabilní verzí 2.21 vydanou 16. ledna 2019 a krátkým náhledem do repozitáře se zdrojovými kódy, který je pravidelně aktualizován, lze vývoj této knihovny považovat za aktivní.

Z vybraných frameworků jde o jeden ze dvou nejmenších z hlediska velikosti knihovny.

Parametr	Hodnota
Podpora OS Android	2
Aktivita vývoje	2
Velikost knihovny	2
Celkové hodnocení	**6**

2.2.3 Tiger

Dependency injection framework Tiger je silně inspirován frameworkem Dagger. Má za cíl být ve své oblasti nejrychlejším a velice jednoduchým na použití [15].

Poslední aktualizace frameworku byla v květnu 2018, přičemž by se projekt dal považovat za stále aktivní. Narozdíl od ostatních frameworků však postrádá jakékoliv oficiální vydání nebo stabilní verzi, ani oficiální repozitář neobsahuje nikde označené vydání.

Stejně jako Dagger, framework Tiger využívá pouze generovaný kód, díky čemuž snadno podporuje OS Android. Ačkoliv dokumentace se něm pouze lehce zmiňuje, oficiální repozitář obsahuje ukázkové kódy právě pro systém Android.

Parametr	Hodnota
Podpora OS Android	2
Aktivita vývoje	1
Velikost knihovny	1
Celkové hodnocení	**4**

2.2.4 Feather

Feather je framework pro realizaci vzoru dependency injection, který si za cíl klade být nejméně náročný z hlediska velikosti, ideálně bez obětování výkonu a hlavně přehlednosti [16].

K zajištění závislostí používá framework Feather reflekce. Oficiální dokumentace sice zmiňuje příklad použití pro systém Android, nicméně z důvodu použití reflekce považuji podporu OS Android za horší, než u frameworků, které se reflekci zcela vyhýbají.

Jde také o nejmenší framework z analyzovaných, nicméně již není v aktivním vývoji, s poslední verzí 1.0 z října roku 2015 a neaktivním oficiálním repozitářem.

Parametr	Hodnota
Podpora OS Android	1
Aktivita vývoje	0
Velikost knihovny	2
Celkové hodnocení	**3**

2.2.5 Souhrn

Ve výběru nejlépe vyšel framework Dagger, který nejlépe kombinuje dobrou aktivitu vývoje a malou velikost knihovny. Zároveň používá generovaný kód při kompilaci oproti reflekci, což přispívá k rychlosti a podpoře OS Android.

Knihovna	Podpora OS Android	Aktivita	Velikost	**Celkem**
Dagger	2	2	2	**6**
Tiger	2	1	1	**4**
Guice	1	2	1	**4**
Feather	1	1	2	**3**

2.3 Další použité knihovny

Jádro má za úkol starat se o zajištění síťové komunikace se vzdáleným serverem, například pro stahování metadat budov, nebo pro zprostředkování komunikace modulů. Musí být také schopno stažené budovy správně vykreslit. Tyto dva úkony jsou samy o sobě komplexní a pro jejich realizaci, zejména jako součást prototypu aplikace, je vhodné použít již existující řešení ve formě softwarových knihoven.

2.3.1 Komunikace se vzdálenými REST službami

Jelikož pro realizace síťové komunikace s REST službami již existuje řada implementací ve formě knihoven, je vhodné jednu z těchto knihoven použít, než komunikaci a zpracování odpovědí implementovat znovu.

Jednou z takových knihoven je knihovna Retrofit. Umožňuje snadno použitím Java anotací definovat rozhraní REST služeb, následně pak generuje kód, který provádí samotnou komunikaci [17]. Knihovna má svůj vlastní systém rozšíření. Jedno z těchto rozšíření umožňuje odpovědi REST služeb převádět přímo do Java objektů příslušných tříd, což výrazně zjednodušuje následné zpracování vrácených dat.

Vzhledem ke snadné práci s touto knihovnou, faktu, že je knihovna stále udržována a již existujícím osobním zkušenostem je knihovna Retrofit vybrána k usnadnění implementace komunikace jádra se vzdálenými REST službami.

2.3.2 Vykreslování budov a obrazu kamery

Pro zjednodušení implementace rendereru jádra, tedy části, která má za úkol vykreslení modelů budov a zobrazení obrazu kamery na pozadí, je vybrána knihovna Sceneform od firmy Google. Důvodem volby knihovny Sceneform je její snadnější propojení s knihovnou ARCore, jelikož byla tato knihovna původně tvořena právě pro použití spolu s knihovnou ARCore [18]. Knihovna ARCore bude použita pro implementaci prototypu trackovacího modulu [7].

2.4 Funkční a nefunkční požadavky

Na základě instrukcí a požadavků zadavatele jsou definovány následující funkční a nefunkční požadavky na jádro mobilní aplikace.

Funkční požadavky jádra vůči modulům

- **FRM1 - Oddělení jádra od modulů rozpoznání obrazu** Jádro bude od modulu rozpoznání obrazu oddělené tak, aby moduly bylo možné snadno zaměnit. Mezi moduly a jádrem musí existovat jednotné rozhraní.

- **FRM2 - Automatický a manuální výběr modulu rozpoznání obrazu** Jádro bude schopné automaticky vybrat modul, který bude použit pro účely rozšířené reality. Zároveň bude jádro poskytovat možnost modul určit manuálně ze seznamu dostupných modulů.

- **FRM3 - Automatický výběr nejvhodnějšího modulu** Na základě parametrů zařízení se jádro pokusí vybrat takový modul, o kterém je známo, že s daným zařízením funguje nejlépe z dostupných modulů. Pokud taková informace není k dispozici, může jádro za běhu moduly měnit, aby modul, který na zařízení funguje nejlépe, nalezlo.

- **FRM4 - Jednoduché aplikační rozhraní jádra vůči modulům** Aby se vývoj jednotlivých modulů mohl co nejvíce soustředit na samotné algoritmy rozpoznání obrazu, bude jediným požadavkem na modul splnění navrženého rozhraní. Zároveň bude jádro poskytovat zjednodušené rozhraní pro síťovou komunikaci, ukládání dat do úložiště zařízení, přístup k senzorům a kameře zařízení.

- **FRM5 - Měření využití systémových zdrojů moduly** Jádro bude za účelem výběru nejvhodnějšího modulu sbírat průběžně o modulech informace o využití paměti zařízení, vytížení síťového připojení, využití úložiště zařízení a vytížení všech výpočetních jader procesoru.

- **FRM6 - Možnost specifikace kompatibilní verze API jádra** Modul bude mít možnost specifikovat cílovou verzi rozhraní jádra. Na základě specifikované verze pak jádro vybere pouze moduly kompatibilní s aktuální verzí, popřípadě nižší verzí za předpokladu, že je rozhraní zpětně kompatibilní vůči cílové verzi.

- **FRM7 - Možnost použití lokálního i vzdáleného modulu** Moduly bude možné využít jak implementované lokálně na zařízení uživatele, tak implementované na vzdáleném serveru, kterému budou posílány všechny potřebné informace pro výpočet.

Funkční požadavky aplikace

- **FRA1 - Zjištění aktuální pozice** Aplikace bude za použití modulů rozpoznání obrazu schopna určit aktuální pozici zařízení.

- **FRA2 - Sledování pohybu zařízení** Pomocí modulů využívajících technik rozpoznání obrazu bude aplikace sledovat pohyb zařízení a udržovat zjištěnou pozici aktuální.

- **FRA3 - Identifikace budov v okolí** Na základě zjištěné pozice a za pomoci vzdáleného serveru bude aplikace schopna zjistit, které zaznamenané budovy se nachází v okolí zařízení.

- **FRA4 - Automatický výběr modelů budov** Zobrazované modely budov budou automaticky vybírány na základě aktuálního počasí, času a data, a zvoleného historického období, ze kterého by měla budova pocházet.

- **FRA5 - Manuální výběr modelů budov** Aplikace bude umožňovat manuální volbu parametrů, které ovlivňují výběr zobrazovaných modelů budov.

- **FRA6 - Zobrazení modelu budov** Použitím zjištěné pozice zařízení a informací o okolních budovách bude aplikace zobrazovat alternativní modely budov na jejich příslušných pozicích.

- **FRA7 - Budoucí dodatečné informace o budovách** Návrh a implementace aplikace bude počítat s budoucím rozšířením o stažení a prezentaci dodatečných informací o zobrazovaných budovách, například rozšiřující text nebo zvukový doprovod.

Nefunkční požadavky

- **NR1** Aplikace bude implementována pro platformu Android 9.0. Možnost zpřístupnění aplikace prostřednictvím Google Play je limitována zvolenou verzí platformy Android, pro kterou je aplikace implementována. Jelikož se počítá s budoucím rozšířením projektu, je vhodné vybrat verzi platformy 9.0, která bude začátkem srpna roku 2019 požadavkem Google Play.

- **NR2** Aplikace bude provedena v českém jazyce.

- **NR3** Aplikace bude využívat standardní lokalizační prostředky dostupné pro systém Android za účelem umožnění snadného překladu aplikace do jiného jazyka.

- **NR4** Uživatelské rozhraní aplikace bude v rámci možností splňovat Augmented Reality Design Guidelines za využití poznatků z analýzy podobných aplikací.
- **NR5** Pro aplikaci bude využita co nejvolnější licence z pohledu možností využití.

Výběr licence

Licence zdrojového kódu práce byla vybírána na základě její kompatibility s licencemi použitých knihoven. Jak DI framework Dagger2, tak další použité knihovny od firmy Google jsou vydané pod licencí Apache 2.0 [19]. Stejnou licenci využívá knihovna Sceneform a také knihovna Retrofit. Jelikož licence Apache 2.0 dovoluje jak vydání výsledného kódu pod jinou licencí, tak komerční využití, není použití těchto knihoven příliš omezující.

Předpokladem je, že tato práce bude dále využita a rozšířena. Je tedy vhodné využít v tomto ohledu co nejvolnější licence. Dobrou volbou, zejména díky své jednoduchosti a srozumitelnosti, je licence MIT [20], které byla pro toto dílo vybrána.

Kapitola 3

Návrh

V této kapitole je nejprve popsána obecná architektura, kterou se bude zbytek návrhu aplikace řídit. Následuje návrh důležitých částí mobilní aplikace, jmenovitě inicializačního procesu, rozhraní pro moduly, sledování výkonu a využití systémových prostředků, komunikace se serverem projektu a tříd, které realizují nejdůležitější funkcionality jádra. Nakonec bude navrženo uživatelské rozhraní aplikace.

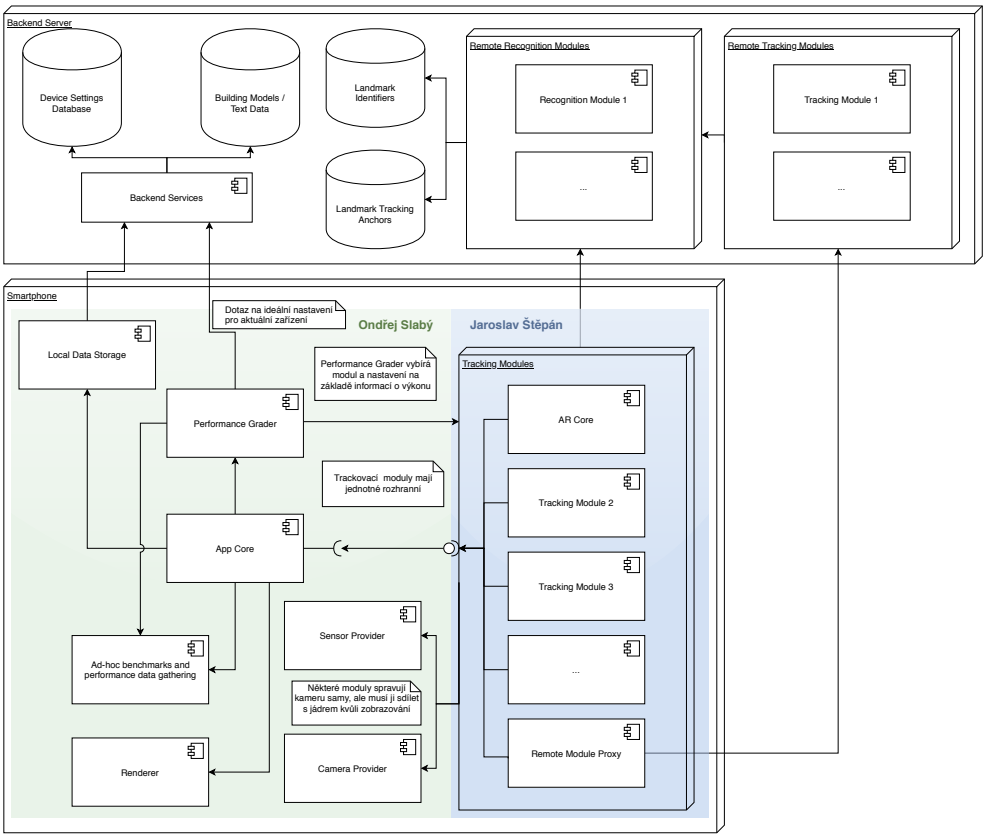
3.1 Obecná architektura

Mobilní aplikace, která bude běžet na uživatelských zařízeních, bude za účelem získávání dat o budovách a využití vzdálených modulů pro lokalizaci a trackování komunikovat se serverem. Tento server je také součástí projektu, nicméně v této práci je pouze zmíněn z důvodu spolupráce, jeho návrh a implementace není součástí této práce.

Na obrázku 3.1 je zobrazena architektura aplikace spolu s oddělením modulů realizujících rozpoznání obrazu, jejich zapojení do jádra a propojení s poskytovatelem kamery, poskytovatelem senzorů a měřičem výkonu. Poskytnuté senzory závisí na požadavcích modulu, každý modul však bude mít k dispozici kameru. Návrh rozhraní modulů bude počítat jak s případem, kdy modul bude od jádra dostávat snímky z kamery na požádání, tak s případem, že modul bude kameru spravovat sám, kdy ale musí jádru poskytnout sdílený přístup ke kameře za účelem vykreslování.

Další důležité součásti mobilní aplikace, které nespadají do samostatně vyvíjených modulů, jsou následující:

- **Lokální úložiště** Jelikož aplikace nebude obsahovat všechny modely budov, které bude zobrazovat, ale bude tyto modely stahovat ze serveru, bude je také lokálně po dobu jejich relevance ukládat. Zároveň bude tato část umožňovat modulům snadno ukládat data, pokud to k výpočtu potřebují, a poskytovat informace o využití úložiště hodnotící části.



Obrázek 3.1: Obecná architektura aplikace

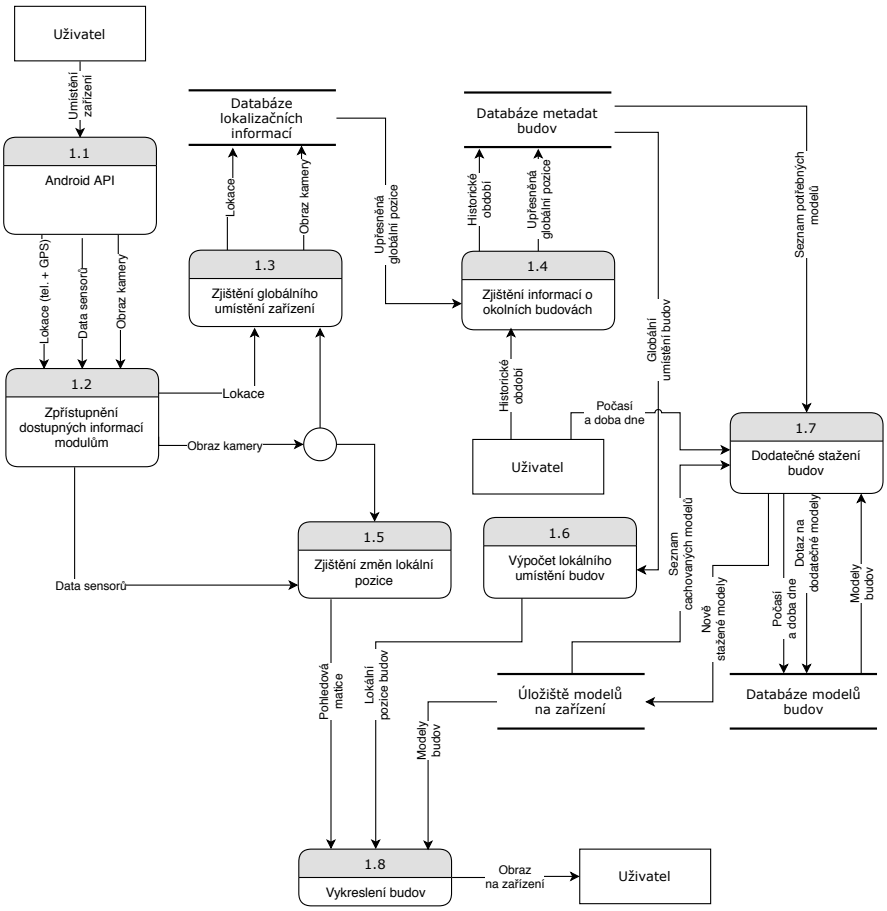
- **Renderer** Část, která slouží k vykreslování budov a snímků z kamery na pozadí. Informace o umístění budov získá jádro ze serveru spolu s modelem budovy, zatímco umístění kamery, resp. zařízení ve vztahu k budovám, dostane od aktivního modulu využívajícího rozpoznání obrazu.

- **Sběr výkonnostních dat** Za účelem výběru nejvhodnějšího modulu bude jádro sbírat informace o výkonu modulů a jejich využití systémových prostředků.

- **Poskytovatel kamery** Pokud modul nevyžaduje možnost spravovat kameru sám, bude mu skrz poskytovatele kamery umožněn přístup k aktuálnímu snímku. V opačném případě bude modul muset jádru poskytnout sdílený přístup ke kameře.

- **Poskytovatel senzorů** Každý modul může mít vlastní požadavky na senzory, které ke své funkci potřebuje, a to povinně, nebo nepovinně. Splnění nepovinných požadavků může znamenat preferenci modulu při výběru nejvhodnějšího modulu.

Možnost použití modulu rozpoznání obrazu, který bude implementován na serveru, bude implementována prostřednictvím modulu, který se jádru bude tvářit jako lokální a bude splňovat veškerá potřebná rozhraní, ale poskytnuté snímky z kamery bude posílat na vzdálený server. Modul implementovaný na serveru přijaté snímky zpracuje a výsledná data pošle zpět lokálnímu modulu, který je poskytne jádru.



Obrázek 3.2: Tok dat mezi komponenty aplikace

Na obrázku 3.2 je zobrazený průtok a transformace dat aplikací, od vstupu uživatele umístěním jeho zařízení až po výsledný obraz zobrazený uživateli. Pomocí GPS souřadnic a obrazu kamery je zjištěna zpřesněná pozice zařízení, pomocí které jsou voleny budovy ke zobrazení. Vybrané budovy jsou staženy, popřípadě načteny z mezipaměti na zařízení. Na základě obrazu kamery a dalších sensorů je následně zjištěna lokální pozice, resp. odchylka od zjištěné globální pozice, která je použita k umístění modelů budov do scény.

3.1.1 Inicializace jádra

V momentě, kdy uživatel spustí aplikaci, není ještě jasné, který z dostupných modulů bude použitý. Jelikož hlavním účelem aplikace je zobrazení historických budov pomocí prvků rozšířené reality a možnost manuálně volit modul, který bude za účelem dosažení tohoto cíle použit, je až druhotným účelem, musí jádro zvolit prvně použitý modul automaticky. Celý proces spuštění až po vstup do rozšířené reality je znázorněn na obrázku 3.3.

Seznam všech dostupných modulů je nejprve filtrován na základě senzorů, kterými zařízení disponuje. Každý modul jádru poskytuje informaci o tom, které senzory jsou pro jeho běh nezbytné. Zároveň mohou moduly specifikovat senzory, které sice nejsou pro běh modulu striktně nezbytné, jejich přítomnost ale může výpočet urychlit nebo upřesnit. Podrobnější popis rozhraní modulů využívajících rozpoznání obrazu je uveden v následující sekci.

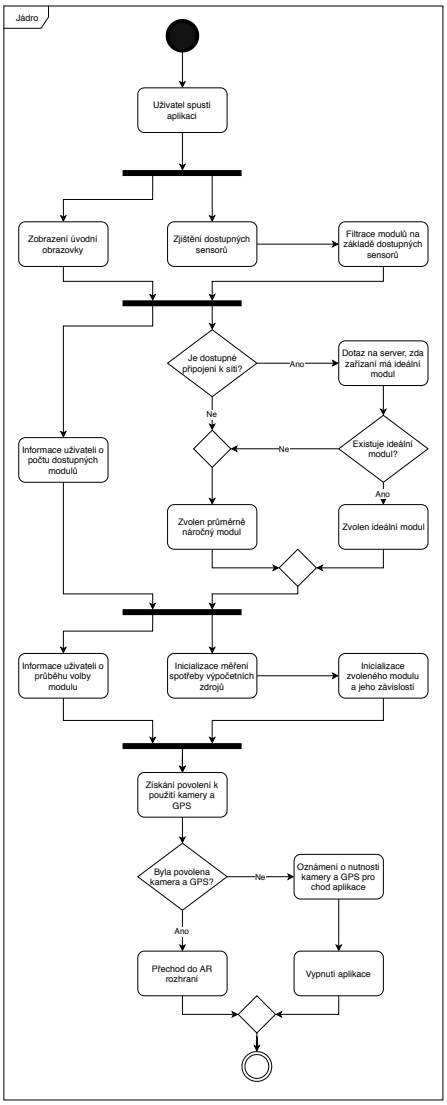
Zatímco uživatel uvidí úvodní obrazovku aplikace, jádro se pokusí navázat spojení se serverem, kterého se dotáže, na základě modelu zařízení a dalších informací o aktuálním hardwaru, který modul by měl být zvolen, aby bylo dosaženo nejvyššího výkonu s přijatelným využitím dostupných systémových prostředků. Databáze nejlepších konfigurací se bude postupně rozrůstat o další zařízení na základě testování vývojářského týmu, popřípadě prostřednictvím dat získaných ze zařízení, na kterých aplikace poběží, která dosud nejsou v databázi zaznamenána.

Server s databází nejlepších konfigurací zařízení nemusí být vždy dostupný, například z důvodu chybějícího připojení k síti. Zároveň může serveru chybět záznam o aktuálně používaném zařízení. V takovém případě musí jádro vybrat jeden z dostupných modulů, pro které jsou splněny všechny nutné požadavky. Primárně jsou brány v potaz moduly, pro které zařízení splňuje i dodatečné požadavky na dostupné senzory.

Každý z dostupných modulů poskytuje skrze své rozhraní jádru informaci o předpokládaném využití systémových zdrojů, pomocí níž jádro při prvotním výběru moduly řadí a vybírá průměrně vytěžující modul. Informace o předpokládaném zatížení je později použita i při volbě nového modulu v případě, že aktuální modul není přijatelně výkonný nebo nadměrně využívá systémové prostředky. Pro řazení modulů je použita i informace o splněných dodatečných požadavcích a to tak, že moduly s nesplněnými dodatečnými požadavky jsou považovány za méně výkonné.

3.1.2 Rozhraní modulů

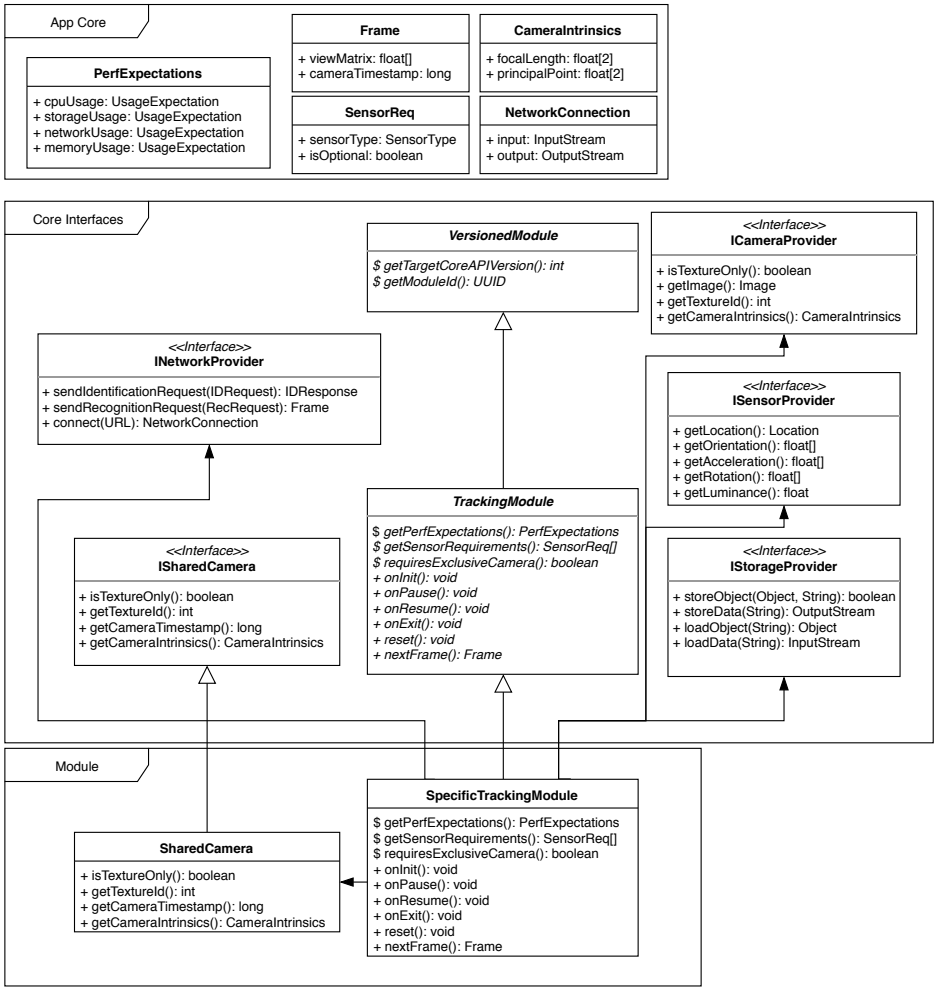
Jádro od jednotlivých modulů předpokládá splnění definovaného rozhraní. Zároveň jádro modulům za účelem zjednodušení přístupu a snížení nutnosti znát specifika práce se systémem Android poskytuje několik rozhraní pro přístup k senzorům zařízení, ke kameře a dalším systémovým prostředkům. UML diagramy tříd jsou v souladu s konvencemi tvořeny v anglickém jazyce.



Obrázek 3.3: Inicializační sekvence jádra

Diagram na obrázku 3.4 obsahuje navržené rozhraní modulu pro realizaci trackování i služeb, které modulu budou poskytnuty jádrem. Aby bylo možné určit úroveň kompatibility modulu s verzí rozhraní jádra, každý modul je potomkem třídy VersionedModule, která obsahuje statickou metodu, pomocí které poskytne modul jádru informaci o cílové verzi rozhraní jádra, pro kterou byl modul implementován. Jádro díky tomu může na základě vlastní znalosti o kompatibilitě mezi aktuální a cílovou verzí rozhraní určit, zda je modul možné použít.

Implementací abstraktních metod třídy TrackingModule následně modul poskytne jádru informaci o senzorech, které k práci potřebuje, a předpoklá-



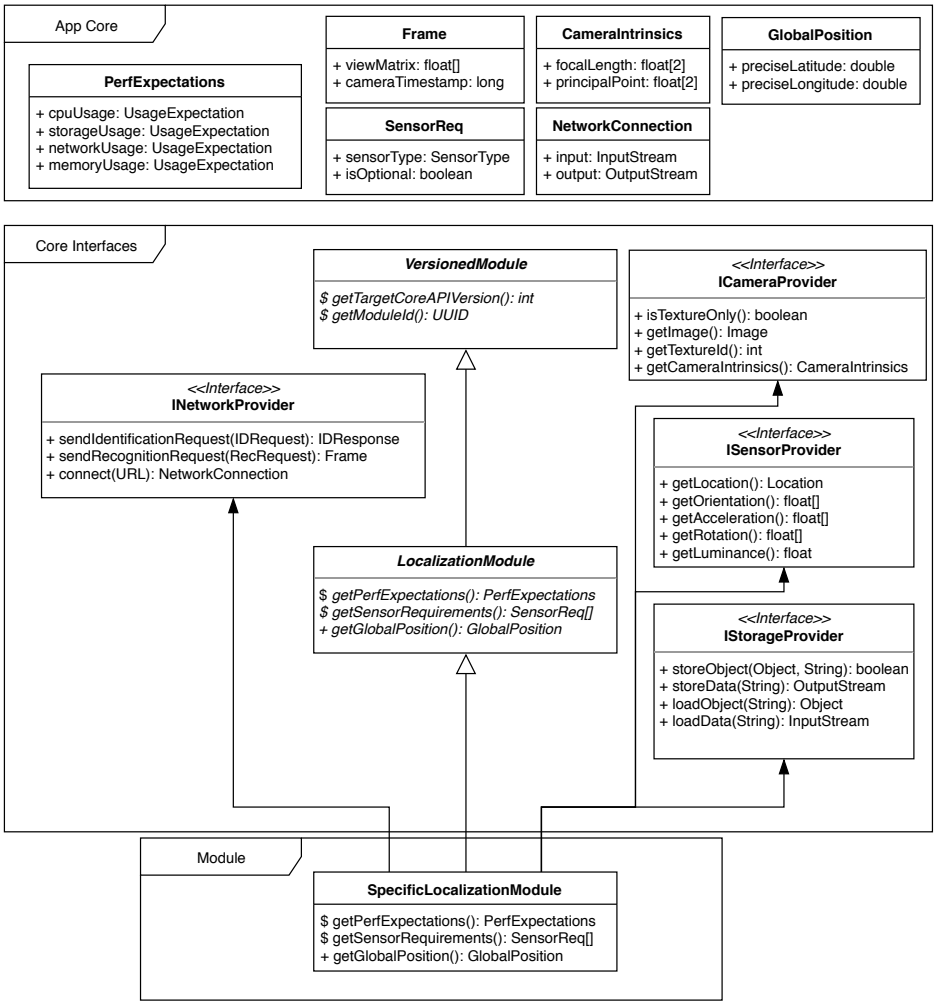
Obrázek 3.4: Diagram tříd API pro trackovací obrazu

daném výkonu modulu, který jádro využívá k výběru nejvhodnějšího modulu pro zařízení při startu a za běhu při potřebě modul změnit. Jádro skrz rozhraní této třídy také modulu dává informaci o aktuálním stavu životního cyklu a může modul požádat o reset aktuálně sledované pozice. K tomu může dojít, když jádro zjistí novou globální pozici od lokalizačního modulu a změní umístění modelů budov.

Modul zpracování obrazu může jádru dát najevo, že potřebuje sám spravovat kameru, například z nutnosti manuálního nastavení parametrů nebo z důvodu omezení knihovnou nebo frameworkem, který modul ke své práci používá. Pokud modul vyžaduje přímý přístup ke kameře, musí dát jádru k dispozici aktuální obraz kamery, aby jej mohlo jádro zobrazit na pozadí scény, a další informace, které jádro potřebuje za účelem poskytnutí těchto informací lokalizačnímu modulu.

Jádro modulu pomocí rozhraní ICameraProvider, ISensorProvider, IStorageProvider a INetworkProvider poskytuje přístup ke kameře, senzorům, poskytuje modulu uložit svá data do úložiště zařízení a navazovat síťová spojení nebo posílat předem definované dotazy na vzdálený server. V případě přístupu k úložišti zařízení a síťových spojení jádro poskytuje příslušné instance InputStream a OutputStream, které budou implementované pomocí návrhového vzoru proxy.

Návrhový vzor proxy spočívá v implementaci stejného rozhraní jako má původní třída a delegaci všech metod na implementaci původní třídy, delegovaná funkcionalita může být obohacena například o správu přístupu. V tomto případě je vzor vhodné využít pro sledování objemu přenesených dat pomocí síťového připojení a objemu uložených dat do úložiště zařízení.



Obrázek 3.5: Diagram tříd API pro lokalizační modul

Na diagramu, který je na obrázku 3.5, je zobrazen návrh tříd a rozhraní pro lokalizační modul. Největší rozdíl oproti trackovacímu modulu je, že lokalizační modul nemá možnost manuální správy kamery, musí se tedy spolehnout na rozhraní poskytnuté jádrem. Na druhou stranu nemusí poskytovat rozhraní kamery zpět jádru. Kromě této změny poskytuje jádro lokalizačnímu modulu identické rozhraní, jako modulu trackovacímu, včetně sledování vytížení systémových prostředků.

3.1.3 Sledování výkonu

Jelikož je počítáno s více dostupnými moduly zpracování obrazu a tyto moduly nebudou všechny mít stejné nároky na systémové prostředky, je třeba jejich spotřebu prostředků sledovat. Pokud pak modul využívá nadměrně prostředků, jádro může vybrat jiný, méně náročný modul, který bude nadále používat. Výměna modulů zpracování obrazu se bude dít hlavně v případě, že při inicializaci server neposkytl nejlepší konfiguraci pro aktuální zařízení a modul nebyl za běhu aplikace nastaven manuálně.

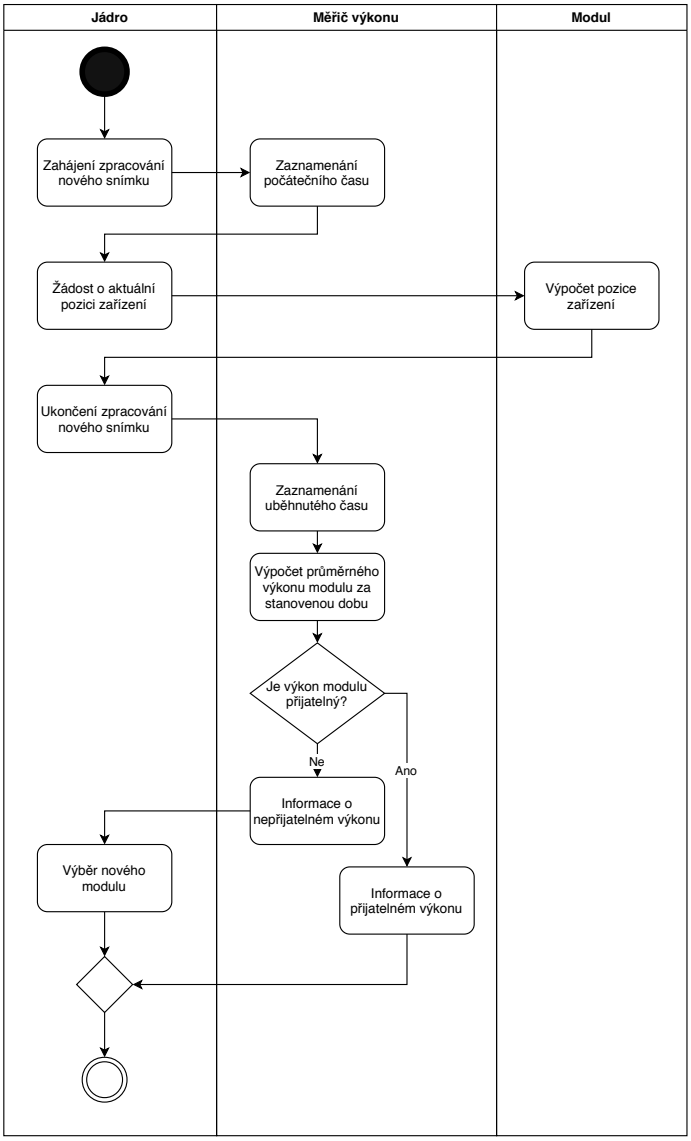
Na obrázku 3.6 je znázorněn proces měření výkonu modulu z hlediska zpracování snímku kamery za účelem určení nové pozice. Aby se samotné moduly nemusely zabývat začátkem a koncem měření a mohly se soustředit pouze na samotný výpočet, tyto úkony provádí jádro. Jádro informuje měřič výkonu o začátku a konci zpracování snímku. Měřič výkonu sbírá za běhu aplikace fixní počet vzorků výkonu formou pohyblivého okénka (například do kruhového pole). Z naměřených hodnot je následně počítán aritmetický průměr, aby byl snížen vliv délky zpracování jednoho složitějšího snímku mezi mnoha rychlými výpočty a byl lépe reprezentován celkový výkon modulu.

Kromě doby zpracování snímku jsou k určení výkonu a vhodnosti modulu použity i data o využití dalších systémových zdrojů, jako je úložiště, operační paměť a využití síťové komunikace. Jelikož pro využití úložiště zařízení a síťovou komunikaci poskytuje jádro modulům jednoduché rozhraní, opět aby se moduly mohly soustředit co možná nejvíce na samotné výpočty, je využití těchto prostředků snadné zaznamenávat pokaždé, když modul jedno z těchto rozhraní použije.

Aby mohl modul být nadále používán, musí jeho využití všech sledovaných prostředků být v přijatelných mezích. Pokud modul tyto meze překročí, dojde k vybrání nového modulu podle překročené meze tak, aby nově vybraný modul vytěžoval prostředek, který byl důvodem výměny, méně.

3.1.4 Komunikace se serverem

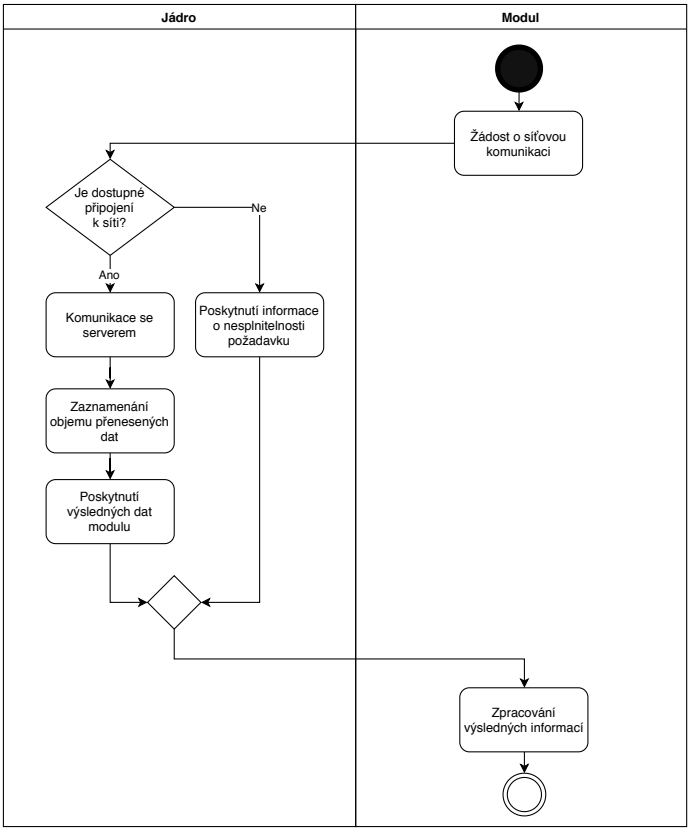
Pro komunikaci se vzdáleným serverem poskytne jádro modulům zjednodušené rozhraní. Veškeré vnitřní detaily komunikace budou pro moduly abstrahovány. Na obrázku 3.7 je znázorněn průběh zpracování dotazu modulu a následné vrácení výsledku. Tvůrce modulu se nemusí zabývat tvořením síťo-



Obrázek 3.6: Měření výkonu modulu

vého spojení ani použitým protokolem. Rozhraní poskytnuté modulům bude obsahovat všechny služby, které server aplikaci poskytuje.

Díky tomu, že veškerá síťová komunikace půjde skrz jádro aplikace, je snadné měřit využití sítě aktuálním modulem. Zároveň je, v případě potřeby, možné komunikaci omezit nebo filtrovat. Pokud není síťové připojení dostupné, je modul o tomto stavu informován.



Obrázek 3.7: Komunikace modulu se serverem

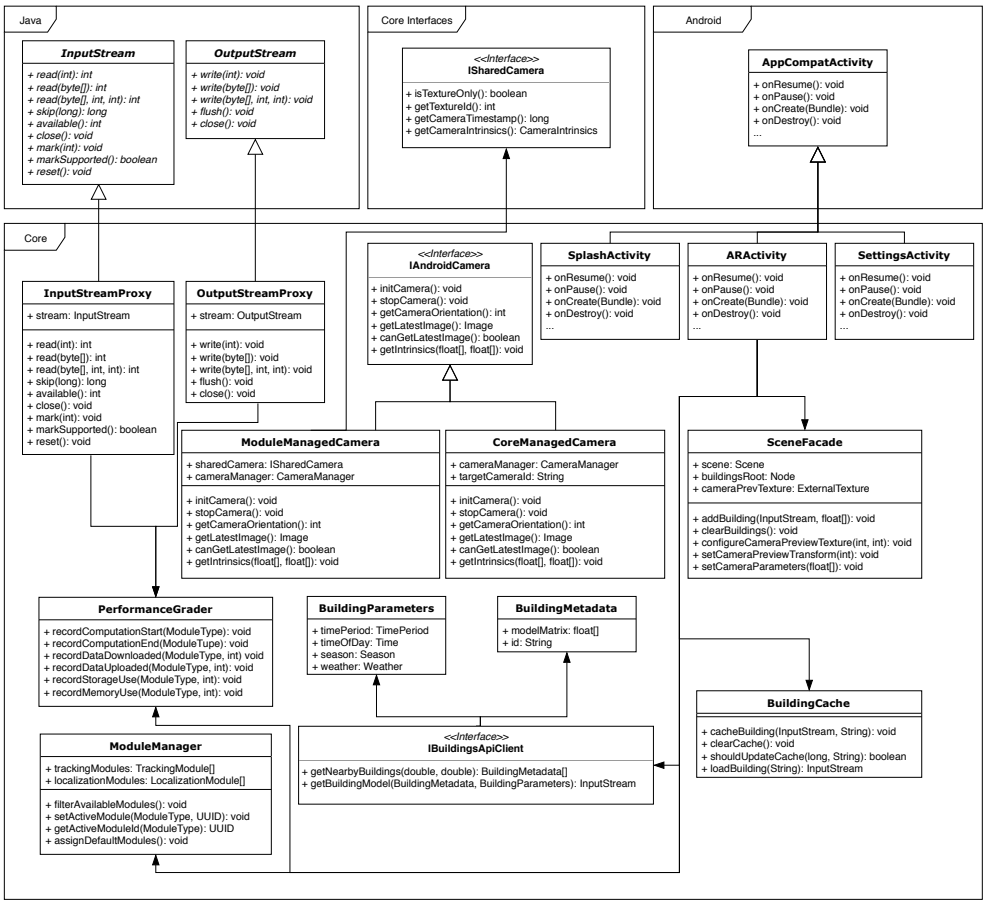
3.2 Třídy jádra

Samotné jádro obsahuje své implementace všech rozhraní, která bude poskytovat modulům. S vynecháním těchto rozhraní je na obrázku 3.8 zobrazen návrh důležitých tříd jádra.

Hlavní je zde třída `ARActivity`, která představuje Android aktivitu s prvky rozšířené reality. Android aktivita je v zásadě třída, jejíž instance bude vytvořena, když se aplikace dostane do té sekce, kterou daná aktivita představuje, například menu nastavení, nebo v tomto případě virtuální scény rozšířené reality. Zde se bude uživatel nejčastěji pohybovat. Třída aktivně využívá správce modulů a fasády scény.

Rozhraní `IBuildingsApiService` definuje metody, pomocí kterých bude jádro získávat metadata o budovách, které se vyskytují poblíž zadaných souřadnic. Pomocí těchto metadat a dalších navolených parametrů další metodou rozhraní získá samotný model budovy, respektive `InputStream`, který lze k nahrání modelu použít.

Třída s metadaty bude v budoucnu, spolu s dokončením celkového návrhu



Obrázek 3.8: Diagram hlavních tříd jádra

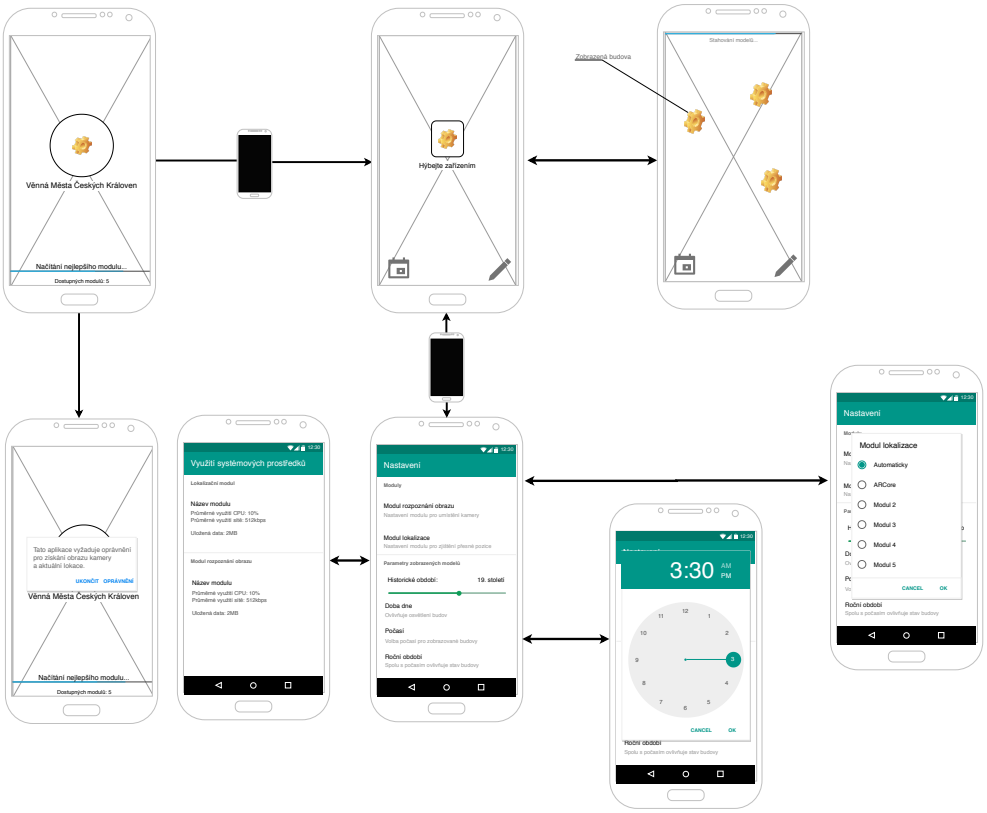
REST rozhraní vzdáleného serveru, upravena tak, aby její obsah reflektoval data vrácená serverem. Ze stejného důvodu bude pro účely prototypu vyhotovena dočasná implementace rozhraní bez komunikace se serverem, vracející předem připravená, statická data.

Ke zjednodušení interakce zbytku jádra s knihovnou Sceneform je navržena podle návrhového vzoru fasáda třída `SceneFacade`. Obsahuje jednoduché metody pro správu zobrazovaných modelů budov. Zároveň umožňuje snadné nastavení pozice a orientace virtuální kamery a přístup k textuře, do které by měl být kopírován obraz kamery zařízení, aby jej mohlo jádro zobrazit uživateli.

Třídy využívající vzoru proxy, `InputStreamProxy` a `OutputStreamProxy`, mají za úkol veškerou komunikaci nahlašovat třídě `PerformanceGrader`, která zaznamenává zdroje využité jednotlivými moduly tak, jak jí jsou nahlašovány proxy třídami a aktivitou. Na základě takto získaných informací se pak může rozhodnout zvolit jiný aktivní modul.

3.3 Uživatelské rozhraní

Uživatelské rozhraní aplikace je navrženo s ohledem na poznatky získané z analýzy podobných aplikací s prvky rozšířené reality. Jsou také brány v potaz Nielsenovy heuristiky a Google Augmented Reality Design Guidelines. Na obrázku 3.9 jsou zobrazeny navržené obrazovky a přechody mezi nimi. Přechody do a ze sekce rozšířené reality jsou zvýrazněny přechodem do černé, což je na grafu vyznačeno malou černou obrazovkou mezi přechody. Jednotlivým obrazovkám se věnují samostatné sekce.



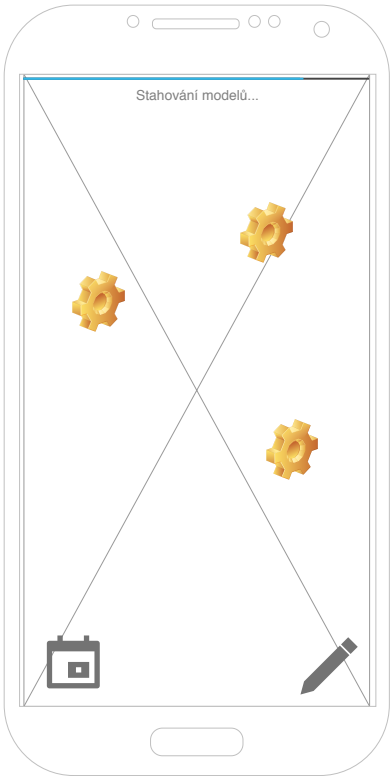
Obrázek 3.9: Graf obrazovek navrženého UI

3.3.1 Sekce rozšířené reality

Na obrazovce rozšířené reality bude typický uživatel trávit nejvíce času. Na pozadí bude vždy aktuální obraz kamery zařízení s ohledem na cílené zpoždění kvůli čekání na výpočet modulu rozpoznání obrazu. Navrch obrazu kamery zařízení jsou vykreslovány stažené modely budov. V horní části obrazovky je umístěn indikátor příběhu detekce a stahování modelů budov. Indikátor je

zároveň použitý za účelem poskytnutí informace o změně použitého modulu a aktuálních parametrech modelů budov.

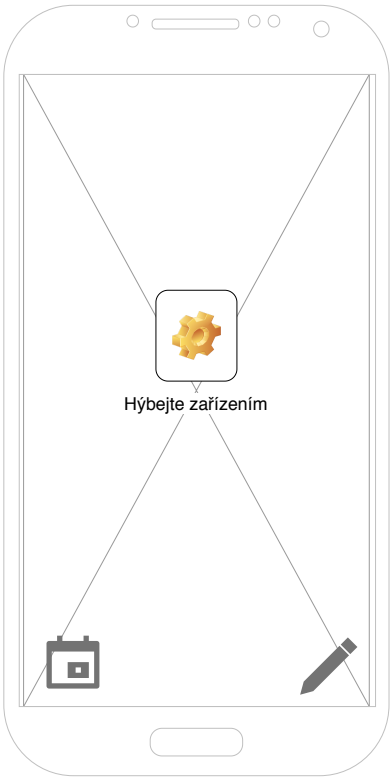
Jelikož může uživatel volně měnit parametry, podle kterých budou vybírány modely budov ke zobrazení, je také možné, že bude zvolena kombinace parametrů, pro kterou nebudou dostupné modely některých okolních budov. Zejména tato situace může nastat volbou historického období. Jestliže pro některé zobrazované budovy modely existují a pro jiné ne, bude uživateli poskytnuta informace o chybějících budovách prostřednictvím popisku v horní části obrazovky. Pokud nebudou žádné z okolních budov zobrazeny z důvodu nedostupných modelů pro zvolené parametry, popisek bude zobrazen výrazněji, aby přitáhl pozornost uživatele, spolu s informací o nutnosti změny parametrů.



Obrázek 3.10: Sekce UI rozšířené reality

V základním stavu jsou na obrazovce, jejíž návrh je na obrázku 3.10, rozšířené reality všechna tlačítka částečně průhledná, aby co nejméně přitahovala pozornost. Levé tlačítko slouží k rychlejšímu přístupu k nastavení historického období, které uživatele zajímá. Období je voleno posuvníkem z předem definované množiny hodnot. Pravé tlačítko umožňuje uživateli přejít na obrazovku nastavení.

Po startu aplikace, popřípadě po změně aktivního trackovacího modulu, může aktivní modul vyžadovat pohyb zařízením před tím, než skutečně začne poskytovat jádru zjištěnou pozici. Podobně také může být nutné některé senzory, které aplikace k práci potřebuje, kalibrovat pohybem zařízení. Na obrázku 3.11 je zobrazeno navržené rozložení prvků v případě, že je nutné, aby uživatel se zařízením hýbal, než bude aplikace schopna pracovat normálně. Ikona a text závisí na tom, jaká akce je po uživateli skutečně požadována.



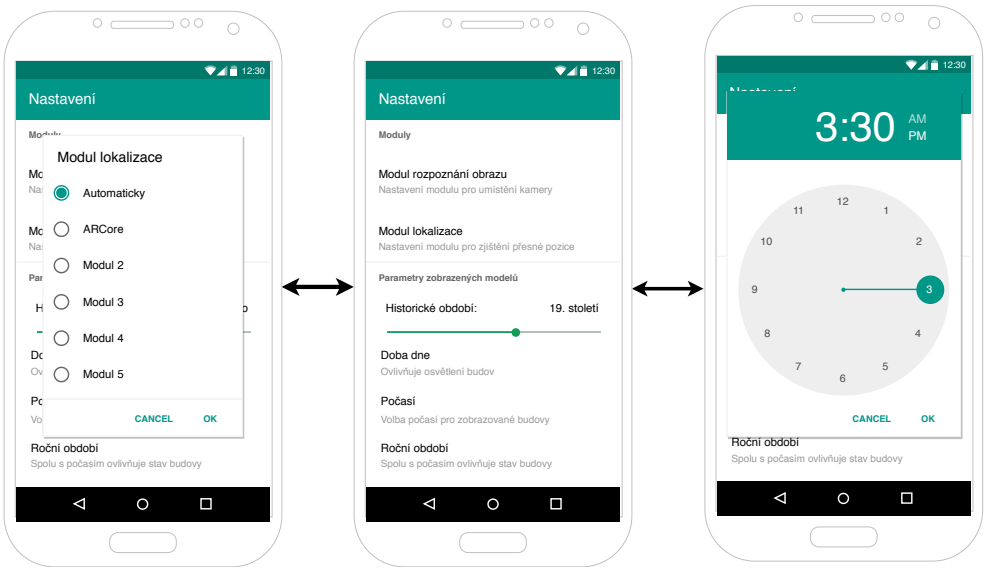
Obrázek 3.11: Sekce UI rozšířené reality s oznámením o nutném pohybu

3.3.2 Obrazovka nastavení

Za obvyklých okolností by měl normální uživatel většinu času trávit v sekci rozšířené reality, pokročilí uživatelé ale mohou chtít změnit například některé z parametrů zobrazovaných modelů. Na obrázku 3.12 je zobrazen návrh obrazovky nastavení, která pokročilým uživatelům poskytne další možnosti změny parametrů.

Pokročilý uživatel pak má možnost na obrazovce nastavení nastavit dvě důležité části aplikace:

- **Použité moduly** Uživatel má možnost změnit aplikací automaticky



Obrázek 3.12: Návrh obrazovky nastavení

vybraný modul lokalizace a modul zpracování obrazu. Uživatelská volba má před výběrem by základě výkonu prioritu. Zpět na automaticky zvolený modul uživatel může přepnout volbou "Automaticky".

- **Parametry zobrazovaných budov** Krom historického období budov, které si uživatel může zvolit i v sekci rozšířené reality, je zde možné zvolit další parametry, které jsou obvykle pro uživatele voleny automaticky na základě aktuálního času a počasí.

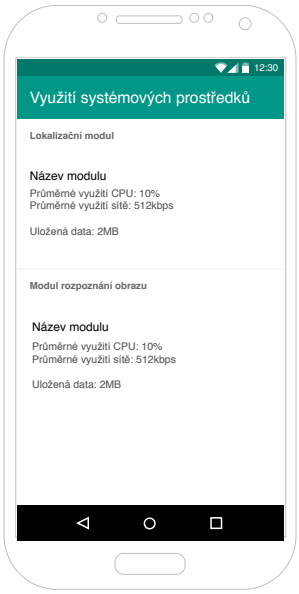
Pro volbu historického období zobrazovaných budov je použitý stejný ovládací prvek, jako v sekci rozšířené reality. Nastavení je zde znovu, aby jej uživatel snadno našel a nemusel přepínat na jinou obrazovku. Pro dobu dne je možné zvolit požadovaný čas, počasí a další možnosti je volba prováděna výběrem ze seznamu předem definovaných možností.

Vzhledem k tomu, že moduly jsou na sobě nezávislé, je možné vybrat jejich libovolnou kombinaci. Na rozdíl od modulů se výběrem jistých kombinací parametrů zobrazovaných budov může uživatel později dostat do stavu, kdy některé budovy nebudou mít k dispozici model se zvolenými parametry. Ačkoliv by se takový stav dal považovat za chybový, nelze mu na obrazovce nastavení zabránit. Je tedy nutné takový stav řešit až při stahování příslušných modelů a před jejich zobrazením uživatele o problému informovat.

3.3.3 Obrazovka výkonu modulů

Obrazovka, která zobrazuje aktuální měření výkonu právě běžících modulů je cílena hlavně na vývojáře nových modulů. Vývojář modulu zde může snadno

vidět, ve kterým směrem s ohledem na výkon by měl svůj modul dále zlepšovat.



Obrázek 3.13: Návrh obrazovky výkonu modulů

Na obrázku 3.13 je zobrazen návrh obrazovky s informacemi o výkonu. K této obrazovce se uživatel může dostat prostřednictvím obrazovky nastavení.

Pro každý z aktivních modulů jsou zde k dispozici informace o průměrném využití procesorového času, který modul využil pro každý dotaz na aktuální výsledek jádrem. Zároveň je zde zobrazen objem dat, který modul přenesl pomocí síťových připojení. Nakonec je zde informace o objemu dat, který má aktuálně modul uložený na zařízení uživatele.

Obrazovka neobsahuje využití paměti modulem z důvodu obtížnosti tohoto měření.

Kapitola 4

Realizace

Tato kapitola se zabývá změnami, které byly z různých důvodů během implementace vůči návrhu učiněny. Dále je popsán způsob, jakým je možné aplikaci pro systém Android automaticky nahrávat na platformu Google Play Store, odkud uživatelé dostávají automatické aktualizace aplikace. Nakonec je uveden způsob, jakým by rozhraní jádra mělo být rozšiřováno a měněno.

4.1 Změny implementace oproti návrhu

V některých případech se návrh projevil jako nevhodný a bylo nutné provést změny, popřípadě některá rozhraní či třídy přidat, zejména kvůli specifikům systému Android a použitých knihoven, která se projevila až během implementace.

4.1.1 Přístup k Android aktivitě a kontextu

Během implementace prototypu jádra a prvního modulu rozpoznání obrazu bylo třeba modulu předat objekt aktuálně běžící aktivity. Modul tento objekt potřeboval za účelem inicializace knihovny ARCore, kterou k výpočtu používá [7].

V původním návrhu nebyly pro moduly žádné části API Android přímo od jádra dostupné, jelikož bylo rozhraní navrhováno tak, aby byla implementace modulů od specifik systému Android co nejvíce oddělena. Aby ale bylo možné prototypový modul a jádro správně propojit, bylo do API jádra, které je pro moduly vystaveno, třeba přidat rozhraní `IAndroidProvider`, které modulům aktuálně poskytuje dva důležité objekty:

- **Aktuální Android aktivitu** Ve většině případů bude aktuální Android aktivita právě běžící instance `ARActivity`.

- **Aktuální Android kontext** Ačkoliv se ve většině případů může jednat o stejný objekt jako je aktuální aktivita, může se kontext od aktivity lišit. Z tohoto důvodu je rozhraním poskytnutý zvlášť.

Důsledkem těchto změn je fakt, že moduly mají snazší přístup k různým částem API systému Android.

Do rozhraní `IAndroidProvider` bylo navíc přidána možnost zjištění aktuálního natočení displeje zařízení. Na rozdíl od fyzického natočení zařízení jde pouze o rotace v násobcích 90 stupňů, jelikož displej zařízení má pouze polohu vodorovnou a svislou spolu se svými zrcadlovými variantami.

4.1.2 Změny dotazů na vzdálený server

Původní návrh rozhraní počítá s možností vzdáleného trackovacího modulu. Dotazy na tento modul by byly zprostředkovány jádrem. Nakonec však bylo kvůli vysokému využití sítě, které by takový modul způsobil, rozhodnuto s tímto způsobem zpracování zatím nepočítat. Příslušná metoda byla tedy odstraněna.

Jelikož v době návrhu rozhraní `INetworkProvider` nebyl k dispozici kompletní návrh REST rozhraní vzdáleného serveru, nemohl být návrh rozhraní úplný. Pro účely prototypu byla do rozhraní přidána jedna metoda REST rozhraní pro lokalizační modul.

4.1.3 Další změny rozhraní

Kromě již zmíněných větších modifikací rozhraní mezi jádrem a moduly bylo během vývoje prototypu učiněno několika dalších změn. Jedná se o změny ve způsobu předávání jednak obrazu kamery modulu, resp. jádru ze sdílené kamery modulu, dále pak o způsob předání informace o poloze a natočení kamery z modulu zpět jádru. Změny vypadají následovně:

- **ICameraProvider a ISharedCamera** - místo poskytnutí ID textury, kam obraz nahrávat, je poskytnutý Android Surface, na který má být obraz posílán.

- **Frame** - místo celé matice je pozice předána pomocí vektoru o třech složkách a natočení pomocí quaternionu.

Důležitou změnou je také přidání metod `onInit` a `onExit` do rozhraní lokalizačního modulu, které jsou jádrem volány ve stejných případech, jako stejnojmenné metody trackovacího modulu. Přidány byly za účelem umožnění lokalizačnímu modulu lepší správy prostředků.

Vzhledem k omezením použitého jazyka a aby bylo možné vyhnout se použití reflekce, která by mohla způsobit pomalejší inicializaci, byly statické metody nahrazeny metodami instančními. Zároveň tím vznikl požadavek, aby

každý modul obsahoval konstruktor bez parametrů, kterým jádro vytvoří objekt pouze za účelem dotazu např. na identifikátor modulu.

4.1.4 Změny v třídách jádra

Jednou z hlavních změn oproti původnímu návrhu tříd jádra a jejich rozhraní je opět použití třísložkového vektoru na určení pozice a quaternionu na určení natočení v rozhraní třídy `SceneFacade`.

Druhou hlavní změnou je nová třída `PauseNotifyingActivity`, která nahrazuje původní nadřazenou třídu třídy `ARActivity`. Umožňuje dalším částem jádra snadno zažádat o zavolání v případě, že je aktivita pozastavena, nebo znovu spuštěna. Takové chování je potřeba hlavně pro správnou funkčnost třídy `AndroidProvider`, jmenovitě poskytování informací o změně natočení displeje zařízení a velikosti zobrazovací plochy, na kterou aplikace vykresluje.

4.2 Použití CI pro automatickou aktualizaci Play Store aplikace

Tato sekce předpokládá použití Gitlab CI pro automatické sestavení a aktualizaci aplikace. Většinu kroků je třeba učinit i za použití jiného CI systému. U kroků specifických pro Gitlab CI bude tato skutečnost zmíněna.

4.2.1 Základní požadavky

- Google účet s oprávněním spravovat vydání aplikací pro příslušný Play Store účet
- Kontakt na vlastníka Play Store účtu (některé kroky musí učinit vlastník)
- Android aplikace vyvíjená pomocí Android Studia a Gradle build systému
- Použití Gitu pro verzování
- Instalace programu Docker (popsáno v pozdější sekci)

4.2.2 Příprava aplikace

Aby bylo možné aplikaci automaticky nasazovat na Google Play, je potřeba učinit kroky, které aplikaci na automatické nasazování připraví.

Verze aplikace

Možnému číslování verzí se věnuje pozdější kapitola, nicméně pokud již aplikace má zavedené značení verzí, je možné takové značení dále používat, důležité jen je, aby bylo možné verzi reprezentovat jedním celým číslem. Pokud aplikace zatím žádné číslování verzí nepoužívá, je vhodné zatím v souboru build.gradle hlavního modulu nastavit `versionCode 1`.

Nastavení podepisování .apk

Aby bylo .apk soubor možné nahrát na Google Play Store, musí být podepsán platným klíčem. Google aktuálně požaduje, aby byl klíč platný alespoň 25 let.

Tento klíč bude po celý život aplikace používán k podepisování vydání. Změna klíče, kterým je aplikace podepisována, prakticky znamená vytvoření nové aplikace na Google Play Store.

Tvorba podepisovacího klíče v Android Studiu

Pro klíč je nejprve třeba vytvořit úložiště (keystore). Zde bude klíč pod zvoleným aliasem uložen a při každém sestavení aplikace, které zahrnuje i její podepsání, je z úložiště klíč znovu načten. Úložiště klíče a samotný klíč je možné vytvořit skrz menu

```
Build -> Generate Signed Bundle(s) / APK(s)
```

Zde je třeba zvolit APK, v další (`Next`) části dialogu je možnost nabídky `Create new...` pod volbou cesty ke keystore.

Pro umístění keystore je důležité, aby nebylo součástí repozitáře, ve kterém jsou zdrojové soubory aplikace, zejména, pokud je či bude aplikace zcela otevřená. V zásadě tedy, pokud bude soubor keystore správně přidán mezi soubory ignorované systémem správy verzí, může být jako umístění zvolen adresář repozitáře. Alternativně je možné zvolit například nadřazenou složku. Platnost klíče musí být minimálně 25 let.

Po vytvoření keystore a klíče k podepisování aplikace máte čtyři důležité informace:

- Cestu, resp. soubor s keystore
- Heslo ke keystore
- Alias klíče k podepisování aplikace
- Heslo ke klíči

S vytvořeným klíčem je možné otevřený dialog pro tvorbu podepsaného sestavení aplikace zavřít, není nutné sestavení v tuto chvíli dokončit.

Tvorba podpisové konfigurace

Aby bylo možné podepisování provádět automaticky, musí být připravena konfigurace, která říká který keystore použít, který klíč, a jejich hesla. Tyto informace by samozřejmě neměly být uloženy v repozitáři s kódem, tomu se věnuje jedna z dalších sekcí.

V Android Studiu je třeba otevřít strukturu projektu:

```
File -> Project Structure...
```

Zde zvolit hlavní modul aplikace a záložku `Signing`. Na této záložce je možné přidat novou konfiguraci (s libovolným jménem), pro její nastavení slouží informace o keystore z konce předchozí sekce. Po vytvoření podpisové konfigurace je potřeba tuto konfiguraci zvolit na záložce `Build Types` pro typ sestavení, který bude nahráván na Google Play, obvykle "release".

Je možné ověřit funkčnost konfigurace spuštěním Gradle tasku signingReport. Gradle tasky, tedy dostupné příkazy pro systém sestavení Gradle, je v prostředi Android Studio možné nalézt v základním rozložení okna na pravé straně jako zavřenou záložku Gradle. Pokud proběhne bez problému, je konfigurace nastavena správně. Pokud je konfigurace v pořádku, je vhodné nechat pomocí Gradle tasku assembleRelease, případně skrze menu

```
Build -> Generate Signed Bundle(s) / APK(s)
```

vytvořit podepsaný .APK soubor pro prvotní nahrání na Google Play.

Oddělení keystore a hesel od repozitáře s kódem

Jelikož je předpokladem použití Gitlab CI, které v době psaní této práce nepodporuje tajné soubory, ale pouze tajné proměnné prostředí, všechny hesla a soubory budou sestavujícímu systému předány pomocí proměnných prostředí.

V souboru build.gradle hlavního modulu vydávané aplikace, ve kterém je podepisovací konfigurace aplikace, je potřeba hesla a jména klíče a keystore nahradit hodnotou proměnné prostředí libovolného jména. Příkladem je následující nastavení:

```
android {
  signingConfigs {
    release {
      // System.getenv() ziska promennou prostredi
      keyAlias System.getenv("ANDROID_KEY_ALIAS")
      keyPassword System.getenv("ANDROID_KEY_PASS")
      storeFile file('keystore.jks')
      storePassword System.getenv("ANDROID_KS_PASS")
    }
  }
  // ...
}
```

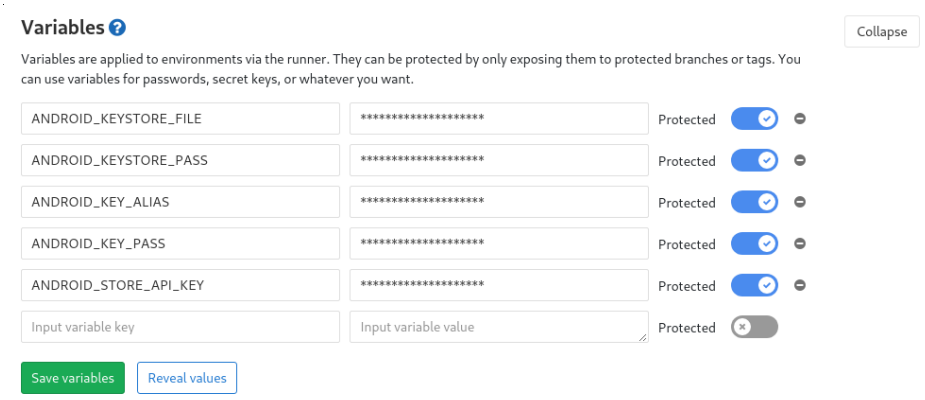
Po nahrazení hesel v podepisovací konfiguraci aplikace je nutné tato hesla včetně samotného souboru s podepisovacím klíčem nahrát jako tajné proměnné prostředí pro Gitlab CI. V uživatelském rozhraní softwaru Gitlab, v nastavení projektu `Settings -> CI/CD`, je sekce pro proměnné, které je možné použít během sestavení aplikace (`Variables`). Sem lze nahrát hesla výše nahrazená proměnnými prostředí pod stejným jménem. Soubor s podepisovacími klíči jde také nahrát, jen je potřeba jej převést na text, například nástrojem `base64`:

```
base64 < keystore.jks
```

Uvedené nastavení počítá s umístěním souboru keystore.jks do stejné složky, ve které je soubor build.gradle hlavního modulu aplikace. Soubor s instrukcemi pro Gitlab CI s tímto umístěním počítá, zároveň počítá se základním pojmenováním hlavního modulu aplikace. Obsahuje také instrukce, co změnit, pokud tomu tak není. Výsledný text je snadno nastavitelný jako proměnná prostředí, pro jejíž pojmenování je počítáno s názvem `ANDROID_KEYSTORE_FILE`, v sekci

```
Settings -> CI / CD -> Variables
```

Před použitím bude soubor samozřejmě potřeba převést zpět do původní podoby, což bude možné provést stejným nástrojem. Na obrázku 4.1 je zobrazena možná konečná konfigurace proměnných prostředí v systému Gitlab CI. Je zde již zahrnutý i klíč k servisnímu účtu Google Play Store, jehož tvorba je obsahem pozdější sekce.



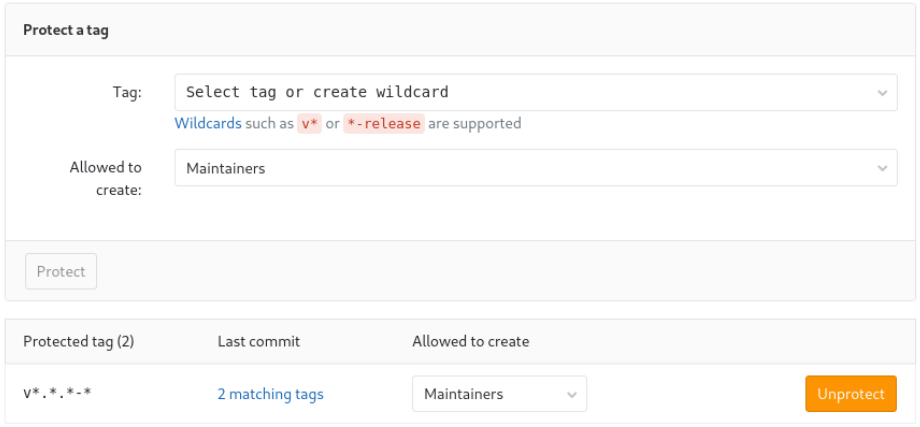
Obrázek 4.1: Nastavení proměnných prostředí v Gitlab CI

V případě použití chráněných (protected) proměnných je také třeba nastavit git tagy, které budou pro vydání používány, nastavit jako chráněné, aby měly k proměnným přístup. Z důvodu omezení systému Gitlab CI je možné chráněné tagy pomocí zástupných znaků (wildcard) přidat pouze tehdy, kdy

výslednému řetězci odpovídá alespoň jeden tag. K nastavení chráněných tagů je třeba se vrátit po nahrání prvního tagu k vydání. Chráněné tagy lze nastavit v sekci

`Settings -> Repository -> Protected Tags`

Na obrázku 4.2 je zobrazeno ukázkové nastavení chráněných git tagů, které mají přístup ke chráněným proměnným.



Obrázek 4.2: Nastavení chráněných tagů v Gitlab CI

Přidání modulu Gradle Play Publisher

Systém sestavení Gradle, který je pro sestavování aplikací pro platformu Android používaný, sám od sebe nepodporuje automatické nahrávání sestavení aplikace na Google Play. Tuto funkcionalitu je třeba přidat pomocí zásuvného modulu Gradle Play Publisher [21]. Na stránce modulu je uvedena aktuální poslední verze vhodná pro použití. Pro přidání modulu stačí do souboru `build.gradle` hlavního modulu aplikace přidat specifikaci použití pluginu. Umístěna musí být na začátku souboru před všemi dalšími specifikacemi, včetně dalších sekcí nastavení pluginů. Specifikace může vypadat následovně:

```
plugins {
    // ...
    id 'com.github.triplet.play' version '2.1.0'
}
```

Po přidání modulu je třeba poskytnout ve stejném souboru konfiguraci, která bude pro nahrávání na Google Play použita. Konfigurační blok, který musí být umístěn až za specifikacemi pluginů, může vypadat následovně.

```
play {
    serviceAccountCredentials file('play-api.json')
    track 'internal'
    releaseStatus 'completed'
}
```

Nastavení `serviceAccountCredentials` popisuje cestu k souboru s přihlašovacími údaji k servisnímu účtu, pomocí kterého bude sestavení na Google Play nahráváno. Tvorba servisního účtu a získání těchto údajů je popsáno v pozdější sekci. Nastavení `track` určuje typ vydání, pro každý typ dostupný na Google Play zde existuje možnost. V případě příkladu se jedná o interní verzi aplikace, která je dostupná uzavřené skupině pozvaných uživatelů. Nastavením možnosti `releaseStatus` na `'completed'` je přeskočeno postupné vydávání aktualizace pro zvětšující se procento uživatelů a aktualizace je vydána rovnou všem uživatelům. Více informací a další možná nastavení jsou dostupná na stránce modulu [21].

4.2.3 Tvorba nové aplikace na Google Play

Aby bylo aplikaci možné na Google Play nahrávat, je nejdříve nutné vytvořit v konzoli Google Play její záznam. Nový záznam aplikace je možné vytvořit tlačítkem `VYTVOŘIT APLIKACI` na stránce `Všechny aplikace`. Po vyplnění jazyka a názvu nové aplikace je záznam ve fázi konceptu.

Za účelem automatického vydávání aplikace je nutné aplikaci vydat, tedy alespoň pro uzavřenou skupinu uživatelů. V levé nabídce pro nově vytvořený koncept je několik sekcí, které mají vedle názvu nevyznačené zaškrtnuté pole. Tyto sekce musí být adekvátně vyplněny, aby bylo možné aplikaci vydat. Za zmínku také stojí, že sekci Hodnocení obsahu lze vyplnit pouze po nahrání prvního sestavení aplikace, což lze učinit v sekci Vydání aplikace. Během nahrání prvního vydání bude správce dotázán, zda chce aplikaci zapojit do programu podepisování aplikací na serverech firmy Google. V tomto kroku návod počítá s odmítnutím podepisování aplikace na serverech firmy Google, tomuto nastavení se bude věnovat pozdější sekce.

S každou dostatečně vyplněnou sekcí se u popisku sekce vyznačí zaškrtnuté pole. Po vyznačení všech požadovaných sekcí je možné učinit vydání aplikace, čímž bude jednak zpřístupněna zvolené skupině uživatelů, a také bude možné aplikaci na Google Play nahrávat automaticky.

Tvorba servisního účtu

Servisní účet může vytvořit pouze vlastník Play Store účtu. Instrukce v této sekci musí provádět právě vlastník, jiný účet do příslušné sekce nemá přístup.

Pro automatizovaný přístup ke Google Play Store vývojářskému účtu, resp. nastaveným aplikacím, a jejich úpravě, je nutné vytvořit servisní účet, který

bude mít přístup k API a příslušná oprávnění. Tvorbu servisního účtu lze začít v sekci

```
Nastavení -> Účet vývojáře -> Přístup k~rozhraní API
```

Jelikož se servisní účet samotný neváže na žádnou z nastavených aplikací, do zmíněné sekce se lze dostat z hlavní nabídky, do které je možné přejít volbou `Všechny aplikace`.

Po zvolení možnosti vytvoření nového servisního účtu bude zobrazen stručný návod. V prvním kroku návodu je odkaz, který směřuje na Google API konzoli, kde je nejprve třeba vytvořit nový servisní účet tlačítkem zobrazené nabídky `Create service account`. Po vyplnění jména účtu a popisu je v dalším kroku třeba přiřadit účtu roli. Role `Project -> Editor` je pro účely nahrávání aplikace na Google Play postačující.

V posledním kroku tvorby servisního účtu v Google API konzoli je důležité vytvořit soubor s klíčem, který bude později možné použít k autentizaci uvnitř kontejneru, který bude aplikaci sestavovat a nahrávat. Tlačítkem `Create Key` a následným zvolením formátu JSON dojde ke stažení souboru s nově generovaným klíčem. Celý stažený soubor bude při automatickém nahrávání aplikace muset být umístěn pod cestou, která byla v dřívější sekci nastavena modulu Gradle Play Publisher, například pod jménem `play-api.json` ve složce s hlavním modulem aplikace.

Je samozřejmě vhodné soubor s klíčem nenahrávat do repozitáře s aplikací. Místo toho je lepší jej nastavit jako proměnnou prostředí (v případě Gitlab CI) podobným způsobem jako soubor s podepisovacím klíčem v předchozí sekci. V případě klíče servisního účtu ale není nutné soubor převádět pomocí nástroje `base64`.

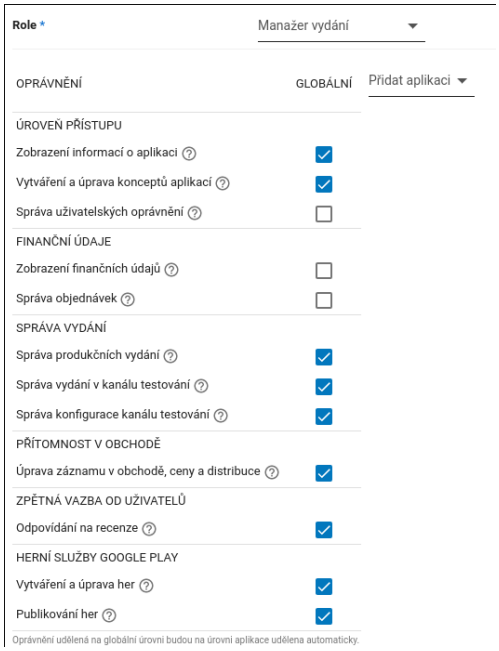
Po úspěšném vytvoření servisního účtu v Google API konzoli je nutné zavřít dialog s návodem v konzoli Google Play Store, načež se na stejné stránce obnoví seznam servisních účtů a načte se nově vytvořený účet. Novému účtu lze tlačítkem `Povolit přístup` přidělit práva, která potřebuje, aby bylo pomocí něj možné nahrávat nové verze aplikací. V případě potřeby lze servisnímu účtu později práva, jak je vidět na obrázku 4.3, změnit, popřípadě účet omezit jen na správu specifických aplikací, v menu

```
Nastavení -> Uživatelé a oprávnění
```

4.2.4 Využití Google Play pro podepisování aplikace

Google nabízí dvě možnosti pro podepisování. Jednou z možností je na Google Play nahrávat .apk soubory již podepsané vydávacím klíčem. Druhou je nahrát vydávací klíč přímo na Google Play a dále nahrávat už jen nový App Bundle, který bude na Google Play automaticky zabalen do .apk souboru a podepsán nahraným vydávacím klíčem.

Vydávací klíč lze nahrát do konzole Google Play v sekci



Obrázek 4.3: Nastavení oprávnění servisního účtu na Google Play konzoli

`Správa vydání -> Podepisování aplikací`

příslušné aplikace. Je zde i návod, nicméně pro úplnost jde ve stručnosti o následující:

V aplikaci Android Studio pod volbou

`Build -> Generate Signed Bundle(s) / APK(s)`

je nutné zvolit sestavení App Bundle a v další části formuláře zaškrtnout možnost exportu podepisovacího klíče. Po dokončení sestavení je v notifikaci o úspěchu odkaz krom hotového App Bundle také na exportovaný klíč, který lze nahrát na Google Play.

Pro zvýšení bezpečnosti je také vhodné vytvořit tzv. nahrávací klíč, který bude sloužit k ověření na straně Google Play, že je nové nahrané sestavení pravé. Za tímto účelem je třeba vytvořit nový podepisovací keystore stejně, jak je zmíněno v příslušné části přípravy stávající aplikace, a tento keystore nastavit jako podepisující. Po přípravě nového klíče musí klíč být vytvořen certifikát, který bude použitý ke kontrole vydání podepsaných nahrávacím klíčem, příkazem

```
$ keytool -export -rfc -keystore upload.jks -alias key0 \
          -file upload_cert.pem
```

kde `upload.jks` je soubor s klíčem, `key0` název klíče a soubor s názvem `upload_cert.pem` je výsledný certifikát, který lze nahrát do dodatečné sekce na stejné stránce, jako exportovaný klíč.

Nakonec je nutné nově vygenerovaný klíč, ze kterého byl tvořen certifikát pro kontrolu nahrávané verze, použít v podpisové konfiguraci, pomocí které bude tvořen výsledný nahrávaný balíček aplikace.

4.2.5 Příprava Gitlab CI

Systém Gitlab CI pro definici procesu sestavení a nahrání aplikace používá soubor `.gitlab-ci.yml` v kořenovém adresáři repozitáře aplikace (cestu k souboru lze popřípadě nastavit). V přílohách této práce je k dispozici ukázkový, komentovaný soubor `.gitlab-ci.yml`, který lze upravený použit pro sestavení a nahrání jednoduché Android aplikace na Google Play. Pro sestavení aplikace je použit Docker image obsahující veškeré potřebné nástroje. Soubory k sestavení použitého Docker image jsou také součástí příloh.

Lokální instalace platformy Docker

Ačkoliv sestavení aplikací mohou obecně běže i například na serveru samotném bez žádného oddělení, často se k těmto účelům využívá platformy Docker, která umožňuje sestavení izolovat od zbytku systému pomocí kontejnerů. Zároveň kontejner, ve kterém je sestavení prováděno, může být pro různé aplikace odlišný a splňovat specifické požadavky, jako jsou například verze knihoven nebo kompilátorů.

Aby bylo možné v pozdější sekci sestavit Docker image, který bude systémem Gitlab CI používán pro sestavení aplikace, je třeba mít lokální instalaci platformy Docker. Instalaci lze provést například instalací balíčku z oficiálních repozitářů distribuce, za předpokladu použití systému Linux. V případě systému Debian by příkaz pro instalaci mohl vypadat následovně:

```
$ sudo apt-get install docker.io
```

Pro jiné systémy a další distribuce systému Linux jsou instrukce pro instalaci k dispozici na oficiálních stránkách platformy Docker [22].

Android Docker image

Pro sestavení Android aplikace je vhodné připravit Docker image, který obsahuje pouze potřebné nástroje a součásti Android SDK. Přílohy obsahují adresář se soubory potřebnými pro sestavení tohoto image, spolu s komentovaným Dockerfile. Do souboru packages.txt je nutné napsat veškeré součásti Android SDK potřebné k sestavení aplikace. Aktuálně nainstalované součásti Android SDK je možné zjistit příkazem

```
$ANDROID_SDK_ROOT/tools/bin/sdkmanager --list
```

kde `$ANDROID_SDK_ROOT` je adresář s instalací Android SDK. Umístění instalace Android SDK je možné zjistit v prostředí Android Studio ve formě záznamu Android SDK Location v nabídce

```
Tools -> SDK Manager
```

Ve sloupečku Path výstupu příkazu jsou pak názvy balíčků, které je možné vkládat do souboru packages.txt. Důležité je také zkontrolovat shodu verze aplikace Gradle uvedené v použitém Dockerfile s verzí, která je aktuálně použitá v projektu, který bude tímto Docker image sestavován. Samotný Docker image lze po nastavení potřebných balíčků sestavit příkazem

```
docker build --tag cesta/nazev .
```

kde `cesta/nazev` je označení image, které bude musí být následně použito v souboru `.gitlab-ci.yml`. V případě použití gitlab.fit.cvut.cz musí cesta začínat `gitlab.fit.cvut.cz:5000/` a pokračovat cestou k repozitáři, který bude image používat, zakončeno názvem samotného image.

Po úspěšném sestavení image je pro umožnění jeho použití nutné jej nahrát do image repozitáře aplikace. Ujistit se, že repozitář aplikace má součást pro uložení Docker image zapnutou, je možné v sekci

```
Settings -> General -> Permissions -> Container Registry
```

Se zapnutou součástí pro uložení Docker image je nahrání sestaveného image možné přihlášením a následným nahráním image následujícími příkazy (za předpokladu použití serveru gitlab.fit.cvut.cz):

```
docker login gitlab.fit.cvut.cz:5000
docker push cesta/nazev
```

kde `cesta/nazev` je označení image použité pro sestavení.

Označení verzí aplikace

Za účelem opakovaného automatického nahrávání aplikace na Google Play, je nutné, aby každé nové sestavení mělo nový kód verze, který je číselně vyšší, než kód posledně nahraného sestavení. samotné označení verze (např. v22) může teoreticky být libovolné, jelikož tato verze se používá pouze k zobrazení pro uživatele aplikace, nicméně použitím skriptů v příloze této práce se verze omezí na označení pomocí sémantického verzování [23] s přidaným označením technického vydání za pomlčkou na konci verze. Výsledné označení může vypadat například `v1.2.8-5`.

Takto označované anotované tagy verzovacího systému (vytvořené příkazem `git tag -a NAZEV`) budou sloužit k označení verzí, které budou na Google Play vydány, jelikož není dobrým zvykem na Google Play nahrávat sestavení každého nového commitu.

Recept pro Gitlab CI

Součástí příloh práce jsou soubory `.gitlab-ci.yml` a `tag-to-vercode.sh`, které je možné přidat do kořenového adresáře repozitáře s aplikací. Soubor `.gitlab-ci.yml` obsahuje celý recept pro systém Gitlab CI, který se skládá z instrukcí, jak aplikaci sestavit, otestovat a následně, pokud se jedná o označené vydání, nahraje na Google Play. Kromě použitého obrazu Docker by nemělo být pro typickou Android aplikaci potřeba moc částí měnit. Recept již počítá s výše zmíněným oddělením hesel a podepisovacích klíčů od repozitáře s kódem a využívá příslušné proměnné prostředí, které by měly být na Gitlab CI nastaveny.

Pro získání verze z označeného vydání používá recept přiloženého skriptu `tag-to-vercode.sh`. Skript přijímá označení ve formátu `vXX.XX.XX-XX`, kde `XX` představuje jedno- až dvoumístné číslo. Skript následně v obsahu souboru `build.gradle`, který mu je ve formě cesty předán jako druhý argument, nastaví označení verze a zároveň nastaví kód verze tak, aby byla vyšší verze podle sémantického verzování označena vyšším kódem.

Jelikož je pro každé vydání na Google Play možné poskytnout poznámky o nové verzi, tyto poznámky recept vytvoří použitím textu, který je součástí označení verze (aby označení text obsahovalo, je nutné použít přepínač `-a` u příkazu `git tag`). Poznámky jsou později k dispozici uživatelům aplikace spolu s aktuální verzí.

Po přípravě receptu a pomocného skriptu do repozitáře aplikace je za předpokladu, že na Google Play již byla první verze ručně nahrána, možné označením commitu (pomocí příkazu `git tag -a vXX.XX.XX-XX` se zvoleným označením verze) a nahráním nového označení na Gitlab (např. příkazem `git push origin OZNACENI`) spustit sestavení a následné nahrání nové verze na Google Play. V závislosti na nastaveném typu vydání bude za nějaký čas nové vydání k dispozici ke stažení a stávající uživatelé dostanou o upozornění o aktualizaci aplikace.

4.3 Instalace a přiřazení specifického Gitlab runneru

Gitlab runner je server, na kterém je spouštěno sestavení aplikace prostřednictvím Gitlab CI. V případě serveru gitlab.fit.cvut.cz je obvykle k dispozici runner vytvořený administrátory a sdílený s ostatními projekty na serveru, které využívají Gitlab CI.

V případě, že sdílený runner není z libovolného důvodu dostačující nebo je potřeba provést sestavení na serveru, který žádný sdílený runner neposkytuje, je možné sestavovanému projektu přidělit vlastní, specifický Gitlab runner. Zakázáním sdíleného runneru v nastavení projektu je možné se sdílenému runneru zcela vyhnout a používat pouze runner specifický.

Tato část návodu předpokládá použití čistě nainstalované Linuxové distribuce Debian na platformě amd64 (resp. x86_64) a základní znalost práce s tímto operačním systémem. U všech příkazů, které vyžadují administrátorská oprávnění, je použitý program `sudo`.

4.3.1 Instalace platformy Docker

Instalace platformy Docker je možná přidáním příslušného repozitáře systémových balíčků a následnou instalací balíčků platformy. Instalace z repozitáře zároveň umožňuje jednoduché aktualizace. Jelikož oficiální repozitář platformy Docker nejsou součástí instalace distribuce Debian, je nutné repozitář přidat.

Před přidáním repozitáře je nutné nainstalovat balíčky, které umožní správci balíčků `apt` stahovat balíčky pomocí protokolu https. Lze tak provést následujícími příkazy:

```
$ sudo apt-get update
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    software-properties-common
```

Za účelem ověření autenticity balíčků využívá správce balíčků veřejného GPG klíče repozitáře. Pro oficiální repozitář platformy Docker je nutné klíč stáhnout a přidat příkazem:

```
$ curl -fsSL \
    https://download.docker.com/linux/debian/gpg \
    | sudo apt-key add -
```

Po stažení klíče je vhodné zkontrolovat, že byl správný klíč úspěšně přidán. Otisk klíče lze zkontrolovat příkazem `sudo apt-key fingerprint 0EBFCD88`. Posledním parametrem příkazu je posledních osm míst otisku. V době psaní této práce je otisk staženého klíče následující:

```
9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
```

Pokud byl stažen správný klíč s odpovídajícím otiskem, je možné přidat samotný repozitář. Existuje několik variant z hlediska stability, nicméně pro účely provozování Gitlab runneru je vhodné využít stabilní variantu. Repozitář lze přidat následujícím příkazem:

```
$ sudo add-apt-repository \
    "deb [arch=amd64] \
    https://download.docker.com/linux/debian \
    $(lsb_release -cs) \
    stable"
```

Vnitřní příkaz `lsb_release` slouží k zjištění verze systému Debian, pro kterou repozitář volit. Jakmile je repozitář úspěšně přidán do systému, platformu Docker lze nainstalovat příkazem:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce \
    docker-ce-cli containerd.io
```

Nyní je platforma Docker nainstalována. Příslušná služba, která umožňuje kontejnery vytvářet a spouštět, byla při instalaci spuštěna a bude spuštěna i po restartu systému. Varianty instrukcí pro jiné systémy jsou dostupné na stránce platformy Docker [22].

4.3.2 Instalace služby Gitlab runner

Podobně jako pro instalaci platformy Docker, i pro instalaci služby Gitlab runner je vhodné přidat do systému nový repozitář s oficiálním vydáním služby, přestože v oficiálních repozitářích systému Debian balíček `gitlab-runner` existuje. Pro výrazné zjednodušení procesu je poskytnutý skript, který lze stáhnout a spustit, jehož účelem je provést veškeré úkony potřebné pro přidání správného repozitáře. Skript je dostupný na následující adrese:

`https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh`

Před spuštěním skriptu je z bezpečnostních důvodů vhodné jej projít za účelem kontroly. Po spuštění poskytnutého skriptu je v systému již přidán nový repozitář obsahující balíček `gitlab-runner`. Aby bylo zajištěno, že se balíček bude vždy aktualizovat z oficiálního, nově přidaného repozitáře, je nutné připnout zdroj balíčku s dostatečnou prioritou. Lze tak provést následujícím příkazem:

```
$ cat <<EOF | sudo tee \
    /etc/apt/preferences.d/pin-gitlab-runner.pref
Explanation: Prefer GitLab provided packages
Package: gitlab-runner
Pin: origin packages.gitlab.com
Pin-Priority: 1001
EOF
```

Po zajištění zdroje instalace příslušného balíčku lze nejnovější verzi Gitlab runneru nainstalovat následujícími příkazy:

```
$ sudo apt-get update
$ sudo apt-get install gitlab-runner
```

Nově nainstalovaný Gitlab runner již zbývá jen konfigurovat k připojení ke správnému repozitáři. Po konfiguraci není již nutné runner nijak restartovat,

aktuální konfigurace bude načtena automaticky. Zároveň je při instalaci runner nastaven tak, aby byl spuštěn i po restartu systému. Varianty instrukcí pro jiné systémy jsou dostupné na stránce o instalaci služby Gitlab runner [24].

4.3.3 Přiřazení Gitlab runneru projektu

Jakmile je na serveru nainstalována jak platforma Docker, tak služba Gitlab runner, je možné server přidat jako specifický runner pro libovolný projekt. Server bude schopen zpracovávat libovolná sestavení spuštěná prostřednictvím Gitlab CI, která využívají platformy Docker.

Informace, které runner potřebuje, aby mohl být k projektu přidán, jsou dostupné v sekci

```
Settings -> CI / CD -> Runners
```

Jde hlavně o token, který zajistí spárování runneru s příslušným projektem. Druhá informace, kterou runner potřebuje, je adresa serveru, ke kterému by se měl připojit. Připojení runneru k serveru Gitlab lze začít následujícím příkazem:

```
$ gitlab-runner register --run-untagged
```

Parametr `run-untagged` slouží k zajištění, že runner bude moct zpracovávat sestavení bez tagů i v případě, že runner samotný je tagy označen. Po spuštění příkazu se program dotáže na adresu Gitlab serveru, ke kterému se má připojit, tedy například `https://gitlab.fit.cvut.cy`. Následně bude aplikace požadovat token, díky kterému se runner připojí ke správnému projektu.

Po zadání adresy a tokenu je třeba zvolit jméno runneru, dále. Při volbě způsobu, jakým bude runner své úlohy spouštět (docker, shell, atd.) je třeba vhodné zvolit možnost docker. Instalaci platformy Docker popisuje předchozí sekce.

Tím je runner připojen a připraven k použití projektem. Opět ve stejné sekci, kde byl zjištěn token pro připojení runneru, je možné zakázat použití sdílených runnerů pro účely projektu. Jakmile jsou sdílené runnery zakázány, bude k sestavení použitý pouze přihlášený specifický runner.

4.4 Rozšiřování možností a možných hodnot parametrů modelů

Ačkoliv prototyp obsahuje několik navržených parametrů modelů budov a hodnoty, kterých tyto parametry mohou nabývat, dá se předpokládat, že jak počet těchto parametrů, tak jejich možné hodnoty, by se v budoucnu mohly měnit.

V prototypu jsou všechny parametry modelů definovány jako preference, tedy uživatelské nastavení aplikace. Tím jsou nastavení parametrů zachována

napříč restarty aplikace systémem Android. Zároveň je tím usnadněna implementace jejich zobrazení a nastavení.

Samotné parametry jsou definovány v souboru, který je umístěn ve složce se zdrojovými kódy aplikace:

`app/res/xml/preferences.xml`

Soubor ve formátu XML definuje aktuálně veškerá nastavení, která aplikace obsahuje, rozdělené do kategorií. Jednou z těchto kategorií, pod příslušným uzlem `PreferenceCategory` jsou obsaženy možné parametry modelů. Každý parametr má svůj vlastní záznam `*Preference`, například v případě nastavení počasí se jedná o `ListPreference`. Každé nastavení má jako součást uzlu definován název klíče, pod kterým bude nastavení uloženo, a nadpis, který bude použitý při zobrazování uživatelského rozhraní na úpravu nastavení. Nadpis nastavení by měl, z důvodů lokalizace aplikace, odkazovat na lokalizovaný řetězec formátem `@string/`.

Možné hodnoty nastavení lze definovat staticky, jak tomu je u všech aktuálních nastavení parametrů modelů, jejich zmíněním přímo v definici nastavení. Jak pro nadpisy, tak pro samotné hodnoty, je možné se opět odkázat na řetězce, tentokrát na celou skupinu pomocí formátu `@array/`. V souboru se samotnými řetězci se pak takový seznam vyskytuje jako uzel `<array>`. Nadpisy by se opět měly vyskytovat v lokalizované verzi pro všechny podporované jazyky.

Je také možné nedefinovat možné hodnoty staticky, ale získat možné hodnoty například ze vzdáleného API. Za tímto účelem je třeba upravit třídu `MainSettingsFragment`, která v současné době řeší dynamické nastavení možných hodnot volitelných modulů. Podobným způsobem, jako jsou nastaveny možné moduly ke zvolení, lze také nastavit možné hodnoty ostatních nastavení. Pokud by však měly hodnoty být brány ze vzdáleného zdroje, je vhodné použít asynchronní operace a dočasně informovat o načítání uživatele, pokud se pokusí volbu provést. Zároveň je třeba dbát na lokalizaci nadpisů, pokud to nastavení vyžaduje.

4.5 Rozšiřování API jádra

API jádra, které je vystavené a použitelné moduly, spolu s třídami, které musí moduly implementovat, aby je mohlo jádro použít, jsou navrženy tak, aby bylo možné API jádra v budoucnu měnit jak se zpětnou kompatibilitou nových rozhraní vůči starým, tak bez zpětné kompatibility. Moduly mají možnost specifikovat, pro kterou verzi API jádra byly implementovány. Aktuální, první verze rozhraní jádra je obsažena v balíčku:

```
cz.cvut.fit.vmck.clientapp.core.api.v1
```

Poslední sekce názvu balíčku označuje verzi rozhraní. Novější verze rozhraní by tuto verzi měly postupně zvyšovat, zbytek názvu by měl ideálně být zachován. Za účelem dodržení zpětné kompatibility je možné nová rozhraní definovat jako rozšíření rozhraní starších, například novější verze rozhraní `ISensorProvider`, která poskytuje nový senzor nadmořské výšky, by mohl vypadat následovně:

```
package cz.cvut.fit.vmck.clientapp.core.api.v2;

public interface ISensorProvider
        extends cz.cvut.fit.vmck.clientapp.
        core.api.v1.ISensorProvider {
    float getAltitude();
}
```

Díky tomu, že nové rozhraní splňuje rozhraní předchozí, lze stále splnit požadavky starších modulů, které byly implementovány pro starší verzi jádra. Jediné rozhraní, které by se mezi verzemi nemělo měnit a jehož změna by pravděpodobně znamenala nemožnost použití starších modulů, je rozhraní `IVersionedModule`, které jádru umožňuje modul identifikovat a zjistit, pro kterou verzi jádra byl implementován.

Kapitola 5

Testování

Funkcionalita jádra byla podrobena několika druhům testů. Jednalo se o unit testy a manuální vývojářem prostřednictvím Google Play Store. Aplikace bude také podrobena uživatelskému testování. Kromě samotné aplikace byly uživatelsky testovány instrukce pro použití Gitlab CI za účelem automatické aktualizace Play Store aplikace.

5.1 Unit testy

Jednotkové (unit) testy mají za cíl ujistit se o správné funkčnosti samostatných částí kódu odděleně. Jedná se nejčastěji o jednotlivé metody klíčových tříd. Testovat se může nejen jejich správný výsledek, ale také jejich vedlejší efekty. Při vývoji aplikace pro systém Android lze využít nejen klasických unit testů (ve vývojovém prostředí Android Studio je pro tento účel k dispozici systém JUnit), ale také instrumentované unit testy, které jsou spouštěny buď na Android emulátoru, nebo na skutečném připojeném zařízení.

Kromě testovacího systému JUnit, který jsem měl k dispozici, jsem pro testování vedlejších efektů využil mockovací knihovnu Mockito. Knihovna Mockito umožňuje předem specifikovat, které metody mají na mockovaných třídách být volány a jaký mají vracet výsledek. Na základě toho lze pak kontrolovat, zda testovaná metoda proběhla správně a využila správných částí mockované třídy.

Pomocí unit testů byla kontrolována hlavně funkcionalita oznamování přenesených dat pomocí proxy tříd. Testovány byly také některé ze tříd implementujících rozhraní poskytnuté jádrem pro moduly. Instrumentovanými testy byla ověřena funkčnost implementací těch částí rozhraní, které ke své práci musí běžet na reálném nebo emulovaném zařízení, například pro poskytnutí informace o aktuálně běžící aktivitě.

5.2 Manuální testování

Během vývoje aplikace bylo sestavení aplikace za využití instrukcí pro využití Gitlab CI pro automatickou aktualizaci aplikace nahráváno na Google Play Store. Nahrané sestavení bylo k dispozici uzavřené skupině uživatelů, kteří měli možnost si aplikaci stáhnout do svého zařízení a nechat ji aktualizovat, když byla dostupná nová verze.

Do skupiny uživatelů, kteří měli k aplikaci přístup, patřili vývojáři aplikace spolu s vedoucím práce. Pomocí tohoto testování byly zjištěny a opraveny problémy s aplikací, které se nevyskytují na emulátoru, ale pouze na některých fyzických zařízeních. Například se jednalo o nevhodné nastavení vykreslování obrazu kamery, což způsobilo nechtěné grafické artefakty na výsledném obrazu.

5.3 Uživatelské testování

Uživatelské testování se zaměřuje na interakci uživatele s rozhraním softwaru. Testování má za úkol zjistit, zda je uživatelské rozhraní přívětivé a zda se snadno používá. Testování návodu, který je součástí práce, se zaměřuje na jeho snadnou aplikovatelnost a jednoduchost jeho použití. Uživatelské testy budou probíhat ve spolupráci s Usability laboratoří na Fakultě informačních technologií, ČVUT.

5.3.1 Testování uživatelského rozhraní aplikace

Testování výsledného prototypu aplikace Věnná města českých královen bude naplánováno na jeden den. Subjekty uživatelského testování budou další členové týmu celého projektu. Po vyplnění vstupního dotazníku budou umístěni do monitorované místnosti, kde bude na monitoru zobrazen testovací obraz, pomocí kterého aplikace zjistí aktuální umístění. Moderátor se subjekty projde scénář typického použití aplikace. Nakonec bude se subjekty vyplněn výstupní dotazník.

Veškeré materiály k uživatelskému testování jsou k dispozici jako přílohy práce. Zároveň jsou k dispozici na adrese `https://tinyurl.com/vmck2019`.

Testovací dotazníky

Pro uživatelské testování výsledného prototypu aplikace bylo nutné vytvořit vstupní a výstupní dotazníky. Oba dotazníky jsou tvořené tak, aby je bylo možné vyplnit elektronicky, nicméně jejich verze v přílohách práce jsou převedeny do formátu PDF.

Vstupní dotazník má za úkol zjistit vztah účastníka testování k aplikacím, které využívají prvky rozšířené reality, a jejich zkušenosti s takovými aplikacemi. Výstupní dotazníky se účastníka ptají na snadnost ovládání aplikace, splnění jejich očekávání vůči aplikaci a celkový dojem.

Výsledky testování

Z organizačních důvodů a po dohodě s vedoucím práce proběhne uživatelské testování až po odevzdání práce. Výsledky budou předloženy u obhajoby práce.

5.3.2 Testování postupu nastavení automatické aktualizace Play Store aplikace

Postup pro přípravu a nastavení projektu aplikace, založení příslušného záznamu na Google Play Store a následného nastavení Gitlab CI za účelem automatické aktualizace byl testován jedním z týmů předmětu BI-SI1. Jedním z úkolů týmu bylo proces automatické aktualizace nastavit a tím otestovat správnost a smysluplnost návodu.

Výsledkem byla řada nejasností a chybějících informací v návodu. Nejasnosti byly opraveny a chybějící informace doplněny. Hlavní změny za účelem řešení těchto problémů, kterými návod trpěl, byly následující:

- Lepší vysvětlení použití klíče k podepsání aplikace a jeho úložiště.
- Přidání informace o umístění nabídky vývojového prostředí pro spuštění úkonů systému sestavení Gradle.
- Přidání instrukcí ke zjištění umístění instalace Android SDK.
- Přidání sekce s instrukcemi pro instalaci platformy Docker.

Výsledný, již opravený návod je součástí kapitoly realizace této práce.

Závěr

Tato práce se zabývala implementací prototypu aplikace Věnná města českých královen, resp. jejího jádra. Cílem bylo analyzovat uživatelské rozhraní několika již existující aplikací, které v sobě obsahují prvky rozšířené reality. Zároveň byl součástí analýzy výběr knihovny, pomocí které lze snadno realizovat architektonický vzor dependency injection.

Na základě vybrané knihovny a vzoru dependency injection byly navrženy třídy jádra spolu s rozhraním pro moduly, které budou k jádru připojovány. Za použití poznatků z analýzy existujících aplikací bylo navrženo uživatelské rozhraní jádra, které bylo následně implementováno.

Aplikace byla během vývoje průběžně nahrávána na obchod Google Play Store. Nahrávání probíhalo automaticky využitím postupu, který je popsán v kapitole realizace. Výsledný návod je použitelný i pro další aplikace pro systém Android. Prototyp jádra byl podroben jednotkovým testům a spolu s moduly manuálním testům za použití verze nahrané na obchod Google Play Store.

Všechny body zadání byly splněny. Výsledný kód lze dále rozšířit a stavět na něm, nicméně zejména pro automatický výběr a přepínání modulů nebyla implementována veškerá navržená funkcionalita, převážně z časových důvodů. Některá funkcionalita jádra také vyžaduje hotový návrh rozhraní vzdáleného serveru a jeho alespoň částečnou implementaci, která v době implementace jádra nebyla k dispozici.

Do budoucna by bylo možné návrh jádra rozšířit o možnosti předzpracování obrazu použitého pro lokalizaci a trackování, pravděpodobně také formou modulů. Dále je třeba dokončit automatický výběr a výměnu modulů, kde existuje také možnost zlepšení způsobu měření výkonu modulů. Možné hodnoty parametrů modelů, které aplikace zobrazuje, by mohly být dynamicky získávány ze vzdáleného serveru a v aplikaci použity způsobem, který je popsaný v kapitole implementace.

Literatura

[1] Niantic, Inc.: Pokémon GO. [online], [cit. 2018-02-25]. Dostupné z: https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo&hl=en

[2] Analytical Graphics, Inc.: Satellite AR. [online], [cit. 2018-02-25]. Dostupné z: https://play.google.com/store/apps/details?id=com.agi.android.augmentedreality

[3] Terminal Eleven: SkyView® Free. [online], [cit. 2018-02-25]. Dostupné z: https://play.google.com/store/apps/details?id=com.t11.skyviewfree

[4] Southern Trans Atlantic Gaming: IKEA Place. [online], [cit. 2018-02-25]. Dostupné z: https://play.google.com/store/apps/details?id=com.STAG.Table_Top_AR

[5] Inter IKEA Systems B.V.: IKEA Place. [online], [cit. 2018-02-25]. Dostupné z: https://play.google.com/store/apps/details?id=com.inter_ikea.place

[6] A&E Television Networks Mobile: Knightfall^{TM} AR. [online], [cit. 2018-03-16]. Dostupné z: https://play.google.com/store/apps/details?id=com.aetn.games.android.history.knightfall.ar&hl=en_US

[7] ŠTĚPÁN, J.: *Věnná města českých královen – Modul rozpoznání obrazu.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 05 2019.

[8] Nielsen Norman Group: 10 Heuristics for User Interface Design. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics

[9] Google, Inc.: UI - Interaction - Augmented Reality Design Guidelines. Dostupné z: https://designguidelines.withgoogle.com/ar-design/interaction/ui.html

[10] Google, Inc.: UX - Interaction - Augmented Reality Design Guidelines. Dostupné z: https://designguidelines.withgoogle.com/ar-design/interaction/ux.html

[11] JENKOV, J.: Dependency Injection. [online], [cit. 2018-02-25]. Dostupné z: http://tutorials.jenkov.com/dependency-injection/index.html

[12] Google, Inc.: Guice. [online], [cit. 2018-03-08]. Dostupné z: https://github.com/google/guice

[13] KRASOV, A.: How Slow is Reflection in Android? [online], [cit. 2018-02-25]. Dostupné z: https://blog.nimbledroid.com/2016/02/23/slow-Android-reflection.html

[14] Google, Inc.: Dagger. [online], [cit. 2018-03-08]. Dostupné z: https://google.github.io/dagger/

[15] LIU, F.: Tiger. [online], [cit. 2018-03-08]. Dostupné z: https://github.com/google/tiger

[16] HERPAI, Z.: Feather. [online], [cit. 2018-03-08]. Dostupné z: https://github.com/zsoltherpai/feather

[17] Square, Inc.: Retrofit. [online], [cit. 2018-04-29]. Dostupné z: https://square.github.io/retrofit/

[18] Google, Inc.: Sceneform overview. [online], [cit. 2018-04-29]. Dostupné z: https://developers.google.com/ar/develop/java/sceneform/

[19] Tldrlegal.com: Apache License 2.0 (Apache-2.0) Explained in Plain English - TLDRLegal. [online], [cit. 2019-03-28]. Dostupné z: https://tldrlegal.com/license/apache-license-2.0-(apache-2.0)

[20] Tldrlegal.com: MIT License (Expat) Explained in Plain English - TLDRLegal. [online], [cit. 2019-03-28]. Dostupné z: https://tldrlegal.com/license/mit-license

[21] Triple-T: Gradle Play Publisher. [online], [cit. 2018-11-08]. Dostupné z: https://github.com/Triple-T/gradle-play-publisher

[22] Docker, Inc.: About Docker CE. [online], [cit. 2018-04-15]. Dostupné z: https://docs.docker.com/install/

[23] PRESTON-WERNER, T.: Semantic Versioning 2.0.0. [online], [cit. 2018-12-11]. Dostupné z: https://semver.org/

[24] GitLab, Inc.: Install GitLab Runner using the official GitLab repositories. [online], [cit. 2018-04-15]. Dostupné z: `https://docs.gitlab.com/runner/install/linux-repository.html`

Příloha A

Seznam použitých zkratek

API Application Programming Interface

AR Augmented Reality

CI Continuous Integration

CD Continuous Delivery

GPG GNU Privacy Guard

GPS Global Positioning System

REST Representational State Transfer

RPG Role Playing Game

UI User Interface

UML Unified Modeling Language

Příloha **B**

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace pro OS Android
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `gitlabci` ..... přílohy k sekci 3.1
- `testing` ..... materiály k uživatelskému testování
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF