Zadání bakalářské práce

Název: Věnná města českých královen - Mobilní klient
Student: Tomáš Kiec
Vedoucí: Ing. Jiří Chludil
Studijní program: Informatika
Obor / specializace: Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra: Katedra softwarového inženýrství
Platnost zadání: do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat mobilní aplikaci primárně pod OS Android pro vizualizaci virtuálních objektů (budov) v reálné scéně. Aplikace bude součástí projektu Věnná města českých královen.

1. Analyzujte existující aplikace Věnných měst českých královen.
2. Analyzujte možná řešení pro vývoj multiplatformních aplikací používající rozšířenou realitu.
3. Pomocí metod softwarového inženýrství navrhněte aplikaci s využitím modulárních lokačních modulů.
4. Implementujte prototyp aplikace, zaměřte se především na možnosti multiplatformního vývoje, udržovatelnost a uživatelsky přívětivé rozhraní.
5. Implementovaný prototyp podrobte vhodným testům (akceptační, integrační a uživatelské).

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 28. ledna 2022 v Praze.

Bakalářská práce

VĚNNÁ MĚSTA ČESKÝCH KRÁLOVEN - MOBILNÍ KLIENT

Tomáš Kiec

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Chludil
12. května 2022

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2022 Tomáš Kiec. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Kiec Tomáš. *Věnná města českých královen - Mobilní klient.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Obsah

Poděkování **vi**

Prohlášení **vii**

Abstrakt **viii**

Seznam zkratek **ix**

Úvod **1**

1 Cíle práce **3**

2 Analýza **5**
2.1 Věnná města českých královen 5
2.1.1 Rozšířená realita 5
2.2 Prototyp mobilní aplikace 6
2.2.1 Využité knihovny 6
2.2.2 Lokalizační a trackovací moduly 6
2.2.3 Uživatelské rozhraní 7
2.3 API pro získávání dat pro mobilní aplikaci 8
2.3.1 REST 8
2.3.2 Nedostatky API 8
2.4 Analýza použitých technologií 8
2.4.1 Výběr vizualizačního engine 9
2.4.2 Knihovna pro DI 11
2.4.3 Funkční a nefunkční požadavky 12

3 Návrh **13**
3.1 API klient 13
3.1.1 Části 13
3.1.2 Změny v API 16
3.2 Správa struktur 17
3.3 Lokalizačních moduly a jejich API 18
3.3.1 Ukotvení a kalibrace virtuální scény vůči scéně reálné 20
3.4 Uživatelské rozhraní 21

4 Implementace **23**
4.1 Implementační detaily 23
4.1.1 Nullable typy 23
4.1.2 Asynchronní programování a coroutines 24
4.2 Mapa a zobrazování HTML5 stránek 26
4.2.1 Javascript konvertor 27
4.3 Správa scén 30
4.4 Nasvícení budov 32

4.5 Uživatelská příručka . . . 32
4.6 Instalační příručka . . . 32
4.6.1 Instalace přiloženého souboru . . . 32
4.6.2 Instalace ze sestavení zdrojových kódů . . . 33
4.7 Programátorská příručka . . . 34
4.7.1 Debugging . . . 34
4.7.2 Struktura projektu . . . 34
4.7.3 Bundles . . . 35
4.7.4 Knihovna Runtime OBJ Importer . . . 35

5 Testování 37
5.1 Scénář testování . . . 37
5.2 Výsledky testování . . . 37

Závěr 39

Obsah přiloženého média 45

Seznam obrázků

3.1 Doménový model architektury aplikace 14
3.2 Diagram tříd API klient části aplikace 15
3.3 Diagram tříd správy modelů 18
3.4 Diagram tříd lokačních modulů a jejich API 19
3.5 Diagram aktivit nastavení ukotvení a kalibrace polohy 20
3.6 Wireframe uživatelského rozhraní 22

4.1 Diagram tříd Javascript konvertoru 28
4.2 Ukázka využití návrhového vzoru visitor 30
4.3 Diagram aktivit otevírání a zavírání scén 31
4.4 Snímky obrazovky aplikace 33
4.5 Důležité adresáře/soubory projektu 35

Seznam tabulek

2.1 Porovnání výběru technologií 11

Seznam výpisů kódu

1 Příklad JSON objektu transformací 17
2 Příklad JSON objektu ikony na mapě 17
3 LateInit metoda 23
4 Rozhraní IEnumerator 25
5 Příklad iterátoru 25
6 Příklad iterátoru s callback parametrem 26

Chtěl bych poděkovat svému vedoucímu, Ing. Jiřímu Chludilovi, za možnost spolupráce na tomto projektu, za pohotové reakce na mé dotazy a za skvělý přístup k vedení práce. Také bych chtěl poděkovat své rodině, přítelkyni a přátelům za psychickou podporu během studia a psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při 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. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, 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, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 12. května 2022 . . . .

Abstrakt

Tato bakalářská práce navazuje na prototyp aplikace rozšířené reality projektu *Věnná měst českých královen*, na kterém se FIT ČVUT podílí. Zabývá se analýzou současného návrhu a možností jeho využití v této práci, návrhem změn v jiných částech projektu VMČK, výběrem vhodných technologií pro vývoj nové aplikace a návrhem samotné aplikace vyvíjené ve zvoleném vizualizačním engine Unity. Součástí návrhu jsou přepracované lokační moduly, kterých může být aktivních více najednou a mohou se navzájem využívat ke zlepšení přesnosti výsledné lokace, rozhraní API klienta oddělené od detailů jak projektu VMČK, tak od detailů technického řešení API. Dále popisuje mé rozhodnutí v rámci vývoje zapnout si C# nullables a preferovat Unity coroutines a způsoby, jakými jsem vyřešil problémy, které jsem potkal v průběhu implementace, jako je konvertor objektů využívající prohledávání do šířky pro nalezení vhodného typu a správa scén, pro kterou jsem si implementoval řešení umožňující otevírat scény jak exkluzivně, tak aditivně. Na závěr je návrh uživatelského testování a návrhy pro budoucí rozvoj aplikace.

Klíčová slova mobilní aplikace, Android, Věnná města českých královen, rozšířená realita, rozšiřitelnost, přenositelnost, modulární architektura, vkládánjí závislostí, Unity, C#

Abstract

This bachelor thesis builds on the prototype of the augmented reality application which is part of the project *Dowry Towns of Bohemian Queens* of which the FIT CTU is part of. It deals with the analysis of the existing design and the possibilities of its use in this work, proposes changes in other parts of the *Dowry Towns of Bohemian Queens* project and the selection of appropriate technologies for the development of the new application, and proposes the design of the application itself developed in the selected visualization engine Unity. The design includes redesigned location modules that can be active several at once and can use each other to improve the accuracy of the resulting location, a client API interface separated from the details of both the *Dowry Towns of Bohemian Queens* project and the details of the technical API solution. It describes my decision in development to turn on C# nullables and prefer Unity coroutines, and the ways in which I solved problems I encountered during implementation, such as an object converter that uses a breadth-first search to find the appropriate type, and scene management for which I implemented a solution that allows me to open scenes both exclusively and additively. Finally, there is a proposal for user testing and suggestions for future development of the application.

Keywords mobile application, Android, Dowry Towns of Bohemian Queens, augmented reality, extensibility, portability, modular architecture, dependency injection, Unity, C#

Seznam zkratek

API	Application Programming Interface
AR	Augmented reality / rozšířená realita
CIL	Common Intermediate Language
DI	Dependency Injection / vkládání závislostí
ECS	Entity Component System
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IL2CPP	Intermediate Language To C++
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
OS	Operační Systém
REST	Representational State Transfer
TCP/IP	Transmission Control Protocol / Internet Protocol
UI	User Interface / uživatelské rozhraní
USB	Universal Serial Bus
VMČK	Věnná Města Českých Královen
XML	Extensible Markup Language

Úvod

Rozšířená realita (AR) je poměrně starý koncept, kde myšlenka rozšířit vnímání reálného světa se datuje již na počátek 20. století [1, str. 93], který se však masově rozšířil až v posledních pár letech, hlavně díky rozšíření výkonných (chytrých) mobilních telefonů. Nicméně i přes jeho masové rozšíření mnoho lidí nemá ani ponětí, že něco jako rozšířená realita existuje, nebo že dokonce tuto schopnost má i jejich mobilní telefon.

Dalším problémem rozšířené reality je dle mého názoru provedení, které sdílí většina aplikací využívajících AR – chybějící ukotvení virtuálních prvků do reálného světa. Pokud si otevřete téměř jakoukoliv aplikaci využívající AR, zjistíte, že fungují na principu „na aktuálním místě si vytvořím AR svět, a do blízkosti tohoto místa se umístí virtuální prvky". Jinými slovy, pokud potkáte v oblíbené hře Pokémon GO na pokémona, tak ho neuvidíte koupat se v místní kašně kde byl vidět na mapě v aplikaci, ale objeví se v rozšířené realitě před vámi, nehledě na směr kterým se díváte, a přesnější lokaci na které se nacházíte.

Tento problém ukotvení se snaží řešit již existující práce [2],[3], nicméně samotná aplikace [4] je pouze ve stádiu prototypu, a pouze na platformu OS Android. Z toho důvodu jsem si zvolil toto téma BP, protože bych rád v blízké budoucnosti viděl AR aplikace s dlouhodobě věrohodně ukotvenými virtuálními prvky v reálném světě.

Kapitola 1

Cíle práce

Cílem této práce je posunutí projektu VMČK dále – vytvoření mobilní aplikace pro OS Android, která však bude určena již koncovým uživatelům, bude dlouhodobě udržovatelná, jednoduše rozšířitelná v případě potřebných změn (například pokud by přibyla potřeba zobrazovat historicky oblečené osoby) a případně jednoduše přenositelná i na jiné platformy než je OS Android. Aplikace by také měla být uživatelsky přívětivá – neměla by být příliš komplikovaná.

V kapitole *Analýza* posoudím stávající řešení, rozhodnu se mezi úpravou stávajícího řešení nebo návrhem nového a případně zanalyzuji technologie pro návrh a implementaci nového řešení. Dále také budu analyzovat API [5], ze kterého se získávají modely historických budov.

Následně v kapitole *Návrh* navrhnu změny API pro potřeby aplikace a komunikaci s API, způsob správy modelů na scéně, zlepším způsob získávání lokace v aplikaci, navrhnu systém ukotvení a kalibrace ukotvení virtuální scény vůči scéně reálné a nakonec navrhnu uživatelské rozhraní aplikace.

Poté v kapitole *Implementace* popíši rozhodnutí které jsem udělal, problémy které jsem potkal při implementaci aplikace a způsoby, jakými jsem tyto problémy řešil.

Nakonec v kapitole *Testování* podrobím aplikaci uživatelskému testování a ověřím, že je uživatelsky přívětivá a není příliš komplikovaná.

Kapitola 2

Analýza

V této kapitole se zaměřím na analýzu prototypu mobilní aplikace [4] pro zařízení s operačním systémem Android vytvořené v práci Ondřeje Slabého, použitých technologií a modulární architektury modulů pro rozpoznání obrazu. Dále analyzuji backend, který je v aktuální podobě navržen a implementován Dominikem Sivákem [5], pro získávání historických budov a informací o nich, abych mohl navrhnout potřebné změny pro potřeby aplikace. Nakonec navrhnu funkční a nefunkční požadavky aplikace.

2.1 Věnná města českých královen

Projekt VMČK [6] – města věnována českými králi jejich (budoucím) manželkám jako svatební dar – má za cíl mapování a prezentaci historie těchto měst pomocí moderních technologií.

Projekt je velice rozsáhlý – historie věnných měst je zmapována, historické budovy vytvořeny pomocí nástrojů počítačové grafiky, konkrétně programu Blender, a na jejich textury automaticky aplikovány efekty přírodních vlivů (déšť, sníh atp.). Vytváření a implementacemi těchto modulů jako zásuvných modulů do programu Blender se věnovali Michal Zajíc a Jan Tislický ve svých bakalářských pracích [7],[8].

Výsledné modely budov jsou poté nahrány do backendu kde musí projít schvalovacím procesem, což zajistí jejich správnost a autenticitu. Backend tyto modely poté vystavuje pomocí svého REST rozhraní pro frontendy projektu VMČK. Tento backend byl v aktuální podobě navržený a implementovaný Dominikem Sivákem v jeho diplomové práci [5].

Mezi frontendy v projektu patří například webová aplikace pro schvalovací proces 3D modelů [9], webový vizualizátor 3D modelů [10], aplikace pro interakci s modely ve virtuální realitě [11] nebo mobilní aplikace pro zobrazení modelů v rozšířené realitě, kterou navrhoval a implementoval Ondřej Slabý ve své diplomové práci *Věnná města českých královen - Jádro mobilního klienta* [4], na kterou tato práce navazuje.

2.1.1 Rozšířená realita

Rozšířená realita, anglicky také často potkávaný název augmented reality nebo mixed reality, je umístění virtuálních objektů (modelů budov) do reálné scény. Augmented reality má ve slovníku [12] následující definici:

„A technology that superimposes a computer-generated image on a user's view of the real world, thus providing a composite view.“

Tato definice v překladu znamená „Technologie, která překrývá počítačem vygenerovaný obraz přes pohled na skutečný svět uživatele, (…)“.

Do této definice spadá široký rozsah zařízení, od brýlí Google Glass, zařízení (headsetu) Microsoft HoloLens, přes různé průhledové (head up) displeje v autech, až po chytré mobilní telefony podporující potřebná API jako je ARCore vyvíjený společností Google, nebo ARKit od společnosti Apple.

V případě rozšířené reality v mobilních telefonech se nejčastěji jedná o virtuální objekty ukotvené do reálné scény. To znamená, že pokud uživatel pohne zařízením, tak virtuální model zůstane na stejném místě vzhledem k reálné scéně. Toho využívá například známá hra Pokémon GO, díky které se můžete dívat na virtuálního pokémona, který pobíhá například po vašem obývacím pokoji. Další příklad je IKEA Place, který uživateli umožní umístit virtuální nábytek do místnosti aby se mohl podívat, jak by místnost vypadala.

Tato práce je zaměřena na nejvíce rozšířený způsob AR – pomocí mobilního telefonu.

2.2 Prototyp mobilní aplikace

Původní aplikace byla navržena a vyvíjena jako prototyp pro operační systém Android Ondřejem Slabým v jeho diplomové práci [4]. Při návrhu nebylo myšleno na budoucí přenositelnost na jiné platformy, což ovlivňuje výběr použitých technologií. Při návrhu byl kladen důraz na modularitu aplikace, nicméně i přesto je výsledná aplikace těžce závislá na ekosystému OS Android a programovacího jazyka Java.

Aplikace v aktuálním stavu poskytuje základní implementaci jádra a API pro moduly, avšak neobsahuje hotovou implementaci modulu využívajícího OpenCV, ani není napojená na REST API pro získávání dat budov.

2.2.1 Využité knihovny

Při návrhu aplikace bylo využito několika různých knihoven a technologií:

ARCore / Google Play Services for AR je platforma pro vývoj aplikací využívající rozšířenou realitu – virtuálních objektů umístěných do reálné scény – která poskytuje API jak v jazyce Java, tak v jazyce C pro OS Android a zároveň iOS, které jsou však od sebe odlišné.

SceneForm je knihovna pro vykreslování budov a obrazu kamery, uzpůsobená pro vývoj AR aplikací, která je postavena nad ARCore. Poskytuje API pro jazyk Java.

Dagger je knihovna pro dependency injection v jazyku Java – automatické vkládání závislostí při vytváření objektů.

Retrofit je knihovna pro deklarativní vytvoření REST API klienta – pomocí Java anotací metod vývojář nadeklaruje endpointy API, a knihovna sama vytvoří implementaci těchto metod.

Od návrhu původní aplikace se situace ohledně těchto knihoven změnila, konkrétně knihovna SceneForm je nyní deprecated [13]. Ač má dostupný zdrojový kód, tak by bylo třeba knihovnu udržovat, což by přidalo nemálo práce při udržování samotné aplikace. Alternativně lze využít jednoho z nástupců vyvíjeného komunitou [14],[15].

Knihovna Dagger již není doporučována pro DI na platformě Android, místo ní je doporučena její nadstavba Hilt [16].

2.2.2 Lokalizační a trackovací moduly

V návrhu aplikace se počítá se dvěma moduly: lokalizační a trackovací. Lokalizační modul zajišťuje zjištění souřadnic mobilního zařízení v reálném světě. Trackovací modul zajišťuje ukotvení virtuálních budov do reálné scény. Jádro aplikace modulům zajišťuje několik rozhraní, které mohou moduly využívat pro svoji funkčnost. `INetworkProvider` umožňuje komunikaci modulu se

vzdáleným serverem, IStorageProvider umožňuje modulu ukládat data do lokálního úložiště, ICameraProvider poskytuje modulu přístup k datům z kamery a ISensorProvider poskytuje modulu přístup k datům ze senzorů jako je například lokace (nejčastěji z GPS), orientace, …[4, 31]

Tento přístup umožňuje jednoduché odstínění modulů od specifik operačního systému, ale kvůli rozhraní ISensorProvider zavádí závislost aplikace na senzory, které nemusí být vždy dostupné a potřebné k funkčnosti aplikace a modulů. Dále se musel Ondřej Slabý kvůli různým komplikacím uchýlit k vytvoření nového rozhraní, IAndroidProvider, a změně ICameraProvider, kde namísto identifikátoru textury do které má modul zapisovat obraz je dostupná instance třídy AndroidSurface. Těmito kroky i přes jeho snahu vznikla závislost modulů na specifikách operačního systému Android. Současný návrh rozhraní také modulu poskytuje pouze jedinou možnost jak získat lokaci zařízení, a to pomocí lokace v ISensorProvider. To vytváří explicitní závislost na lokačním hardware jako je například kompas v zařízení v případě potřeby například pouze GPS, nebo naopak. Alternativa je nezáviset na tomto rozhraní, důsledkem toho je ale nemožnost využít žádného senzoru.

V samotné implementaci aplikace se počítá s pouze jedním lokalizačním a jedním trackovacím modulem. Aktuálně neexistuje způsob (bez využití systémových API), jak získat lokaci zařízení jinak, z například jiného lokačního modulu, než z rozhraní ISensorProvider. Kvůli tomu je třeba mít buď explicitní závislost na ISensorProvider v lokalizačním modulu využívající obraz z kamery, nebo rozhraní ISensorProvider nevyužívat.

V první případě je vytvořena zbytečná závislost na ISensorProvider, což by mohl být samotný lokalizační modul. Tímto přístupem se také přijde o celý lokalizační modul v případě, že není k dispozici lokační hardware v zařízení, ale existují jiné informace o poloze zařízení z jiného modulu (například se jedná o zařízení fyzicky umístěné v jedné místnosti, a existuje modul který tuto lokaci vrací. V takovém případě je známá lokace s přesností na maximálně několik desítek metrů, ale i přes to nelze modul využít).

V druhém případě by byla pozice zařízení získána pouze v případě, že se uživatel dívá na krajinu pro existují data. To také zvětší počet fotografií potřebných k lokalizaci, což se rychle stává nepraktické jelikož tyto data jsou většinou stahována na mobilním datovém připojení.

Trackovací modul realizoval Jaroslav Štěpán ve své diplomové práci [2] pomocí ARCore, a lokalizační modul Pavel Kříž ve své diplomové práci [3]. V implementaci bylo využito přístupu s explicitní závislostí na rozhraní ISensorProvider poskytované jádrem aplikace.

Pro trackovací modul se Ondřej Slabý a Jaroslav Štěpán rozhodli využít ARCore – dle popisu na Google Play přejmenován na „Google Play Services for AR“, dle dokumentace pořád ARCore. Kvůli tomuto návrhu musela být nevyužita implementace využívající ARCore v knihovně Sceneform, a nahrazena vlastní, která umožňuje sdílení kamery s lokalizačním modulem.

Lokalizační modul realizoval Pavel Kříž v jazyce C++ za pomoci knihovny OpenCV.

2.2.3 Uživatelské rozhraní

Aktuální prototyp aplikace má uživatelské rozhraní uzpůsobeno především pro potřeby vývojářů. Obsahuje vývojářské nastavení, a mnoho zbytečných funkcionalit, jako například manuální volbu lokalizačního a trackovacího modulu, nebo počasí a denní doby využité při volbě modelu.

Návrhu nové verze uživatelského rozhraní se věnoval Lukáš Brhlík ve své bakalářské práci [17]. Osobně si myslím, že v ní však při návrhu přehlédl náročnost implementace (například navigace na mapě k lokaci budovy), a zároveň mi přijde návrh uživatelského rozhraní zbytečně komplikovaný. Dle mého subjektivního názoru, pokud bych poprvé otevřel aplikaci a první co bych uviděl by byla přihlašovací obrazovka, tak aplikaci ihned vypnu a odinstaluji.

2.3 API pro získávání dat pro mobilní aplikaci

Jádro celého projektu věnných měst – systém pro správu a poskytování historických modelů a informací o nich – jehož současnou podobu navrhoval a implementoval Daniel Vančura technologií REST ve své bakalářské práci [18]. API je zdokumentováno v aplikaci Swaggerhub.

2.3.1 REST

REST je architektonický styl návrhu a vývoje API dostupných přes webové rozhraní. REST API vystavuje jednotlivé entity (objekty) – v tomto případě modely budov, textury, ...-- jako URI HTTP serveru. Při zavolání vhodné HTTP metody (GET, POST, PUT a DELETE) na danou URI server provede vyžádanou operaci, a případně vrátí vyžádaný objekt ve formátu jako je například JSON nebo XML. REST je bezstavový, což znamená že si server neudržuje žádné informace o sezení klienta, což vede k většímu výkonu. Na druhou stranu bezstavovost komplikuje situace kdy je třeba zachovávat kontext mezi požadavky, například přihlášení uživatele.

2.3.2 Nedostatky API

API má aktuálně několik nedostatků, které brání jeho využití v mobilní aplikaci.

Vyhledávání dle pozice zařízení a roku není aktuálně implementované. Vyhledávání budov, kdy uživatel pošle dotaz se souřadnicemi a očekává budovy ve své blízkosti je zdokumentované a navržené ve `/structures`, ale neexistuje implementace. To samé vyhledávání dle roku kdy budova stála. API aktuálně vrací všechny budovy.

Automatické dosazení parametrů jako je aktuální počasí, lokální čas a pouze historikem schválené budovy díky čemuž se automaticky zvolí vhodný model budovy, také není ve `/structures` implementované.

Transformační matice modelu ve `/structures` je v dokumentaci, i v aktuálně dostupných datech, 3×3. Jelikož se jedná o velikost, rotaci a pozici ve třírozměrném prostoru, tak matice musí být 4×4.

Textury ve `/structures` jsou aktuálně vraceny jako pole objektů, kde každý objekt má identifikátor, název souboru, datum nahrání na server a referenci na lokaci samotné textury. Nelze však poznat, o jaký typ textury se jedná (normálová mapa, specular mapa, ...).

Popis a název budovy v objektu ve `/structures` je aktuálně pouze jako položka description/name, kde je uložen řetězec s popisem/názvem. To je nedostačující, nelze takto udělat složitější popisy s například obrázky, a popis a název nelze lokalizovat.

Mapa navrhovaná v bakalářské práci [17] Lukáše Brhlíka vyžaduje pro svoji funkcionalitu název, popis a pozici všech budov. Toho lze aktuálně dosáhnout také díky `/structures`, ale spolu s těmito informacemi API vrací i nemálo dalších. To je v této situaci zbytečně mnoho dat jako je například status schválení historikem, což představuje nejen zbytečnou zátěž na přenesené data, ale také na procesorový čas co mobilní telefon bude muset zpracovat příchozí soubor. (a tím pádem rychlejší vybití baterie mobilního telefonu)

2.4 Analýza použitých technologií

Jelikož použité technologie v prototypu aplikace jsou vysoce závislé na platformě JVM a Android, nelze na prototyp navázat, ale musím si zvolit jiné.

2.4.1 Výběr vizualizačního engine

Z požadavku na lehkou přenositelnost a ukončeného vývoje knihovny Sceneform plyne požadavek na novou vizualizační knihovnu/engine. Hlavní požadavek je podpora rozšířené reality, nejlépe ARCore, který vybral Jaroslav Štěpán ve své diplomové práci jako nejlepší technologii z jím analyzovaných. Další důležitý požadavek je na přenositelnost nejen na aktuálně nejpoužívanější mobilní platformy Android a iOS, ale na případné budoucí. Do výběru zahrnu pouze knihovny/engine, které jsou pro nekomerční využití bez poplatku.

Výběr budu provádět na základě následujících kritérií: přenositelnost na jiné platformy, programovací jazyk(y), kvalita dokumentace, kvalita komunity (dostupnost různých ukázkových projektů, návodů, otázek a odpovědí na různých fórech, …), jak obtížné je začít v dané technologii pracovat (pro člověka který nikdy předtím danou technologii neviděl. Zde bude nazváno „getting into“), a otevřenosti.

Každé kritérium ohodnotím na stupnici 0--9, kde 0 znamená nejhorší hodnocení a 9 nejlepší hodnocení. Do hodnocení bude promítnut můj subjektivní názor, protože považuji za důležité, aby se mně osobně ve vybrané technologii dobře pracovalo.

2.4.1.1 Pokračování aktuální implementace

Aktuální implementaci byla analyzována v 2.2, kde bylo zjištěna neoddělitelná závislost na OS Android, kvůli čemuž udělím za přenositelnost 0.

Programovací jazyk je určen projektem na Java (staticky typovaný jazyk kompilovaný do bajtkódu), Android umožňuje také použití programovacího jazyku Kotlin (též staticky typovaný jazyk kompilovaný do bajtkódu), který nabízí interoperabilitu s jazykem Java a opravuje nemálo špatných rozhodnutí udělaných při návrhu jazyku Java. Za programovací jazyky udělím 7.

Dokumentace k technologiím které projekt využívá je dostatečná, dokumentace Sceneform by však mohla být dle mého názoru lepší. Nicméně samotná implementace projektu je skoro nezdokumentována a orientaci v něm považuji za relativně obtížnou, proto za dokumentaci udělím 4.

Kvalita komunity technologií využitých v projektu je dobrá, kromě Sceneform. Proto za komunitu udělím 5.

Samotná implementace mi přijde chaotická. Obsahuje metody u kterých závisí na pořadí jejich volání v místech kde by nemělo a v projektu je nemálo návrhových antivzorů (například aktivit místo fragmentů, nebo rozsáhlých metod), což ztěžuje iniciální pochopení projektu. Za getting into kvůli tomu udělím 2.

Původní projekt je šířen pod licencí MIT, proto mu za otevřenost udělím 9.

2.4.1.2 Vlastní engine

Naprogramování vlastního 3D engine by umožnilo přenositelnost na libovolnou platformu. To by ale vyžadovalo nesmírné úsilí, čas a znalosti 3D grafiky, což není v rámci možností této práce, proto tuto možnost v tomto výběru nezahrnu.

Pro rozšířenou realitu by však šlo využít knihoven ARCore pro Android nebo iOS a ARKit pro iOS. I přesto, že ARCore má API pro iOS, tak se liší od API pro Android, což by stejně vedlo ke dvěma implementacím i při volbě ARCore pro iOS.

2.4.1.3 Unreal Engine

Unreal Engine je profesionální 3D engine pro vývoj grafických aplikací a her od Epic Games. Poskytuje jednotné API pro vývoj AR aplikací na různých platformách, což zjednodušuje vývoj, a v případě nové platformy není třeba implementovat další API. Na platformách mobilních telefonů využívá ARCore na OS Android a ARKit na iOS. Unreal Engine má také dostupný

zdrojový kód, takže je možné engine upravit tak, aby bylo možné aplikaci přenést na jinou platformu než Unreal Engine podporuje.

Je dostupná široká škála zařízení a ani ta není limit, ale přenést celý engine na jinou platformu by bylo nesmírně náročné. Přenositelnost na jiné platformy díky tomu ohodnotím 8.

Pro programování aplikací v Unreal Engime lze využít C++, což je nízkoúrovňový, staticky typovaný jazyk překládaný do strojového kódu. Sice dosahuje dobrého výkonu, ale to je pro požadované využití relativně zbytečné. Druhá možnost je Python, což je vysokoúrovňový, dynamicky typovaný, interpretovaný / překládaný jazyk. Dále je možné „programovat" ve vizuálních jazycích, těm zde ale nebudu dále věnovat pozornost. Skóre pro programovací jazyky zde udělím 5.

Kvalita dokumentace je dobrá, ale je označena jako rozpracovaná. Také v ní chybí ukázky použití daného kódu. [19] Proto ji ohodnotím 7.

Komunita ohodnotím 4, protože jsem našel jen málo referencí k tvorbě AR aplikací. Unreal Engine má i vlastní fórum kde odpovídají i zaměstnanci Epic Games, ale nepřijde mi moc aktivní.

Getting into ohodnotím 6, protože Unreal Engine má kvalitní dokumentaci, ale zaměření na profesionální využití je znatelné. Aktuální verze Unreal Engine (v době psaní práce 5.0.1) také vyžaduje 59,09 GB místa na disku, což může být také překážka.

Unreal Engine má k dispozici zdrojový kód, který ale nelze redistribuovat. Lze do něj ale nahlížet a modifikovat ho, díky čemuž otevřenost ohodnotím 7.

2.4.1.4 Unity

Unity je také profesionální 3D engine pro vývoj grafických aplikací a her, ale je také cílen především na začátečníky. Díky AR Foundation podporuje rozšířenou realitu a poskytuje jednotné API také na platformách mobilních telefonů na OS Android a iOS.

Přenositelnost ohodnotím 7 kvůli již zmíněnému API a široké škále podporovaných zařízení.

Programuje se v C#, což je staticky typovaný jazyk, překládaný do CIL, nebo do strojového kódu technologií IL2CPP. Jeho nevýhodou je podpora pouze starší verze C# a .NET frameworku, kvůli čemuž nelze využívat některé nové funkce, například abstraktní statické metody v rozhraních, nebo Span<T>. Kvůli tomu programovací jazyk ohodnotím 8.

Unity má kvalitní dokumentaci s popsanými metodami, properties atd., spolu s vysvětlenými ukázkami kódu. Pokud se jedná o .NET API, tak Microsoft dokumentace je na podobné úrovni, s rozsáhlými okomentovanými ukázkami kódu. Velice dobře se mi v obou dokumentacích naviguje a zjišťuje potřebné věci, proto za dokumentaci udělím 9.

Oficiální Unity fórum je velice aktivní, často tam lze najít odpovědi na téměř jakékoliv otázky. Odpovídají tam i vývojáři samotného Unity. Také je k nalezení nemalé množství různých blogů a videí s tutoriály. Proto komunitě udělím 8.

V Unity se také velice jednoduše začíná. Nemám žádné zkušenosti s počítačovou grafikou, a i přes to pro mě bylo triviální vytvořit jednoduchou Unity aplikaci. Některé Unity specifické koncepty, jako například coroutines (korutiny), vyžadují více času a jsou zrádné, ale i přes to je nepovažuji moc složité na pochopení. Proto bude getting into ohodnotím 7.

Unity nemá otevřený zdrojový kód, jen C# část, která je navíc pouze „reference only". To znamená, že ji nelze modifikovat, pouze do ní nahlížet (pro účely optimalizace a ladění neobvyklého chování) [20]. Existuje možnost pro organizace dostat se k celým zdrojovým kódům, což je ale mimo můj dosah. Proto Unity udělím 3 body za otevřenost.

2.4.1.5 Godot

Godot je 3D engine pro vývoj grafických aplikací a her, který je vyvíjen komunitou a má otevřený zdrojový kód. V době psaní této práce však Godot nemá hotovou implementaci AR pro Android [21]. Jelikož má otevřený zdrojový kód, tak je možné tuto funkcionalitu implementovat bez podpory Godotu, ale to je přehnaně náročné a zbytečné. Implementace pro iOS existuje a je

funkční, nicméně tato práce vyžaduje aplikaci pro OS Android, což nelze jednoduše splnit. Proto přenositelnost ohodnotím 1.

Výběr programovacích jazyků pro vývoj v Godotu je široký, Godot plně podporuje C# 8 [22], C, C++ a GDScript. (vizuální „programovací“ jazyk VisualScript zde vynechám) C je stejně jako C++ nízkoúrovňový, staticky typovaný jazyk překládaný do strojového kódu, ale na rozdíl od C++ nepodporuje objektově orientované programování. GDScript je dynamicky typovaný jazyk překládaný do strojového kódu. Jeho výhoda je nejlepší podpora v Godotu ze všech oficiálně podporovaných jazyků. Dále Godot umožňuje udělat pomocí bindings podporu dalších jazyků, díky čemuž vznikla podpora další škály jazyků, které zde nebudu dále rozepisovat. Výběr programovacích jazyků ohodnotím 9 díky široké škále podporovaných jazyků a podpory poměrně nové verze C#.

Dokumentace Godotu je rozsáhlá, třídy a metody jsou rozsáhle popsány, spolu s ukázky kódu. Dokumentace proto ohodnotím 9.

Komunita je malá, nebyly k nalezení skoro žádné tutoriály na AR, a i například přidání modelu dynamicky za běhu nebylo lehké najít. Z toho důvodu mu udělím 2 body za kvalitu komunity.

Godot cílí na jednoduché použití. Samotné stažení, spuštění a vytvoření první aplikace je velice jednoduché, proto getting into ohodnotím 8 body.

Godot má otevřený zdrojový kód pod MIT licencí, za což mu udělím 9 za otevřenost.

2.4.1.6 WebXR

WebXR je API pro tvorbu VR / AR aplikací na webových technologiích.

WebXR je v době psaní této práce ve stavu „Candidate Recommendation Snapshot“ [23], což znemožňuje jeho (nebo knihoven nad ním postavených) využití z důvodů možných změn, nebo jeho případného zrušení, proto mu v tomto porovnání nebudu dále věnovat pozornost.

2.4.1.7 Závěr

Po zvážení všech přínosů a nedostatků, a zvážení hodnocení které mi vyšlo (viz tabulka 2.1) jsem se rozhodl použít Unity, jelikož poskytuje dobrou podporu AR, je jednoduché naučit se v něm pracovat a programuje se v C#. C# je jazyk, který již umím, je vysokoúrovňový a díky statickému typování a kompilaci poskytuje větší statickou kontrolu než například Python.

Tabulka 2.1 Porovnání výběru technologií

	Předchozí implementace	Unreal Engine	Unity	Godot
Přenositelnost	0	8	7	1
Programovací jazyk(y)	7	5	8	9
Kvalita dokumentace	4	7	9	9
Kvalita komunity	5	4	8	2
Getting into	2	6	7	8
Otevřenost	9	7	3	9
Celkem	27	37	42	38

2.4.2 Knihovna pro DI

Pro dependency injection existuje mnoho knihoven pro .NET framework. Kvůli technickým detailům Unity potřebuji DI knihovnu přímo pro Unity. Proto jsem zvolil knihovnu pro DI Zenject. Knihovna má otevřený zdrojový kód a je využívána například AR hrou Pokémon GO [24], nebo novější AR hrou The Witcher: Monster Slayer.

2.4.2.1 Dependency Injection

Dependency injection, česky Vkládání závislostí, je návrhový vzor – technika Inversion of Control (IoC). DI odděluje vytváření objektů (tříd/struktur) (závislostí) od jejich využití. Namísto explicitního volání konstruktoru objektu a předávání mu potřebných závislostí je objekt vytvořen DI frameworkem a jeho závislosti jsou mu automaticky injektovány, dle dříve nainstalovaných/nadefinovaných závislostí.

2.4.3 Funkční a nefunkční požadavky

2.4.3.1 Funkční požadavky

F1 Zjištění aktuální pozice zařízení – aplikace bude schopna využít lokačních modulů pro zjištění aktuální pozice zařízení ve světě.

F2 Zobrazení virtuálních budov v reálné scéně – aplikace bude schopna zobrazit virtuální modely budov ukotvené do reálné scény.

F3 Zobrazení informací o budově – po kliknutí na virtuální budovu budou uživateli zobrazeny informace o dané budově.

F4 Nasvícení virtuálních budov – virtuální budovy budou nasvíceny ze směru, ze kterého svítí slunce a budou vrhat stíny správným směrem.

F5 Šipka (hint) pozice virtuální budovy – pokud bude virtuální budova mimo zorné pole zařízení, tak se na okraji displeje zobrazí šipka, jakým směrem se má uživatel podívat.

F6 Mapa – aplikace bude mít mapu všech budov, aby uživatel věděl, kam má jít, aby mohl vidět virtuální budovu.

F7 Informace o budovách v mapě – po kliknutí na budovu v mapě se uživateli zobrazí popis a základní informace o budově.

2.4.3.2 Nefunkční požadavky

N1 Aplikace bude implementována pro Android API 24 až 33.

N2 Aplikace bude lehce přenositelná na iOS 15.4.

N3 Aplikace bude implementována ve vizualizačním engine Unity 2022.2.0a10.

N4 Aplikace bude využívat techniky dependency injection knihovny Zenject 9.2.0.

N5 Aplikace bude využívat Unity AR Foundation pro trackování prostoru a ukotvení virtuálních objektů do reálné scény.

N6 Aplikace bude podporovat více spolupracujících lokačních modulů.

N7 Aplikace bude modely a textury stahovat z API až v době, co je bude potřebovat.

N8 Aplikace bude umět zobrazit HTML5 stránky.

N9 Aplikace bude mít mapu implementovanou za pomoci webových technologií.

Kapitola 3

Návrh

V této kapitole se zaměřím na návrh architektury aplikace – zobrazování budov a jejich správa na scéně, návrh komunikačního rozhraní s API, získávání lokace a její využití atp. Z toho mi vzejdou požadavky na změny API a ovlivní mi požadavky na uživatelské rozhraní.

Ač je aplikace určena pro zobrazování virtuálních budov, a odpovídá tomu i zadání této práce, tak se vynasnažím architekturu aplikace navrhnout nezávisle na typu zobrazovaných objektů. Specifické vlastnosti a požadavky VMČK oddělím od zbytku kódu za účelem jednoduchého oddělení, s minimální změnou kódu.

Aplikace se bude skládat z několika částí. Část starající se o získání modelů z API, jejich ukládání do mezipaměti a načítání ze získaných formátů do Unity objektů (game objects) scény. Dále část starající se o správu modelů na scéně, jejich přidávání/odstraňování a de-duplikace těchto operací. Další část se bude starat o získávání a poskytování lokace v reálném světě ze senzorů jako je například GPS, poskytovat ukotvení virtuální scény a korekci pozice ve virtuální scéně. UI část bude uživateli zobrazovat informace o budovách a poskytovat možnost volby období budov. Aplikace je popsána obrázkem 3.1.

3.1 API klient

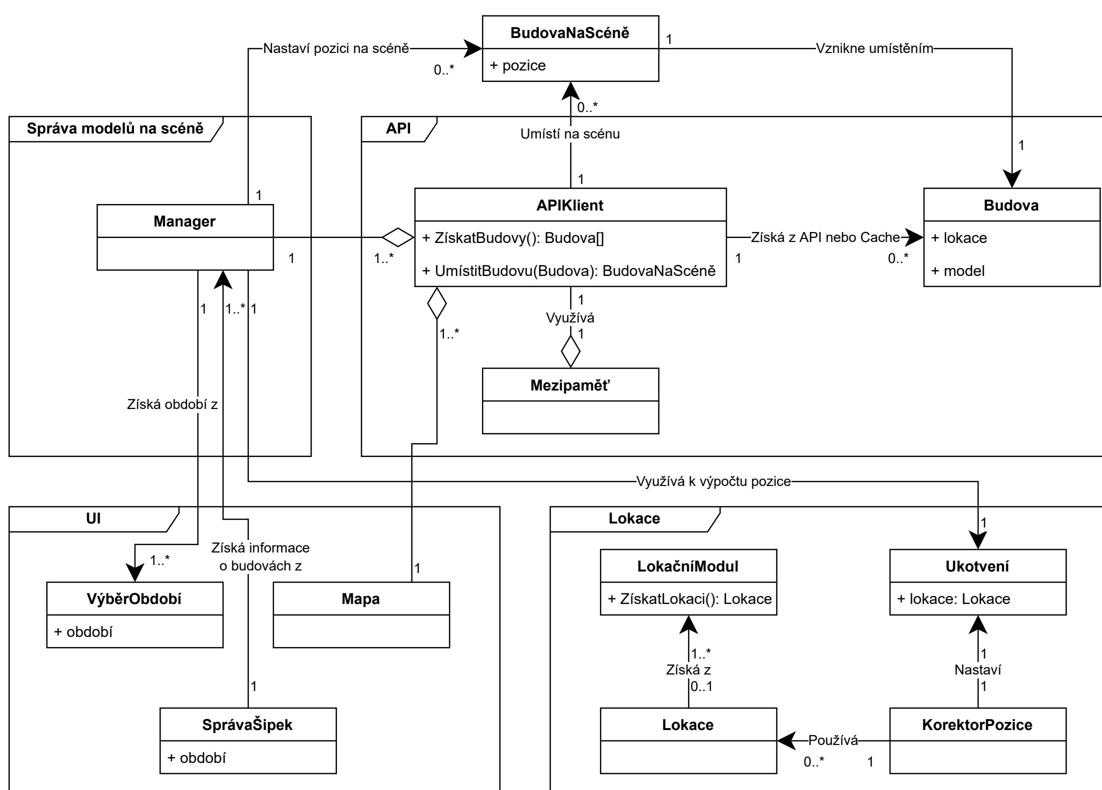
API vrací odděleně modely a textury. Tyto objekty budu ukládat do mezipaměti (cache) nezávisle na sobě. Dále oddělím rozšířenou funkcionalitu jako je zobrazování mapy a rozšířené informace o budovách od „hlavní“ části API klienta. Kvůli tomu rozdělím API klienta na tři části: interní, hlavní, a rozšiřující.

3.1.1 Části

Interní část se bude získávat data ze samotného API, bude abstrahovat REST rozhraní, a vracet objekty (téměř) korespondující se samotnými daty. Tato část bude popsána rozhraním `IInternalApiClient` a implementována třídou `InternalApiClient`.

Hlavní část API klienta bude popsána rozhraním `IApiClient` a implementována třídou `ApiClient`. V této části budou již skryté implementační detaily jako například existence textur a formát 3D modelů. Třída `ApiClient` bude využívat návrhový vzor Fasáda (Facade) nad rozhraním `IInternalApiClient`.

Rozšiřující část API klienta se bude starat o věci specifické k této aplikaci, nezávislých od generického zobrazování 3D objektů v rozšířené realitě. To znamená specifické požadavky



Obrázek 3.1 Doménový model architektury aplikace

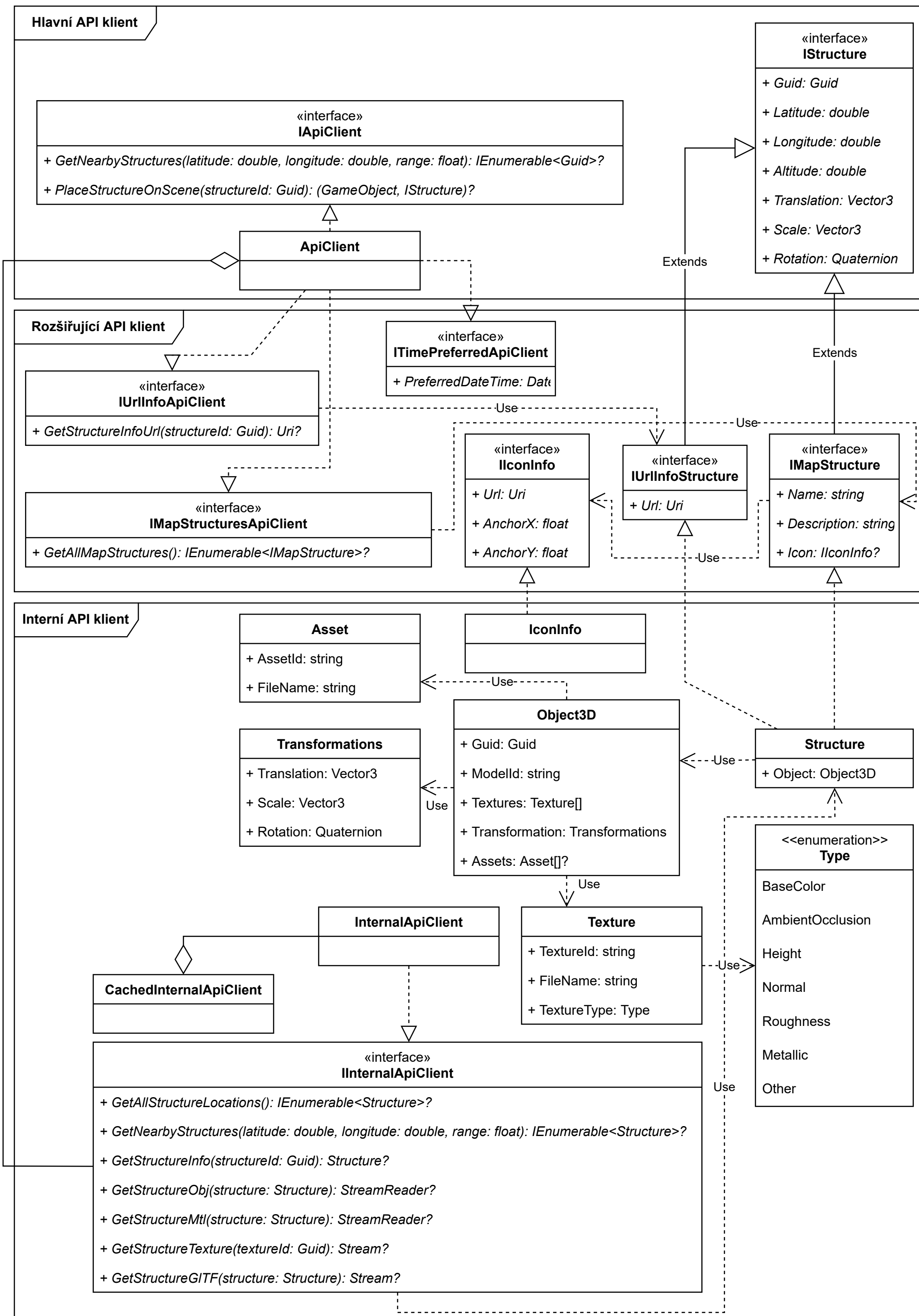
mapy (popsány rozhraním `IMapStructuresApiClient`), informace o budově (popsány rozhraním `IUrlInfoApiClient`), a možnost nastavení preferovaného roku budov (popsána rozhraním `ITimePreferredApiClient`).

Mezipaměť modelů udělám, díky abstrakci obou tříd za rozhraní, návrhovým vzorem proxy. V implementaci bude mezipaměť implementovat třída `CachedInternalApiClient`, která bude také implementovat rozhraní `IInternalApiClient`. Rozdíl však je, že požadavky na modely které nenajde v mezipaměti, bude delegovat na jinou třídu implementující `IInternalApiClient`, což je v tomto případě `InternalApiClient`. Viz obrázek 3.2.

3.1.1.1 Návrhový vzor fasáda

Tento návrhový vzor se používá pro zjednodušení práce se složitým objektem, případně více objekty. Fasáda abstrahuje tuto složitost za jednoduché, snadno použitelné rozhraní.

V mém případě mám třídu implementující `IInternalApiClient`, která poskytuje model rozložený na 3D model a jeho textury. To je však z pohledu použití zbytečně složité, stačila by mi metoda na získání již složeného modelu, připraveného k zobrazení. Proto mám třídu implementující `IApiClient`, která od třídy implementující `IInternalApiClient` získá potřebné data, ty složí do objektu, a vrátí ho. Díky tomu při volání metod z `IApiClient` není třeba vědět, jak skládání objektu probíhá.



Obrázek 3.2 Diagram tříd API klient části aplikace

3.1.1.2 Návrhový vzor proxy

Návrhový vzor proxy se využívá v situacích, kdy je třeba provést operaci před/po zavolání nějaké metody jiného objektu. Nový objekt (proxy) původního objektu implementuje stejné rozhraní jako původní objekt, deleguje volání metod na původní objekt, a přidává do volání metod vlastní logiku.

V mém případě nebude třída `CachedInternalApiClient` v určitých případech volat metody třídy `InternalApiClient`, ale nejdříve se podívá do mezipaměti. Pokud v ní najde kýžený objekt, tak vrátí ten. Pokud ho však nenajde, tak deleguje volání na `InternalApiClient`, vrácený objekt uloží do mezipaměti a teprve poté vrátí objekt volajícímu. Díky tomu není nutné pokaždé stahovat všechny data z API, ale mohou se získat z lokální mezipaměti.

3.1.2 Změny v API

API je aktuálně vytvořeno jako neveřejné. Z toho plyne, že některé metody jako například `/structures`, vrací všechna data při nezadaných filtrovacích parametrech. To je vhodné například pro aplikaci správy modelů, nehodí se to však pro využití v této aplikaci. Proto je potřeba navrhnout potřebné změny API.

Vyhledávání dle pozice zařízení a dle roku

Aktuální verze API má navrhnuto ve `/structures` možnost předat objekt s aktuální pozicí. To je třeba implementovat, API by mělo vracet pouze budovy maximálně ve vzdálenosti, kdy dává smysl je uživateli zobrazit na zařízení.

Vyhledávání dle roku je také navrhnuto a není implementováno, navíc vyhledávání funguje pouze na principu rozsahu „od-do“. To je pro požadavky aplikace nevhodné, proto navrhuji nový parametr `preferredDate`, díky kterému bude možné zadat preferovaný rok varianty budovy. API po přijetí tohoto roku klientu pošle variantu budovy, která bude vyhovovat zadanému roku, nebo bude nejblíže zadanému roku ze všech variant.

Automatické dosazení parametrů

Automatické dosazení parametrů není v prototypu aplikace implementováno. Tyto parametry je možné manuálně měnit, což vyžaduje zbytečně složitou interakci s uživatelem. To je nepraktické řešení ve finální aplikaci (kdy by si uživatel musel správně zvolit jaké je počasí, případně denní doba), nebo zbytečnou komplexitu na straně klienta. (bylo by nutné si zjišťovat například aktuální počasí, a tuto informaci poté posílat na API)

Proto navrhuji, aby si tyto informace zjišťovalo API samotné z pozice, kterou klient pošle. To umožní i případně tyto informace ukládat do cache, a klientům je poté vracet bez zbytečného volání na další API. Jelikož je potřeba zachovat původní funkčnost – vrácení všech variant bez aplikovaných filtrů – tak navrhuji tuto možnost automatické dosazení těchto parametrů neaplikovat pokud nejsou dostupné parametry lokace. Pokud jsou dostupné parametry lokace, tak API automaticky vybere vhodné parametry (počasí z nějakého API na zjišťování počasí, denní dobu dle aktuálního času a pozice klienta, …).

Transformační matice modelu

Matice 3×3 je v aktuální situaci nepoužitelná. Spravit ji lze nahrazením za 4×4. Nicméně i s maticí $4{\times}4$ se špatně pracuje, proto navrhuji místo matice použít objekt který by obsahoval vektor pozice, vektor velikosti, a quaternion rotace. Objekt transformací ve formátu JSON by tedy měl vypadat následovně:

```
"transformation": {
  "translation": [ 0, 0, 0 ],
  "scale": [ 1, 1, 1 ],
  "rotation": [ 0, 0, 0, 1 ],
}
```

Výpis kódu 1 Příklad JSON objektu transformací

Textury

Textury by u sebe měly mít informaci `textureType` o jaký typ se jedná (`basecolor`, `height`, `ambientocclusion`, `normal`, `roughness`, `metallic`), aby bylo možné odlišit různé typy od sebe, a nebylo nutné spoléhat na správné pořadí textur v poli. To by také neumožnilo vynechání jedné textury pokud by nebyla potřeba, nebo možnost zobrazit budovu při nedostupné potřebné textuře (například ambient occlusion mapu), ale model nezobrazit pokud se bude jednat o „důležitou“ (například base color) texturu.

Popis a název budovy

Popis a název by měl být lokalizovatelný, nejlépe jazykovou informací v hlavičce požadavku (v `Accept-Language`). Dále by bylo vhodné do objektu budovy přidat odkaz na webovou stránku s informacemi o budově, která musí být optimalizována pro mobilní zařízení.

Mapa

Pro mapu navrhuji mít novou lokaci v API, odkud by bylo možné zjistit pouze základní informace o všech budovách. Dále by bylo vhodné pro každou budovu přidat odkaz na malý obrázek pro zobrazení v mapě, a informace o pozici uchycení tohoto obrázku využité při přibližování/oddalování mapy. (pozice zleva a shora, kde 0 udává vlevo/nahoře, a 1 udává vpravo/dole) JSON objekt ikony na mapě by měl vypadat například následovně:

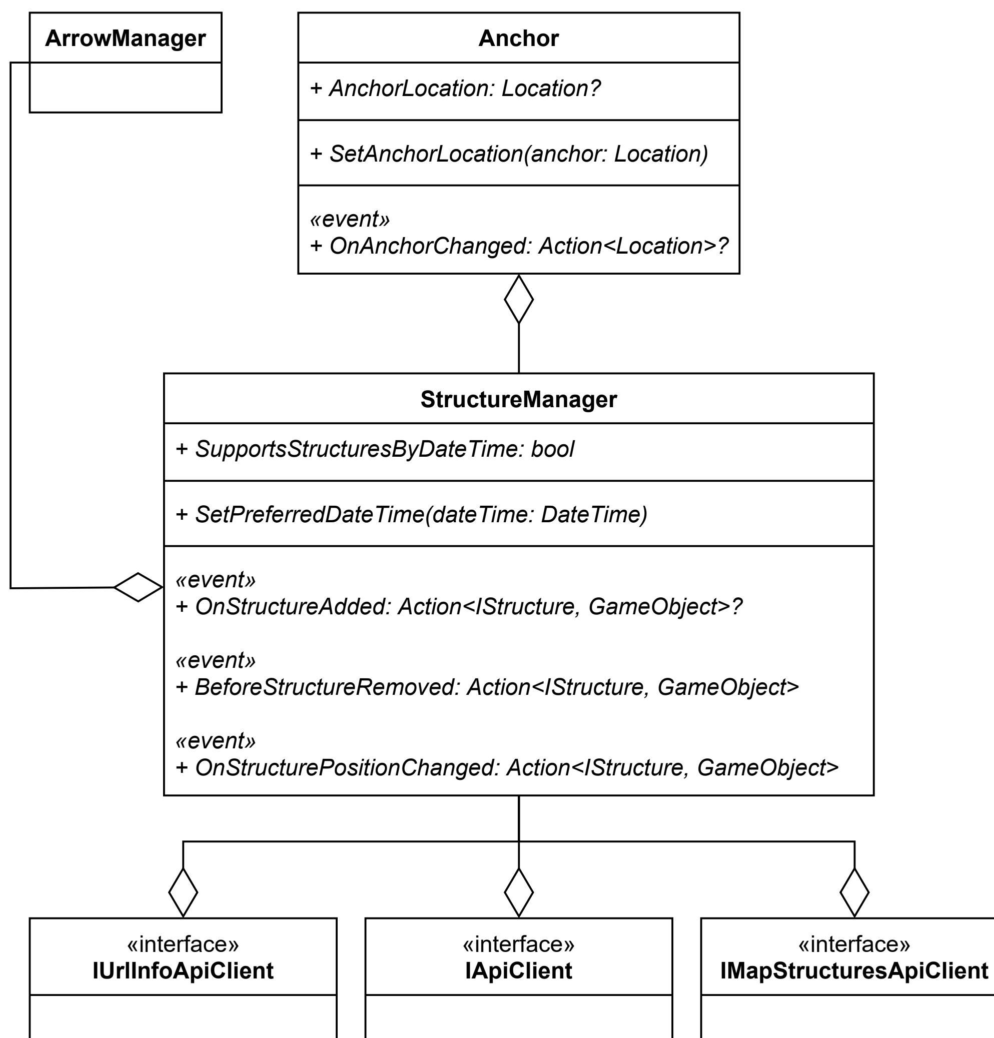
```
"icon": {
  "top": 0.5,
  "left": 0.5,
  "url": "https://www.kralovskavennamesta.cz/img/kropacka.png"
}
```

Výpis kódu 2 Příklad JSON objektu ikony na mapě

3.2 Správa struktur

Jelikož API klient pouze získá seznam všech modelů budov v okolí a umístí model na scénu, tak je třeba spravovat již umístěné modely na scéně a nastavovat metadat nově umístěných modelů na scénu. O to se bude starat třída `StructureManager`. Jejím účelem bude v potřebných situacích (po změně lokace ukotvení, změně období budov, …) získat budovy které se mají zobrazit na scéně, zkontrolovat aktuální budovy na scéně a případně nějaké odebrat, zavolat API klienta aby umístil na scénu budovy které tam aktuálně nejsou ale měly být, a po umístění jim nastavit parametry jako pozici na scéně (kde (0, 0, 0) koresponduje s lokací ukotvení), velikost, rotaci atp. Návrh třídy lze vidět na obrázku 3.3.

`StructureManager` se také bude starat o možnost kliknutí na budovu (v situaci kdy bude existovat implementace `IUrlInfoApiClient`) a otevření informací o dané budově. Také bude poskytovat informace o budovách potřebných pro zobrazení šipek ukazujících na virtuální budovy mimo zorné pole.



Obrázek 3.3 Diagram tříd správy modelů

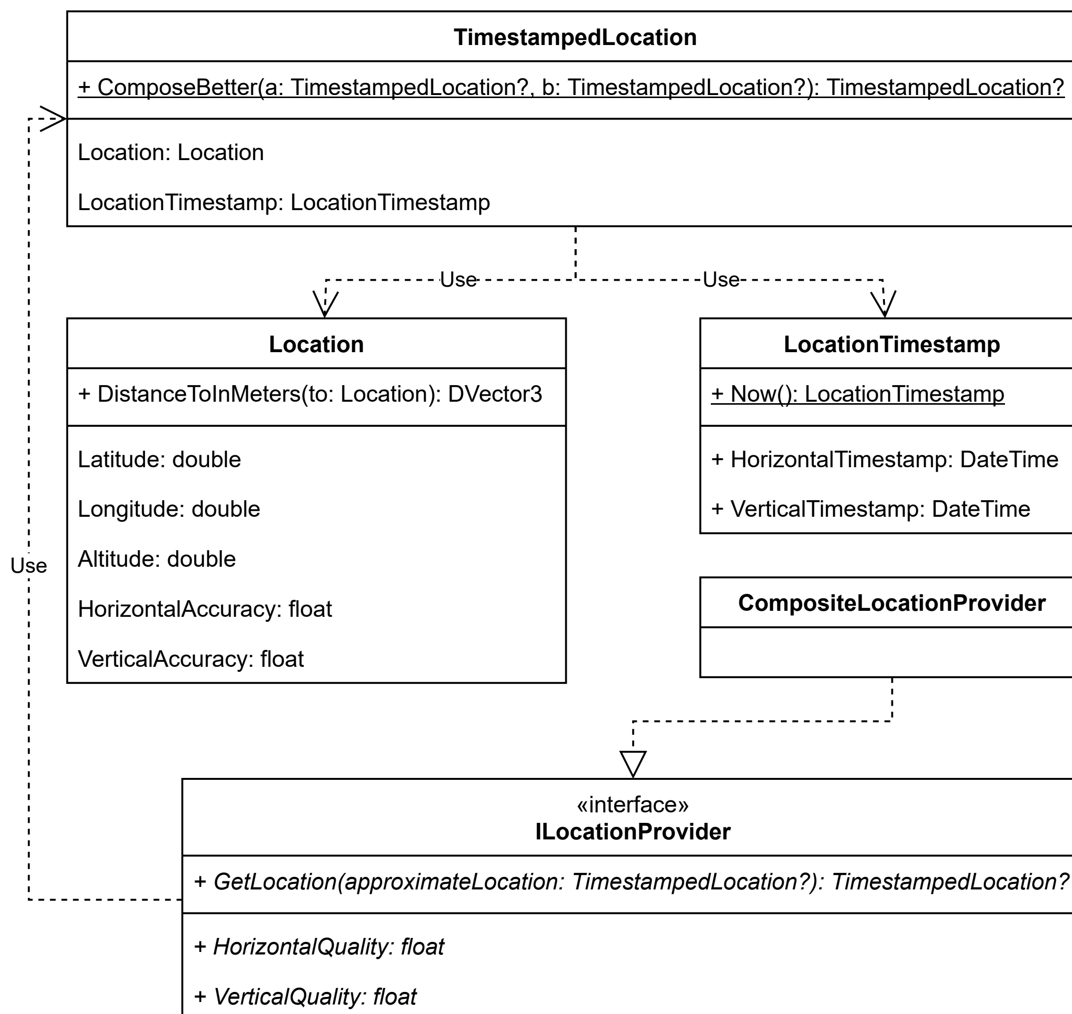
3.3 Lokalizačních moduly a jejich API

Ondřej Slabý ve své práci [4] navrhl rozhraní pro lokalizační a trackovací moduly, které mělo účel oddělení modulů od specifik OS Android. Já z důvodu nevyužití trackovacích modulů nebudu potřebovat pro ně určené rozhraní. Dále nebudu potřebovat oddělení od specifik OS Android, jelikož oddělení od specifik OS bude mít na starost .NET API a Unity API. Rozhraní `ILocationProvider` které musí lokalizační moduly implementovat je popsáno na obrázku 3.4.

Důležitá změna oproti původnímu návrhu je zrušení čekání na výstup z modulů před každým vykresleným snímkem obrazovky. Výpočet modulům může trvat delší dobu (až nízké jednotky sekund), na jiném vlákně aby zbytečně nebylo zatíženo hlavní, a poslední vypočtený výsledek

s časovým razítkem od počátku výpočtu bude vracet metoda `GetLocation`. Ta také v případě chyby modulu vrátí `null`, což bude signalizovat chybu, a daný modul se přeskočí při výpočtu výsledné pozice. Metoda `GetLocation` bude jako parametr bude brát polohu, kterou bude moci využít pro zlepšení/zrychlení výpočtu.

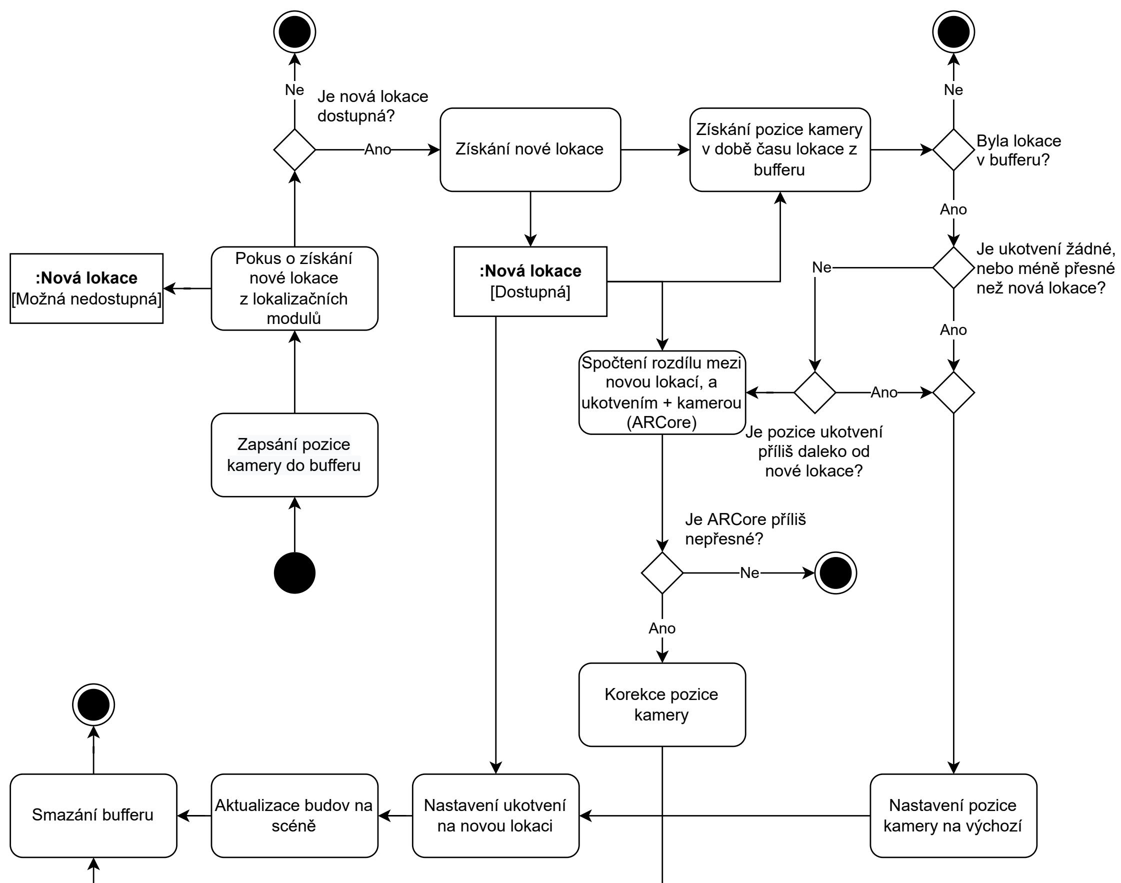
Dále se budu využívat více lokalizačních modulů a jejich výstupů ke zlepšení lokace z aktuálního modulu, pokud to bude možné. O to se bude starat třída `CompositeLocationProvider`, která bude zastřešovat všechny ostatní lokalizační moduly. (její „děti") Při inicializaci si moduly setřídí dle jejich (odhadované) přesnosti, a poté při každém zavolání metody `GetLocation` postupně, od nejhoršího po nejlepší (dle přesnosti), zavolá na dítě metodu `GetLocation`. Z Výstupu z této metody a z předchozí lokace sestaví novou lokaci, která bude sestavená z lepších složek původních lokací.



Obrázek 3.4 Diagram tříd lokačních modulů a jejich API

3.3.1 Ukotvení a kalibrace virtuální scény vůči scéně reálné

ARCore po inicializaci umístí virtuální kameru na pozici (X: 0, Y: 0, Z: 0) odkud se hýbe, kde jedna jednotka je rovna jednomu metru v reálném světě. Proto je třeba virtuální scénu ukotvit vůči reálné – po inicializaci ARCore si zaznamenat, že (X: 0, Y: 0, Z: 0) ve virtuální scéně je rovna nějakým souřadnicím v reálném světě. To mi také poté umožní provádět korekci ARCore. Ukotvení a kalibrace je popsána obrázkem 3.5.



Obrázek 3.5 Diagram aktivit nastavení ukotvení a kalibrace polohy

Při ukotvení je nutné myslet na možnost, že v době potřeby ukotvení nebude dostupná lokace z modulů, nebo že ukotvení bylo provedeno příliš nepřesnou lokací, tím pádem je třeba virtuální scénu znovu ukotvit. Dále jsem se rozhodl virtuální scénu znovu ukotvit v případě příliš velké vzdálenosti zařízení od původního ukotvení, což také využiji k aktualizaci budov na scéně. To mi také umožní eliminovat nastřádané výpočetní nepřesnosti. Pro přesnou vzdálenost se rozhodnu při implementaci.

Další problém nastává kvůli době výpočtu lokace. V době, kdy z lokačních modulů dostanu novou lokaci, se kamera ve virtuální a reálné scéně může nacházet na jiné pozici, než kde byla v době počátku výpočtu lokace (například v době pořízení snímku, ze kterého se lokace vypočítává). To by při korekci vytvořilo nepřesnost, kterou musím eliminovat. Proto pro korekci, která se bude provádět před každým vykresleným snímkem, musím použít pozici kamery v době počátku výpočtu pozice.

Jelikož nějaké moduly (například modul získávající pozici z GPS) nedokáží určit počátek výpočtu, budu si uchovávat posledních několik pozic kamery a časy, ve kterých se na dané pozici kamera nacházela. Tento problém vyřeší kruhový buffer, který současně s pozicí kamery bude také uchovávat i čas, podle kterého umožní vyhledávání. Pokud se nepodaří pozici nepodaří v čase najít (již byla přepsána), tak se buffer zvětší.

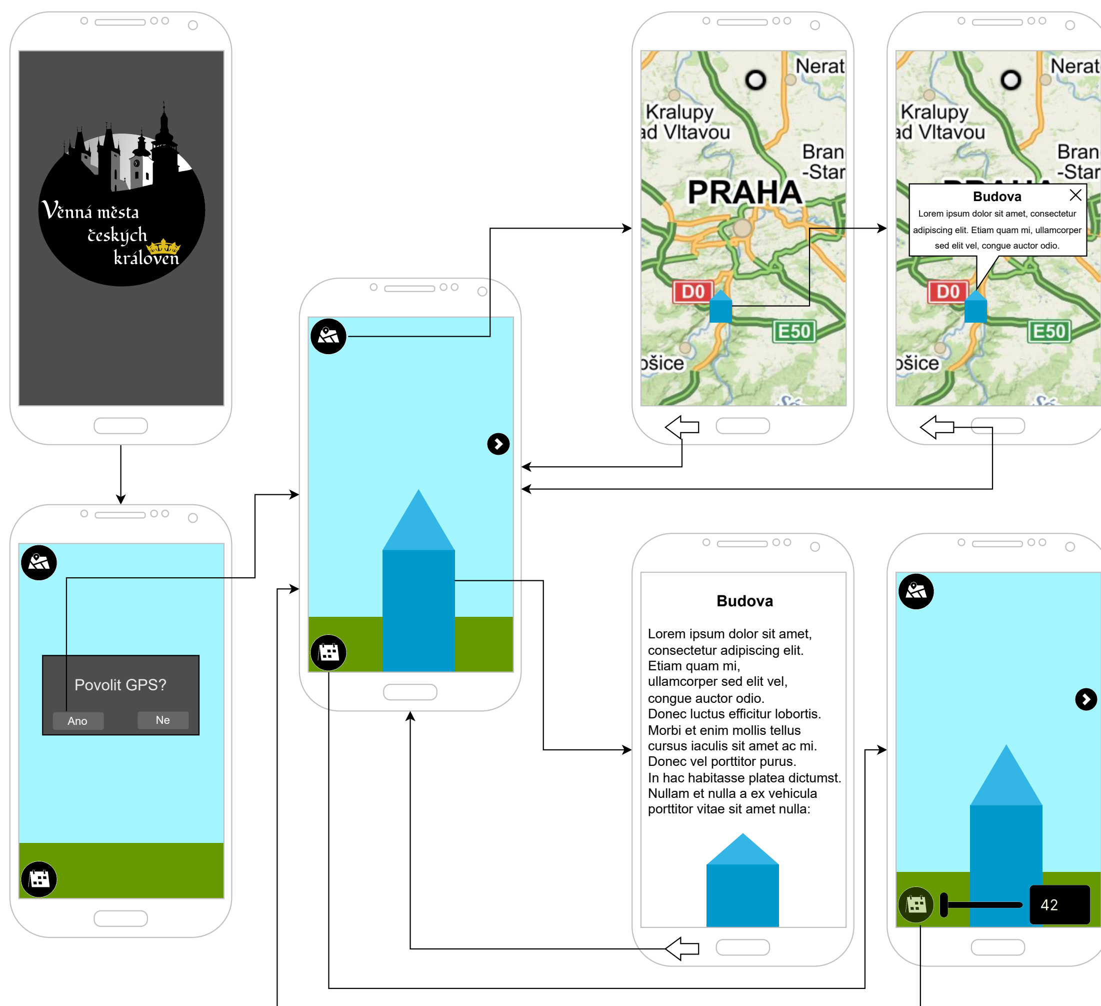
3.4 Uživatelské rozhraní

Při návrhu uživatelského rozhraní se vynasnažím držet se návrhového vzoru KISS – Keep It Simple Stupid. Lukáš Brhlík ve své bakalářské práci [17] navrhl uživatelské rozhraní pro původní prototyp aplikace, nicméně jeho návrh počítal s technologiemi původní aplikace a byl dle mého názoru zbytečně komplikovaný.

Z toho důvodu jsem se rozhodl uživatelské rozhraní omezit se na nutné minimum. Aplikace bude mít celkem 4 obrazovky. Úvodní obrazovku, obrazovku s mapou, obrazovku s informacemi o budově, a AR obrazovku ve které se budou zobrazovat budovy v rozšířené realitě a šipky na budovy mimo zorné pole. Navigace mezi obrazovkami a interakce s nimi je popsána diagramem na obrázku 3.6.

Mapa bude mít v aplikaci tři účely: informovat uživatele o lokacích budov, zobrazovat aktuální lokaci uživatele a poskytovat zjednodušené informace o daných budovách (název a zjednodušený popis).

Jelikož v závislosti na použitých modulech a na aktuálním operačním systému bude třeba zobrazit dodatečné dialogy (například požadavek o přístup k [GPS] lokaci v OS Android), tak jsem je v obrázku 3.6 zobrazil po inicializaci AR obrazovky. Nicméně samotným modulům nic nebrání tento, nebo podobný dialog zobrazit kdykoliv při běhu aplikace.



Obrázek 3.6 Wireframe uživatelského rozhraní

Kapitola 4

Implementace

V této kapitole popíši rozhodnutí, které jsem v průběhu implementace musel udělat, problémy které jsem potkal a způsoby jakými jsem je vyřešil. Nastíním také technické detaily některých dostupných řešení, aby bylo zřetelné, proč jsem si dané řešení vybral.

4.1 Implementační detaily

4.1.1 Nullable typy

C# ve verzi 8.0 zavedl vlastnost „Nullable reference types“ [25], která umožní vynutit ne-null hodnoty referencí, a zavádí novou syntaxi pro nullable reference typy (například non-nullable `string` má nullable ekvivalent `string?`). Unity bohužel na použití této vlastnosti není připraveno, nicméně dle mého názoru výhody převyšují nevýhody. Z toho důvodu jsem se rozhodl pro následující řešení problémů které se objevily:

Injektovaný field Fields injektované atributem `[Inject]` jsou ze začátku nedefinované, což vyústí ve výchozí hodnotu, která je pro reference null. Skutečnost, že hodnota následně není null je vidět z atributu `[Inject]`. Proto hodnotu nastavím na `null!`.

Field inicializovaný ve Start metodě Unity odrazuje od používání konstruktorů (mohou být volány na jiném vlákně, a mohou se volat vícekrát), proto se většina inicializační logiky provádí v metodách `Start`, nebo `Awake`. To ovšem překladač neví, a hlásí varování. Na první pohled však není vidět že field je inicializovaný, proto jsem udělal následující extension metodu:

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static T LateInit<T>() where T : class?
{
  return null!;
}
```

Výpis kódu 3 LateInit metoda

jejíž jediný účel je dokumentování kódu. V případě použití si doporučuji třídu `Extensions` ve které se nachází, staticky importovat: `using (...).Extensions;`, a poté ji použít například: `string _initSceneName = LateInit<string>();`.

Zapnutí vlastnosti „Nullable reference types“ Původně jsem si myslel, že si tuto vlastnost zapnu nastavením atributu `nullable` na `enable` v souboru `.csproj` [26]. Unity ale tento soubor generuje, a přepisuje. Další možnost je použití souboru `Directory.Build.props`, což se ukázalo jako nepoužitelné, protože kompilátor tento soubor přeskočil.

Tuto vlastnost lze také vynutit přidáním přepínače `-warnaserror:nullable` kompilátoru [27]. To se však také ukázalo jako nepoužitelné, protože některé knihovny se do Unity instalují přidáváním zdrojových kódů do projektu, a tyto zdrojové kódy nejsou na tento přepínač připraveny.

Proto jsem se rozhodl do každého souboru přidat direktivu preprocesoru `#nullable enable`. To má nevýhodu že to lze lehce zapomenout udělat, ale nepodařilo se mi přijít na jiný, lepší způsob.

4.1.2 Asynchronní programování a coroutines

Při implementaci se ukázala potřeba v mnoha případech mít asynchronní kód, například při stahování modelů z API, jelikož nechceme na několik sekund pozastavit celou aplikaci.

K tomu se nabízí dvě hlavní možnosti, C# Tasks a Unity coroutines. Vynechávám využití více vláken, jelikož jsem v situacích kde jsem potřeboval asynchronní kód nebyl omezen výkonem, ale čekáním na nějaký další prvek, většinou API. S více vlákny by vzniklo nemálo synchronizačních problémů které by si bylo potřeba ohlídat. Unity také požaduje, aby prakticky všechny metody byly volány z hlavního vlákna.

4.1.2.1 C# Tasks

C# umožňuje vytvořit asynchronní úlohu (task). Jedná se o metodu s klíčovým slovem `async` v její deklaraci, a návratovým typem většinou (existují výjimky) `Task` nebo generickým `Task<T>`. V těle takovéto metody lze poté použít klíčové slovo `await`, které označuje bod ve kterém se čeká na nějaký `Task`. Tato metoda se interně promění ve třídu (s instancí na haldě) reprezentující stavový automat, který vždy provede kus kódu dané metody (než potká klíčové slovo `await`), a vystoupí z těla metody (zpátky volajícímu, do jiné asynchronní metody, nebo do blokujícího čekání v normální metodě), aby mohlo pokračovat spouštění kódu na jiném místě zatímco se čeká na dokončení úlohy.

Poté, když se volající metoda dostane do situace kdy „znovu zavolá“ (stavový automat ve volající metodě se rozhodne pokračovat) tuto metodu, tak se rozhodne dle vnitřního stavu konečného automatu do jaké části kódu se vrátí a zkontroluje jestli úloha na kterou se čekalo je již splněná. Pokud je tak se nastaví stav automatu na jiný, a pokračuje se ve spouštění kódu dané metody, pokud není, tak se znovu vystoupí z těla metody a dále se čeká.

Tento přístup je vhodný v situacích kdy se čeká na nějakou blokující operaci, která není náročná na procesorový čas, například stahování dat ze sítě. Takovéto úlohy běží na stejném vlákně, což má výhodu že není třeba řešit synchronizaci vláken. C# ale také umožňuje úlohu spustit na jiném vlákně pomocí `Task.Run`, což je vhodné na úlohy vyžadující mnoho procesorového času.

4.1.2.2 Unity coroutines

Coroutines jsou v Unity jiný způsob jak psát asynchronního kód běžící na stejném vlákně. Coroutines jsou implementovány pomocí vedlejších účinků C# iterátorů. C# iterátor je objekt implementující rozhraní `IEnumerator`, které vypadá následovně:

```
public interface IEnumerator
{
  object Current { get; }
  bool MoveNext();
  void Reset();
}
```

Výpis kódu 4 Rozhraní IEnumerator

Iterátor obsahuje metodu `Reset` kterou není třeba se zabývat [31], metodu na posunutí iterátoru na další prvek a aktuální prvek na který iterátor ukazuje.

Aby nebylo potřeba pro každý iterátor manuálně vytvářet novu třídu, tak C# obsahuje syntaktický cukr na vytváření iterátorů. Namísto nové třídy stačí vytvořit metodu vracející `IEnumerator` a v jejím těle využít speciální klíčové slovo `yield`, které buď přesune iterátor na další prvek (`yield return prvek`), nebo posune iterátor na konec (`yield break`). Díky tomu lze jednoduše udělat například iterátor vracející sekvenci *1, 2* pokud je lichý den a sekvenci *1, 2, 3, 4* pokud je sudý den, pomocí následujícího kódu:

```
IEnumerator Sequence()
{
  yield return 1;
  yield return 2;
  if (DateTime.Now.Day % 2 == 0) yield return 3;
  else yield break;
  yield return 4;
}
```

Výpis kódu 5 Příklad iterátoru

Unity využívá faktu, že pro přesun iterátoru je potřeba spustit uživatelský kód k vytvoření předvídatelného asynchronního kódu. Coroutines jsou spouštěny komponentami objektů scény pomocí metody `StartCoroutine`, nebo lze `Start` metodu komponenty (viz 4.1.2.2) deklarovat vracející `IEnumerator`. Unity poté každý snímek všechny iterátory/coroutines komponent povolených objektů objektů[1] posune o jeden prvek.

Posunutím iterátoru se provede kód v těle iterátoru než se potká klíčové slovo `yield`, což vytváří (jedno-vláknový) kooperativní multitasking. Podstatné je zmínit, že Unity zahazuje prvek iterátoru, proto je možné vracet například `null` nebo 0, nicméně lepší je vracet `null`, jelikož na rozdíl od 0 nedochází k jehož zapouzdření, což má malý výkonnostní benefit. Pokud však prvek vrácený iterátorem je jiný iterátor, tak ho Unity nejdříve přesune na konec (spustí), což umožní volání jiných coroutines.

Nevýhoda coroutines je však vracení výsledků. Rozhraní `IEnumerable`, nebo ještě lépe generické `IEnumerable<T>`, sice obsahuje mechanismus jak z něj vracet hodnoty (je to dokonce primární účel iterátoru), ale Unity tyto výsledky zahazuje. Lze udělat wrapper nad Unity coroutine který při každém posunutí iterátoru / vrácení výsledku uloží vrácenou hodnotu do nějaké třídní proměnné, ale tento přístup se mi nezdá vhodný, jelikož vede na podstatně víc kódu, a je pomalejší jelikož při každé `yield return` operaci se výsledná hodnota navíc musí přiřadit do proměnné.

[1] Pozor, zakázaných komponent ano, ale zakázaných objektů scény ne.

To však lze vyřešit předáním metody, kterou coroutine zavolá před jejímž skončením (před `yield break`, nebo implicitním `yield break` na jejím konci). Je však třeba si zkontrolovat že coroutine tuto metodu opravdu zavolá. To se při implementaci ukázalo jako častý zdroj chyb, který se obtížně hledá. Získání hodnoty z coroutine, za použití vnořených metod, poté vypadá následovně:

```
IEnumerator CoroutineReturningAnInt(Action<int> onComplete) { ... }

IEnumerator Function()
{
  int val;

  void OnCoroutineComplete(int result)
  {
    val = result;
  }

  yield return CoroutineReturningAnInt(OnCoroutineComplete);
}
```

Výpis kódu 6 Příklad iterátoru s callback parametrem

Alternativně lze, aby metoda `OnCoroutineComplete` byla také coroutine. Poté by metoda `CoroutineReturningAnInt` měla deklaraci `IEnumerator CoroutineReturningAnInt (Func<int, IEnumerator> onComplete)`, a metoda `onComplete` by byla volána jako coroutine (`yield return onComplete(value)`).

Coroutines si s sebou nesou výhodu, že je jejich život vázán na život komponenty na které běží. Díky tomu se není třeba starat, jestli se scéna na které jsou neukončuje, jestli objekt na kterém jsou závislé již neexistuje atp.

Deklarace metod jako coroutines jsem v návrhu a v diagramech nezobrazil, jelikož jejichž zobrazení vedlo na přehnaně rozsáhlé deklarace metod. (například z již tak rozsáhlé deklarace metody `public IEnumerable<Structure>? GetNearbyStructures(double latitude, double longitude, float range);` se stala metoda s deklarací `public IEnumerator GetNearbyStructures(Func<IEnumerable<Structure>?, IEnumerator> onComplete, double latitude, double longitude, float range);`, která pouze obsahuje předání metody pro předání výsledku, jenž je také coroutine)

Objekty scény

Unity scéna se skládá z objektů. Každý objekt má přiřazené komponenty, které dohromady určují chování objektu samotného (například automobil má jako komponent pozici na scéně, mesh potaženou texturou aby mělo vzhled, collider aby mohlo kolidovat se zbytkem scény, …)

4.2 Mapa a zobrazování HTML5 stránek

Prototyp původní aplikace byl navržen pro OS Android, což znamenalo dostupnost široké škály Android API. Android API obsahuje komponentu WebView, které se v původní aplikaci využívalo k zobrazování mapy, za pomoci Seznam Mapy API. Využití Seznam Map jsem se rozhodl ponechat, jelikož jsem obeznámen se základy jejich API.

Problém nastal se zobrazováním HTML5 (dále pouze HTML) stránek, což je potřeba také pro zobrazení mapy. Unity nepodporuje zobrazování HTML stránek. Jako řešení je možné využít nativních Android API, což ale přidá nemalou komplexitu a zamezí jednoduché přenositelnosti. Proto jsem se rozhodl využít knihovny [28] obalující nativní API na různých platformách, což mi umožní neřešit implementační detaily prohlížečů HTML stránek na jednotlivých platformách.

4.2.1 Javascript konvertor

Jako další problém se ukázalo volání Javascript kódu ze C# kódu. Je třeba předávat poměrně složité objekty, které například obsahují textové řetězce. U nich je nutno si pohlídat, jestli neobsahují uvozovky které by předčasně ukončily textový řetězec v Javascriptu. Další problém je například v číslech s plovoucí řádovou čárkou. Ty jsou při převodu na textový řetězec formátovány dle kultury (jazyka systému) uživatele, což vyústí ve využití desetinné tečky při anglické kultuře, ale desetinné čárky při české kultuře, kterou Javascript interpretuje jako oddělovač. Tyto problémy mají celkem jednoduché řešení, lze však snadno tento typ problému zapomenout vyřešit, a často se ukáže až za dlouhou chvíli.

Pro konvertování objektů do Javascriptu jsem si tedy raději navrhl a implementoval konvertor, který takovéto problémy řeší automaticky. Při návrhu jsem přemýšlel nad několika řešeními.

Návrhový vzor visitor (návštěvník) byl mým prvním návrhem. Každý objekt by tedy pouze přijímal návštěvníka a volal jemu příslušnou metodu návštěvníka, která by prováděla konverzi. To se však ukázalo jako neproveditelné, jelikož nelze přidat metody k existujícím „sealed“ (uzavřených, z nichž nelze dědit) objektům.

Návrhový vzor visitor s wrapper (obalovými) objekty je možné řešení předchozího problému. Pro objekty kde lze se využije návrhový vzor visitor. Pro objekty kde nelze návrhový vzor visitor využít, se vytvoří obal (wrapper) který bude přijímat návštěvníka, a implicitní konverze z/do wrapper objektu. Toto řešení jsem zavrhl z důvodu nemálo „boilerplate“ kódu, a zavedení mnoha implicitních převodů, které mohou v budoucnu přinést neočekávané chování.

Dynamické objekty umožňují v C# simulovat chování dynamicky typovaných jazyků. Důsledkem toho je jedna zajímavá vlastnost – double (dokonce multiple) dispatch bez nutnosti využití návrhového vzoru visitor. Stačí instanci objektu přiřadit do typu `dynamic`, zavolat vhodně přetíženou metodu a běhové prostředí jazyka pomocí magie[2] správně zvolí „nejlepší“ metodu kterou zavolá.

Unity IL2CPP však nepodporuje jmenný prostor `System.Reflection.Emit` [29] který je potřeba pro `dynamic` typy, proto tato možnost není přípustná.

Slovník typů objektů na adaptéry pro daný typ je další možné řešení. Slovník se naplní typy konvertovaných objektů a adaptéry k daným typům. Při potřebě konvertovat objekt se podívám do slovníku, a pokud najdu odpovídající typ, tak ho předám odpovídajícímu adaptéru.

Toto řešení má hlavní nevýhodu v tom, že jde proti polymorfismu. Při předání objektu do konvertoru je třeba objekt předávat jako typ `object?`, protože nelze objekt dynamicky přetypovat na typ, aniž bych ho znal v době překladu.

Nakonec jsem se rozhodl využít *Slovník typů objektů na adaptéry pro daný typ*, protože je proveditelná, a na rozdíl od *Návrhový vzor visitor s wrapper (obalovými) objekty* umožňuje rozšíření na prohledávání i nad-typů a implementovaných rozhraní.

O konvertování se stará třída `JSConverter`, což je specializace generické abstraktní třídy `Converter<T>`, která řeší vyhledávání správných adaptérů prohledáváním do šířky. Nejdříve zkusí

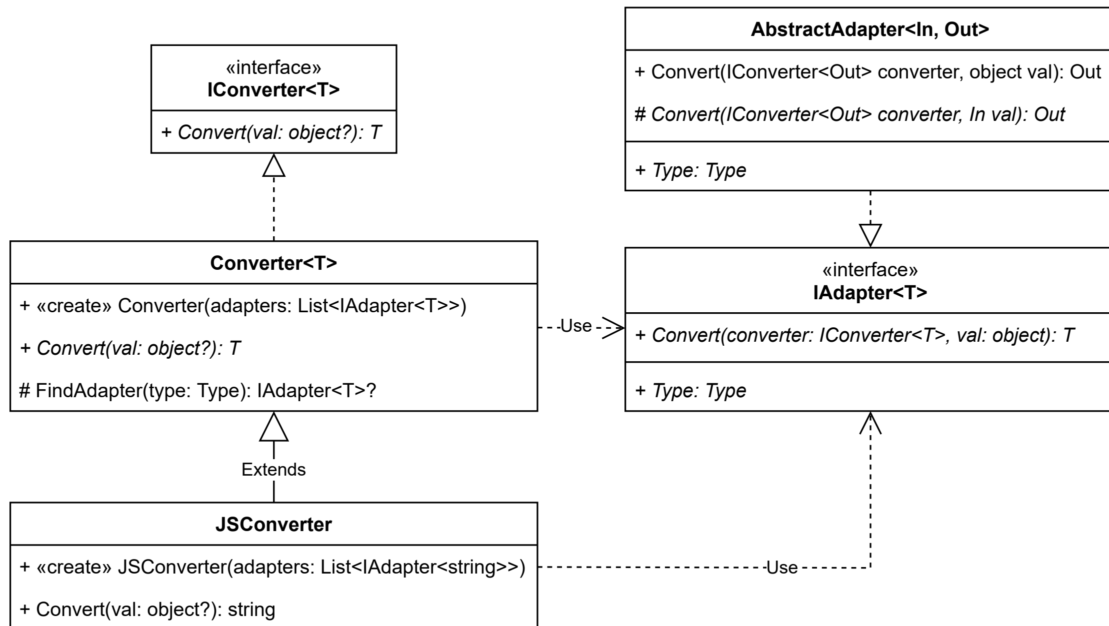
[2] https://en.wikipedia.org/wiki/Magic_(programming)

najít adaptér pro typ který dostal ke konvertování, pokud se mu to nepovede, tak zkusí nad-typ, a pokud se mu ani to nepovede, tak zkusí postupně všechny rozhraní které typ implementuje. Toto rekurzivně zkouší prohledáváním do šířky do té doby, dokud nedojde na konec hierarchie objektů (třídu object). To umožňuje v případě nedostupnosti adaptéru pro daný typ alespoň nějakou konverzi.

Tento systém by šlo rozšířit i o prohledávání implicitních, případně i explicitních operátorů konverze, nicméně si nemyslím, že je to odůvodnitelné, za cenu vyšší asymptotické složitosti (ač v tomto případě nejspíše zanedbatelná), a komplikovanějšího kódu (pro zjištění implicitních a explicitních operátorů je nutné použít reflexi).

Problém s předáváním typu object? který je třeba manuálně přetypovat jsem obešel přidáním adaptéru AbstractAdapter<In, Out>. Tento adaptér implementuje metodu T Convert (IConverter<T> converter, object val) která provádí přetypování, property System.Type Type a zanechává implementovat pouze metodu Out Convert(IConverter<Out> converter, In val), která jako parametr dostává již typově bezpečnou hodnotu (za předpokladu, že konverter vybere správný adaptér dle typu). Návrh lze vidět na obrázku 4.1.

Nabízí se otázka, jestli by nešlo AbstractAdapter<In, Out> vynechat úplně, a takto generické udělat samotné rozhraní IAdapter<T>. To ale nelze, protože na tento adaptér poté nelze použít Zenject dependency injection.



Obrázek 4.1 Diagram tříd Javascript konvertoru

4.2.1.1 IL2CPP

IL2CPP [30], spolu se starším Mono, je skriptovací backend kompilátoru v Unity. Tyto dva backends mají mezi sebou podstatné rozdíly.

Mono backend funguje na principu JIT (Just-In-Time) překladu, kdy se kód aplikace nejdříve přeloží do platformě nezávislého mezikódu nazývaného Common Intermediate Language (CIL), nebo dříve také Microsoft Intermediate Language. Spustitelný soubor aplikace poté obsahuje tento mezikód, a může ale nemusí obsahovat běhové prostředí a potřebné knihovny. Při spuštění aplikace se spustí běhové prostředí které si načte mezikód aplikace, a v době provádění programu si mezikód překládá do strojového kódu dané platformy. Na rozdíl od techniky interpretování

se ale mezikód překládá po blocích, díky čemuž například v cyklu není zpoždění při každém průchodu, ale pouze při prvotním přeložení.

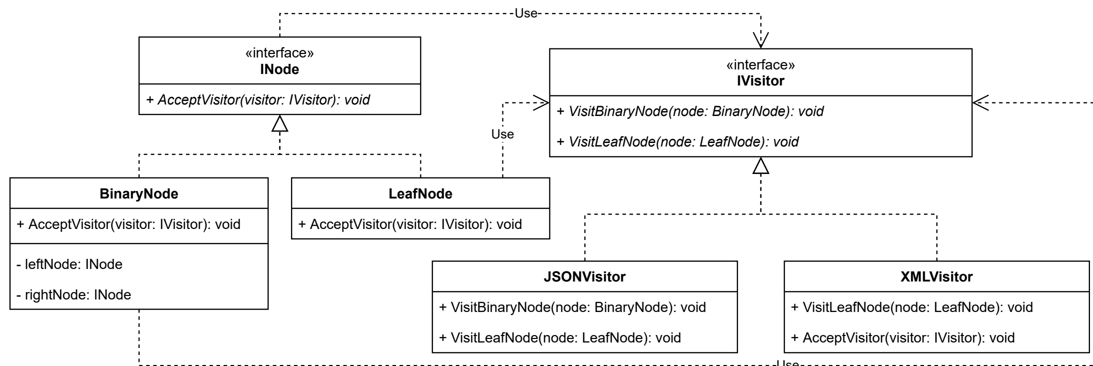
IL2CPP backend funguje na odlišném principu. Místo překládání JIT se mezikód překládá AOT (Ahead-Of-Time) do strojového kódu platformy. To teoreticky umožní rychlejší běh protože překladač není při provádění optimalizací tak časově omezen jako v době běhu, a může výstupní kód lépe optimalizovat. Tato výhoda je však vykoupena nevýhodami jako je například nutnost odlišných spustitelných souborů pro různé platformy (nebo jednoho pro více platforem, v tom případě je ale výstupní soubor větší), nedostupnost jmenného prostoru `System.Reflection.Emit` (a s tím spojená nedostupnost například již zmíněných `dynamic` typů), nebo problémy s virtuálními generickými metodami. IL2CPP je však jediná možnost, která je podporována na platformě iOS, proto se v rámci udržení budoucí přenositelnosti vynasnažím, aby překlad aplikace fungoval i pro tento backend.

4.2.1.2 Návrhový vzor visitor (návštěvník)

Návrhový vzor visitor umožňuje oddělit algoritmus od struktury, s jejíž daty pracuje v jazycích bez podpory double dispatch, pouze single dispatch. Cílem tohoto algoritmu je umožnit přidat implementaci bez změny datové struktury.

Mám například abstraktní rozhraní/třídu `INode`, které implementují třídy `BinaryNode` a `LeafNode` které reprezentují binární strom. Pokud chci implementovat výpis stromu ve formátu JSON, tak můžu v rozhraní `INode` vytvořit abstraktní metodu `ToJSON`, kterou budou třídy implementující `INode` implementovat. Pokud však poté chci implementovat výpis například ve formátu XML, tak musím přidat další abstraktní metodu, a tu poté v každé třídě znovu implementovat. To může být nevhodné v situaci, kdy implementovaný algoritmus nesouvisí s principem jedné odpovědnosti (single responsibility principle), nebo až nemožné v případě, kdy kód těchto tříd nepatří nám a třetí strana odmítne námi požadovanou implementaci přidat.

To lze vyřešit návrhovým vzorem visitor. Rozhraní `INode` bude mít virtuální metodu `AcceptVisitor(IVisitor visitor)` která bude přijímat instanci rozhraní `IVisitor`. Rozhraní `IVisitor` bude mít metodu pro třídy implementující `INode` (`VisitBinaryNode`, `VisitLeafNode`, `VisitAnotherSubclass`, …). Při zavolání `AcceptVisitor(IVisitor visitor)` se díky virtual (single) dispatch zavolá správná metoda dítěte `INode`, která pouze zavolá vhodnou metodu (například `VisitBinaryNode`) návštěvníka, a předá mu svoji instanci. Diagram tohoto návrhového vzoru spolu s ukázkou je na obrázku 4.2.



Obrázek 4.2 Ukázka využití návrhového vzoru visitor

4.3 Správa scén

V průběhu implementace jsem zjistil potřebu obrazovek/aktivit (pomocí Unity scén), kde by po otevření obrazovky uživatel mohl stisknout tlačítko zpět a vrátil by se na předchozí obrazovku („backstack“ operací). K tomuto účelu jsem si musel navrhnout třídu `SceneManager`, která vyhoví tomuto požadavku. Scény se otevírají dle jejich jména, třída umožňuje otevřít scénu exkluzivně nebo aditivně a v případě potřeby do scény vložit pomocí DI objekt `SceneData`, který pouze drží klíč-hodnota páry.

Pokud se scéna otevře aditivně, tak předchozí scéna/y zůstanou otevřené (například pro situace, kdy je potřeba zobrazit dialogové okno nad aktuální obrazovkou). Při stisknutí tlačítka zpět (nebo jiném způsobu zavření aktuální scény, zavoláním `CloseCurrentScene` ve `SceneManager`) se scéna poté pouze zavře.

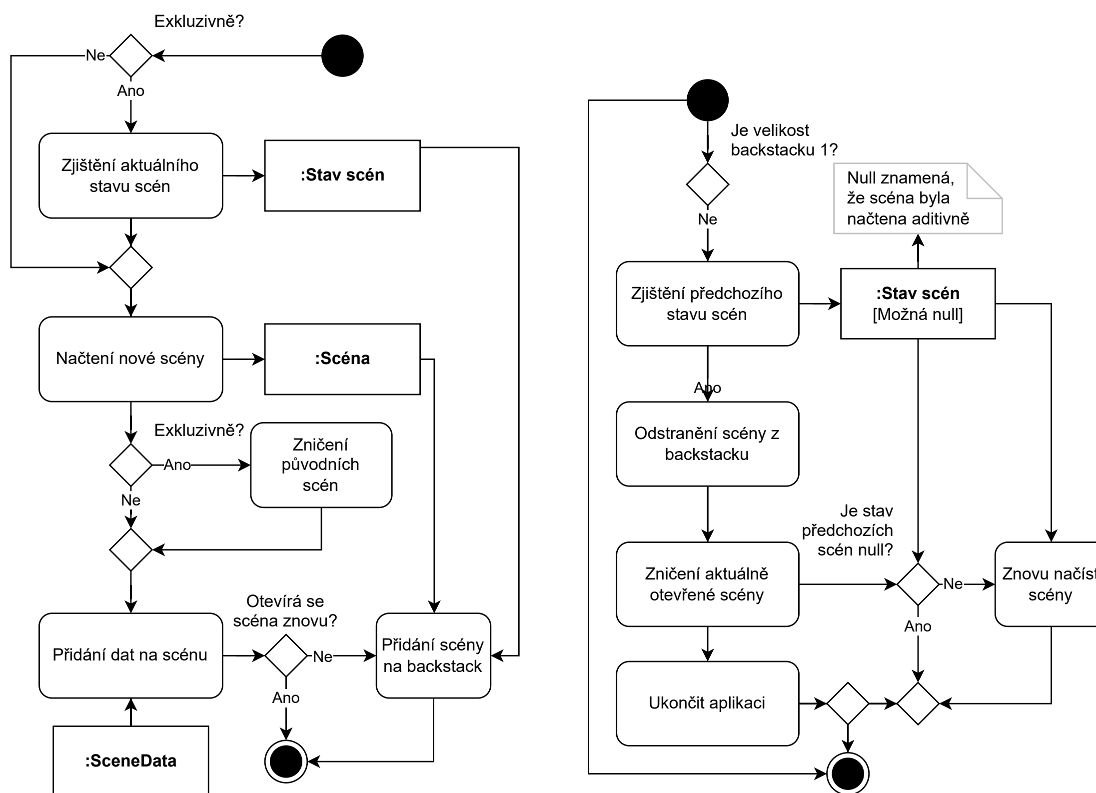
Pokud se scéna otevře exkluzivně, tak se uloží aktuální stav (jestli jsou načteny nebo nikoliv) scén, načte se nová scéna, všem ostatním scénám se uloží objekt `SceneData` a jsou zničeny. Při stisku tlačítka zpět se přečte předchozí stav scén, scény které byly načteny se načtou zpět, předá se jim zpátky `SceneData` a aktuální scéna se zničí.

Otevírání a zavírání scén je popsáno obrázkem 4.3. Zavírání scén obrázkem 4.3b, kde „Znovu načíst scény“ znamená pro každou předtím otevřenou scénu zavolat otevření scény dle obrázku 4.3a s parametrem že se scéna otevírá znovu.

Tento návrh a implementace nepočítá s možností „zachytit“ stisk tlačítka zpět a reagovat na něj, jelikož v době implementace tato funkcionalita nebyla potřeba. Například pokud si uživatel otevře informace o budově, kde by byl odkaz na jinou stránku, tak se v aktuální podobě nelze po kliknutí na odkaz a stisku tlačítka zpět místo zavření prohlížeče vrátit na původní stránku.

Mnou navrhované řešení jsou dvě. První spočívá v přidání možnosti registrace callback metody do `SceneManager`, volané při stisku tlačítka zpět. Nastavení callback metody by poté probíhalo z například `Start` metod komponent. To by však mohlo působit problémy v situacích, kdy se otevřou dvě scény se stejným jménem rychle po sobě. V takovém případě by se mohla stát situace, že se otevřou obě scény a až poté se registrují callback metody. Aktuální implementace `SceneManager` ale nedokáže odlišit jaká callback metoda patří k jaké scéně.

Druhé navrhované řešení je, po vzoru vložení `SceneData`, na scénu vložit objekt který by měl možnost registrace callback metod. Tento objekt by poté `SceneManager` volal při stisku tlačítka zpět. Pokud by nějaká komponenta na scéně chtěla „zachytit“ stisk tlačítka zpět, tak by si atributem `[Inject]` tento objekt „vyžádala“ a poté by do něj registrovala svou callback metodu.



(a) Ukázka otevírání scén **(b)** Ukázka zavírání scén

Obrázek 4.3 Diagram aktivit otevírání a zavírání scén

Z uchovávání objektů `SceneData` který jsem původně zamýšlel pouze jako způsob předávání parametrů scénám, vznikla možnost uchovávat data mezi scénami. To také umožňuje zjistit, jestli se scéna otevírá poprvé, nebo jestli se otevírá znova (po zavření jiné, exkluzivně otevřené scény).

Při implementaci tohoto řešení jsem se setkal s problémem, že Unity neumožňuje nemít otevřenou „žádnou“ scénu. Proto jsem, pro správnou inicializaci, musel vytvořit novou scénu `InitScene`, jejíž jediným účelem je existovat do doby, než ji nahradí jiná, plnohodnotná scéna. Dále jsem při zavírání scény musel vždy vytvořit novou, dočasnou scénu kterou jsem později v procesu zavírání scény zničil.

4.4 Nasvícení budov

V průběhu implementace jsem narazil na problém s plánovaným nasvícením budov, viz funkční požadavek F4. Nasvícení budov vyžaduje renderování světla a stínů v reálném čase, což je zbytečně výkonnostně náročné pro mobilní zařízení, proto od tohoto funkčního požadavku bylo, po dohodě s vedoucím práce, opuštěno.

4.5 Uživatelská příručka

Ukázky snímků obrazovek lze vidět na obrázku 4.4.

Spuštění aplikace: Uživatel spustí aplikaci a počká až se načte AR obrazovka. Pokud ještě tak neučinil, tak povolí potřebná oprávnění v dialogových oknech.

Zobrazení virtuální budovy: Po spuštění aplikace uživatel zamíří mobilním zařízením na místo kde byla historická budova a počká, než se budova zobrazí.

Zobrazení informací o virtuální budově: Uživatel počká na zobrazení virtuální budovy, klikne na zobrazenou virtuální budovu a počká, až se načte stránka s informacemi o budově.

Změna preferovaného roku virtuálních budov: Po spuštění aplikace uživatel otevře, tlačítkem kalendáře (v levém dolním rohu obrazovky), výběr preferovaného roku. Následně může rok vybrat buď přesunutím posuvníku v dolní části obrazovky, nebo manuálním zadáním preferovaného roku v pravém dolním rohu obrazovky.

Zobrazení mapy s pozicemi budov: Mapa s pozicemi budov se zobrazí po kliknutí na tlačítko mapy, dostupné v levém horním rohu obrazovky po spuštění aplikace.

Zobrazení zjednodušených informací o budově: Po spuštění mapy uživatel nalezne ikonu budovy, a klikne na ni.

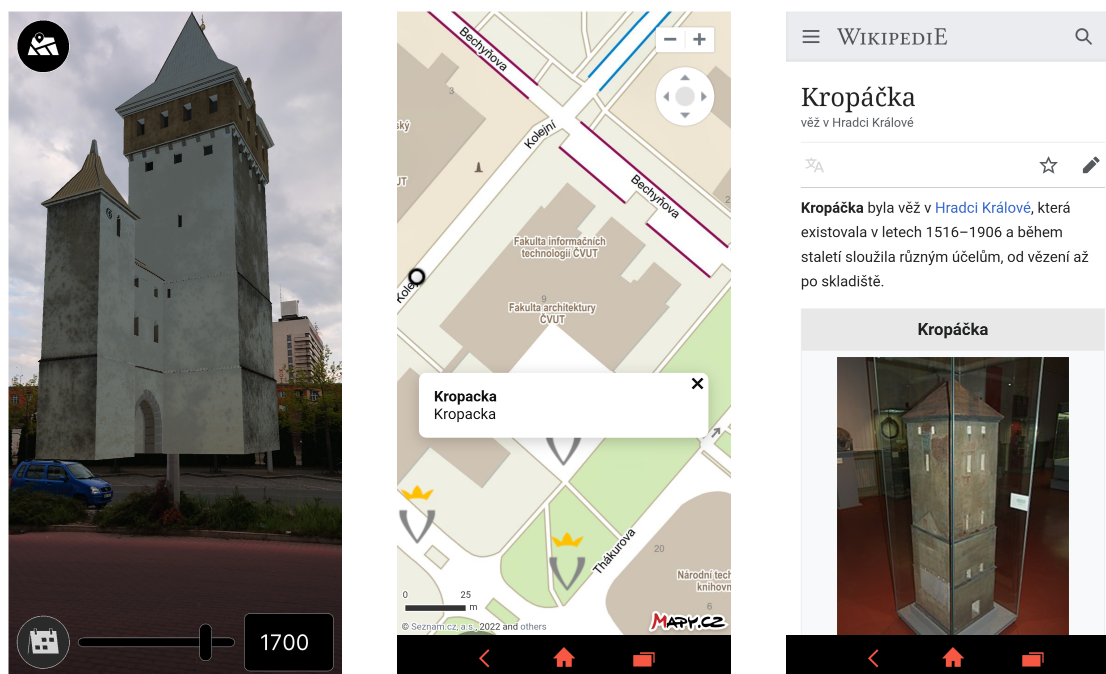
4.6 Instalační příručka

Instalaci lze provést dvěma způsoby, instalací přiloženého *.apk* souboru, nebo ze sestavení zdrojových kódů. Dále je také možné aplikaci spustit v omezené míře v Unity editoru na počítači. To se může hodit například pro návrh a testování UI, nebo komunikace s API.

4.6.1 Instalace přiloženého souboru

Pro instalaci přiloženého *.apk* souboru je třeba povolit v zařízení instalaci z neznámých zdrojů. Postup povolení instalací z neznámých zdrojů a celé instalace se liší pro různá zařízení, proto zde popíši postup na mém zařízení (Sony H8324).

Při instalaci je nejprve nutné zařízení připojit k počítači na kterém je *.apk* soubor aplikace. Po připojení je třeba v zařízení povolit v dialogu „přístup k datům v telefonu“, a z počítače zkopírovat *.apk* soubor do libovolného adresáře zařízení. Poté pomocí prohlížeče souborů v zařízení otevřít adresář do kterého byl *.apk* soubor nakopírován, a spustit ho. Objeví se dialogové okno s textem „Z bezpečnostních důvodů (…)“ a tlačítky *ZRUŠIT* a *NASTAVENÍ*. Po stisknutí tlačítka *NASTAVENÍ* je nutné v nové obrazovce povolit přepínač „Povolit z tohoto zdroje“. Následně se stačí vrátit tlačítkem zpět, a stisknout tlačítko *INSTALOVAT* v dialogovém okně.



(a) AR obrazovka s virtuální budovou a otevřeným výběrem preferovaného roku

(b) Mapa s pozicemi budov a zjednodušenými informacemi o budově (název, popis)

(c) Zobrazení informací o budově (Libovolná HTML5 stránka)

Obrázek 4.4 Snímky obrazovky aplikace

4.6.2 Instalace ze sestavení zdrojových kódů

Pro instalaci je třeba stáhnout a nainstalovat Unity Hub z [32] (v době psaní této práce verze 3.1.2, pro novější se může postup mírně lišit). Po stažení a nainstalování je třeba stáhnout samotné Unity. V záložce *Installs* je třeba stisknout tlačítko *Install Editor* a nainstalovat stejnou nebo vyšší (s tou ale mohou být problémy s kompatibilitou) verzi Unity než byla použita při vývoji. Verze která byla použita při vývoji je alfa verze, kterou lze nalézt v záložce *Pre-releases*.

V průběhu instalace, po stisknutí na tlačítka *Install*, je třeba zvolit *Android Build Support* se všemi pod-části (alternativně lze již nainstalované vynechat, je však poté potřeba správně nastavit jejich lokace). Pokud na cílovém zařízení není nainstalováno IDE pro vývoj C#, tak doporučuji nainstalovat i nabízené Visual Studio / Rider.

4.6.2.1 Přidání a spuštění projektu

Po nainstalování Unity je třeba naklonovat repozitář projektu, z `https://gitlab.fit.cvut.cz/vmck/android/vmck-mobile-client`, do zvoleného adresáře. Po naklonování repozitáře je třeba v Unity Hub v záložce *Projects* stisknout tlačítko *Open* a vybrat podadresář s projektem. Nakonec stačí vybrat projekt, který se objeví v nabídce níže, pokud se sám nespustí.

Po spuštění projektu je třeba ve *File* → *Build Settings* vybrat *Android* a stisknout tlačítko *Switch Platforms*. Poté je třeba vygenerovat streaming assets bundles, viz 4.7.3. Následně lze aplikaci sestavit a spustit na připojeném zařízení pomocí *File* → *Build And Run*.

Při instalaci na zařízení je třeba mít na zařízení aktivované *USB debugging*. Jeho aktivace se také liší pro různá zařízení, proto ho zde také nebudu popisovat.

4.7 Programátorská příručka

Aby bylo možné projekt spustit v editoru, je třeba otevřít scénu InitScene která se nachází v */Assets/Scenes*. Pro vývoj také doporučuji v okně *File* → *Build Settings* vybrat *Development Build*, *Script Debugging* a *Wait For Managed Debugger*. Tyto dvě možnosti umožní (lépe) ladit aplikaci na samotném zařízení.

4.7.1 Debugging

Pro testování věcí které nelze v editoru spustit (jako například AR) doporučuji aplikaci sestavit, nainstalovat na cílové zařízení a poté k aplikaci připojit debugger (ve Visual Studio *Debug* → *Attach Unity Debugger* a vybrat *AndroidPlayer (USB)*, i v případě mobilního telefonu připojeného přes TCP/IP). Existuje aplikace [33], ale ta pouze zrcadlí výstup z editoru do mobilního telefonu a posílá vstupy zpátky do editoru, což je většinou nedostačující a lze nahradit spuštěním přímo z editoru.

V době vývoje doporučuji nastavit skriptovací backend na *Mono* (*Edit* → *Project Settings* → *Player* → *Android settings* → *Other Settings* → *Scripting Backend*), z mých zkušeností má rychlejší sestavení (řád desítek sekund oproti jednotek minut) a rychlejší běh v debug verzi. Nicméně pořád doporučuji co nejvíce využívat spuštění v editoru, takto spustit projekt trvá nízké jednotky sekund.

4.7.1.1 Připojení Android zařízení přes TCP/IP

Při ladění se příležitostně hodí nebýt limitován délkou USB kabelu. Ladění Android zařízení lze provést i po TCP/IP. To znamená, že pokud je přes například Wi-Fi připojené do stejné lokální sítě, po provedení následujícího postupu ho již není potřeba mít připojené přes USB:

Zařízení je třeba mít ze začátku připojené přes USB, s povoleným *USB debugging*. Poté stačí spustit terminál s *adb* na cestě a zadat adb tcpip <port> pro přepnutí zařízení do TCP/IP módu. Následně je možné zařízení odpojit, a zadat do terminálu adb connect <ip_adresa_zařízení>:<port>.

4.7.2 Struktura projektu

Projekt se skládá ze scén. Každá scéna (uložená ve Scenes) reprezentuje jednu „obrazovku“. Scéna má v sobě uložené své objekty a většinou objekt s DI scene installer skriptem. Ty využívají installers z Installers pro nainstalování stejné funkcionality do více scén (například lokačních modulů).

Soubory související s UI (Unity obdoba HTML a CSS – UXML a USS) jsou umístěné v adresáři a podadresářích UI Toolkit kde se nachází deklarace UI, a adresáři UI, kde se nachází samotná logika UI. Nicméně mapa je vytvořena pomocí technologie HTML5 a nachází se v samostatném adresáři map, což je adresář ze které se vytváří bundle, viz 4.7.3. Skripty pro JavaScript konvertor který mapa využívá jsou v adresáři JavascriptConvert. Správa modelů včetně šipek (které nejsou řešeny pomocí UI, ale jako objekty scény) se nachází v podadresáři Structures.

Ve složce API se nachází část aplikace komunikující s API, včetně interní části která provádí samotnou komunikaci (ve složce Internal) a samotných tříd do kterých se deserializují JSON objekty API (ve složce Json).

Adresář AR obsahuje skripty související s ukotvením a korekcí ukotvení. Ty využívají lokační moduly, jež se, spolu s pomocnými soubory, nacházejí ve složce Providers, respektive v Location.

Adresář XR obsahuje konfigurační soubory AR, které obsahují konfiguraci nastavenou v Unity a nedoporučuji je upravovat ručně.

```
Assets
├── Bundles ........................................ streaming asset bundles
│   └── map
│       └── index.html ................................... HTML5 soubor mapy
├── Editor
│   └── BundleBuilder.cs .......... skript pro generování streaming asset bundles
├── OBJImport ................................... knihovna Runtime OBJ Importer
├── Resources ........................................ materiály, textury, atp
├── Scenes ..................................................... soubory scén
├── Scripts ................................................ zdrojové soubory
│   ├── API ........................ komunikace s API, deserializace API objektů
│   ├── AR .......................................... ukotvení, korekce ukotvení
│   ├── Common ........................... kontejnery, třída s extension metodami
│   ├── Installers ................................................. DI installers
│   │   └── Scenes ............................................ DI installers scén
│   ├── JavascriptConvert ..................................... JavaScript konvertor
│   │   └── Adapters ........................... adaptéry JavaScript konvertoru
│   ├── Location ......................................... pomocné soubory lokace
│   │   └── Providers ............................................ lokační moduly
│   ├── Logging ..................................................... logování
│   ├── SceneSwitching .............................................. správa scén
│   ├── Structures .............................................. správa struktur
│   └── UI .......................................................... logika UI
├── StreamingAssets ..................... vygenerované streaming assets bundles
├── UI Toolkit ........................................ deklarační soubory UI
└── XR ............................................... konfigurační soubory AR
```

Obrázek 4.5 Důležité adresáře/soubory projektu

4.7.3 Bundles

Při změně HTML souboru mapy je třeba znovu zabalit/vygenerovat „streaming assets bundles“. O to se stará skript `BundleBuilder.cs`, který lze spustit v Unity kliknutím pravým tlačítkem myši do okna se soubory projektu a zvolením *Build Bundles*. To je také třeba udělat při změně platformy. Tento postup vytvoří/přepíše bundles ve složce `StreamingAssets`. Co vše se do výsledného souboru zabalí určuje štítek *AssetBundle* který lze nastavit souborům v Unity, v inspektoru.

4.7.4 Knihovna Runtime OBJ Importer

Knihovna [34] umístěná ve složce `OBJImport` se využívá k načítání *.obj* souborů po stažení z API. Tako knihovna je však lehce poupravena. V souboru `MTLLoader` je změněn shader z „Standard (Specular setup)“ na „Standard“. To samé je provedeno v souboru `OBJLoaderHelper` V souboru `CharWordReader` je do metody `ReadInt` přidán kód který přeskočí znaménka „+“, stejný jako pro znaménko „-“.

Dále je v souboru `OBJLoader` vytvořen konstruktor přijímající funkci která vrací instanci `MTLLoader`, konstruktor bez parametrů který tuto funkci nastavuje na funkci vracející novou instanci `MTLLoader`, což je podtřída `MTLLoader`. Každé nové vytvoření instance třídy `MTLLoader` v tomto souboru je nahrazeno zavoláním funkce vracející `MTLLoader`. Tato modifikace je kvůli potřebě přepsat metodu `TextureLoadFunction` třídy `MTLLoader` a tuto třídu poté předat třídě

`OBJLoader`. Opraven je v tomto souboru také nekonečný cyklus v případě špatných dat, pokud již byly všechny data přečteny. Další část je oprava pokud má *.obj* soubor pouze negativní indexy.

Tato knihovna načítá modely na hlavním vlákně, má nemálo chyb, a samotný OBJ formát není příliš vhodný (je textový což znamená delší dobu načítání; nepodporuje animace). Proto bych navrhoval podporu tohoto formátu z aplikace odstranit, ve prospěch formátu glTF, který aplikace zvládá v omezené podobě. Načítání formátu glTF aktuálně vyžaduje, aby *.gltf* soubor měl všechny data včetně textur v sobě. To zjednodušuje manipulaci se souborem, avšak nelze využít plného potenciálu mezipaměti (změna jedné textury vyžaduje znovu stažení a uložení celého modelu, místo jedné textury). V době implementace však v API nebyl jediný model v tomto formátu který měl oddělené textury, proto jsem tuto funkcionalitu nemohl implementovat.

Kapitola 5

Testování

V této kapitole navrhnu testovací scénář pro uživatelské testování, které budou využity pro otestování, že výsledná aplikace je jednoduchá a uživatelsky přívětivá.

5.1 Scénář testování

Testovací scénář byl inspirovaný prací [17], ale přepsán pro specifikace této aplikace. Zadavatel řekne uživateli, aby v průběhu interakce s aplikací popisoval své myšlenky a ty si bude zapisovat/nahrávat. Výsledný záznam uživatele bude sloužit jako „návod“, jak uživatelé s aplikací interagují. Scénář a instrukce, které řekne uživateli jsou následovné:

Nacházíte se v nějakém z věnných měst. Na informační tabuli jste věděl/a „reklamu“ na aplikaci ve které si budete moci prohlédnout historickou budovu, která stávala na tomto místě, v rozšířené realitě. Aplikaci si stáhnete a spustíte.[1]

1. Zobrazte a prohlédněte si historickou budovu na místě, kde stojíte.
2. Pokud se ve vašem okolí žádná budova nenachází, tak si otevřete mapu, na které si vyberte budovu v okolí a přesuňte se k ní. Pokud se ve vašem okolí budova nachází, tak se pouze podívejte, kde se nachází ostatní historické budovy.
3. Dále se na mapě podívejte na název budovy ve *<jiné než aktuální věnné město>*.
4. Zobrazte si (znovu) historickou budovu a následně si zobrazte více informací o ní.
5. Zkuste si při zobrazení historické budovy změnit preferovaný rok, a dívejte se, jak se budova měnila v čase.

5.2 Výsledky testování

Z důvodu nekompletní implementace API v době psaní práce a nedostupnosti modelů budov v témže API jsem nemohl provést uživatelské testování. Po dohodě s vedoucím práce testování provedu před obhajobou závěrečné práce.

[1] V [17] bylo uživateli vysvětleno co je to AR, to zde ale záměrně neprovedu.

Závěr

Cílem této práce bylo vytvořit dlouhodobě udržovatelnou, jednoduše rozšířitelnou a uživatelsky přívětivou mobilní aplikaci pro OS Android, která by měla být také jednoduše přenositelná.

V práci jsem provedl analýzu stávajícího řešení, během které jsem zjistil, že pro budoucí rozvoj aplikace není možné v aktuální implementaci pokračovat. Dále jsem analyzoval stav API pro získávání modelů budov. Po zjištění stavu API jsem navrhl změny, které byly třeba udělat, aby bylo možné realizovat implementaci této aplikace.

Při návrhu jsem se také více zaměřil na moduly pro získávání lokace a jejich rozhraní navrhl znovu, aby bylo možné využívat více modulů najednou. S tím také souvisí nekompatibilita s původními moduly, nicméně ta převážně pramení z jiných použitých technologií. Dále jsem navrhl rozhraní pro získávání dat z API tak, aby nebylo tolik závislé na implementované sadě funkcí co API poskytuje (například pokud by API neposkytovalo pozici všech budov, potřebnou pro mapu) a aby bylo i případně možné REST API nahradit za úplně jinou technologii.

V průběhu implementace jsem potkal několik problémů které jsem musel vyřešit: konverzi C# objektů do textové reprezentace potřebné pro JavaScript, musel jsem zvážit a rozhodnout se mezi dvěma způsoby asynchronního programování a navrhnout způsob řízení scén pro uživatelské rozhraní.

Jelikož v době implementace nebyly v API potřebná data a nebylo implementované vyhledávání dle parametrů, nemohl jsem v době psaní práce provést uživatelské testování.

Možnosti dalšího rozšíření/úprav

Integrace existujícího OpenCV modulu (za pomoci [35], instance `ARCameraManager` je instalována na AR scénu, stačí si ji pomocí DI nechat injektovat) by měla být dalším krokem ve vývoji aplikace, což by umožnilo zlepšit přesnost získávané pozice. To by také umožnilo umisťovat modely budov vertikálně také dle pozice, což aktuálně není možné kvůli velké nepřesnosti.

Řazení modulů dle momentální přesnosti by mohl být způsob, jak využít modulů k získání lokace větší přesnosti.

AR Point Clouds [36] by šly využít pro zlepšení odhadu lokace.

Depth API [37] by bylo vhodné zapnout, ale bylo by třeba kolem budov umístit nějaký box ve kterém by se tato technika neaplikovala. Aktuálně může být historická budova v místě jiné budovy, která by „pohltila“ virtuální budovu, což nechceme. Pořád je ale žádané, aby virtuální budova byla například za reálnou kašnou.

Literatura

[1] BAUM, L.F. *The Master Key: An Electrical Fairy Tale Founded Upon the Mysteries of Electricity and the Optimism of Its Devotees. It was Written for Boys, But Others May Read it*. Indianapolis : Bowen-Merrill Co., 1901. 297. LCCN 01024123.

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

[3] KŘÍŽ, P. *Urban scene recognition and editing II*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[4] SLABÝ, O. *Věnná města českých královen - Jádro mobilního klienta*. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

[5] SIVÁK, D. *Věnná města českých královen – Backend administrační části*. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[6] *Věnná města českých královen* [online]. Univerzita Hradec Králové. [cit. 23. 04. 2022]. Dostupné z: https://www.kralovskavennamesta.cz/

[7] ZAJÍC, M. *Věnná města českých královen I. – úprava textur*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

[8] TISLICKÝ, J. *Věnná města českých královen II. – úprava textur*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

[9] ANTOŠ, P. *Věnná města českých královen – Webová aplikace pro schvalovací proces*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[10] PÚČALA, M. *Dowry Towns of Bohemian Queens – web-based 3D model viewer*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[11] MATOUŠKOVÁ, K. *Věnná města českých královen – Interakce v historickém městě ve virtuální realitě*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[12] *Oxford English and Spanish Dictionary, Synonyms, and Spanish to English Translator* [online]. Oxford. [cit. 23. 04. 2022]. Dostupné z: https://www.lexico.com/definition/augmented_reality/

[13] *Sceneform SDK for Android* [online]. Google. [cit. 23. 04. 2022]. Dostupné z: https://github.com/google-ar/sceneform-android-sdk/

[14] *Sceneform Maintained SDK for Android* [online]. SceneView. [cit. 23. 04. 2022]. Dostupné z: https://github.com/SceneView/sceneform-android/

[15] *SceneView Android* [online]. SceneView. [cit. 23. 04. 2022]. Dostupné z: https://github.com/SceneView/sceneview-android/

[16] *Dependency injection with Hilt* [online]. Hilt and Dagger. Google. [cit. 23. 04. 2022]. Dostupné z: https://developer.android.com/training/dependency-injection/hilt-android/#hilt-and-dagger

[17] BRHLÍK, L. *Věnná města českých královen - Uživatelské rozhraní mobilního klienta*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

[18] VANČURA, D. *Věnná města českých královen - jádro*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

[19] *Unreal Engine 5 Documentation* [online]. [cit. 23. 04. 2022]. Dostupné z: https://docs.unrealengine.com/5.0/en-US/

[20] *Unity 2022.2.0a10 C# reference source code* [online]. [cit. 23. 04. 2022]. Dostupné z: https://github.com/Unity-Technologies/UnityCsReference/

[21] *[WIP] Adding ARCore* [online]. [cit. 23. 04. 2022]. Dostupné z: https://github.com/godotengine/godot/pull/26221/

[22] *C# basics*. [cit. 23. 04. 2022]. Dostupné z: https://docs.godotengine.org/en/stable/tutorials/scripting/c_sharp/c_sharp_basics.html

[23] *WebXR Device API* [online]. [cit. 23. 04. 2022]. Dostupné z: https://www.w3.org/TR/webxr/

[24] MORTONSON Ch. Unite 2016 - Unity Architecture in Pokémon Go. [video]. *YouTube* [online]. Unity, 2016. Dostupné z: https://www.youtube.com/watch?v=8hru629dkRY/

[25] *Nullable reference types (C# reference)* [online]. [cit. 23. 04. 2022]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-reference-types/

[26] *Customize your build* [online]. [cit. 23. 04. 2022]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/msbuild/customize-your-build/

[27] *C# Compiler Options to report errors and warnings* [online]. [cit. 23. 04. 2022]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/compiler-options/errors-warnings/

[28] *unity-webview* [online]. [cit. 23. 04. 2022]. Dostupné z: https://github.com/gree/unity-webview/

[29] *Scripting restrictions* [online]. [cit. 23. 04. 2022]. Dostupné z: https://docs.unity3d.com/Manual/ScriptingRestrictions.html

[30] *An introduction to IL2CPP internals* [online]. [cit. 23. 04. 2022]. Dostupné z: https://blog.unity.com/technology/an-introduction-to-ilcpp-internals/

[31] *IEnumerator Interface* [online]. [cit. 23. 04. 2022]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerator/

[32] *Powerful 2D, 3D, VR, & AR software for cross-platform development of games and mobile apps.* [online]. [cit. 02. 05. 2022]. Dostupné z: https://store.unity.com/#plans-individual

[33] *Unity Remote 5* [online]. [cit. 02. 05. 2022]. Dostupné z: https://play.google.com/store/apps/details?id=com.unity3d.mobileremote

[34] *Runtime OBJ Importer* [online]. [cit. 05. 05. 2022]. Dostupné z: https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547

[35] *Accessing the device camera image on the CPU* [online]. [cit. 07. 05. 2022]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/cpu-camera-image.html

[36] *AR Point Cloud Manager* [online]. [cit. 24. 04. 2022]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/point-cloud-manager.html

[37] *Depth adds realism* [online]. [cit. 24. 04. 2022]. Dostupné z: https://developers.google.com/ar/develop/depth/

Obsah přiloženého média

```
 impl
 ├── src ......................................... zdrojové kódy implementace
 └── out.apk ................................. spustitelný soubor implementace
 thesis
 ├── src ............................... zdrojová forma práce ve formátu LaTeX
 └── thesis.pdf ................................... text práce ve formátu PDF
```