FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Věnná města českých královen - jádro
Student:	Daniel Vančura
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

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

1. Analyzujte možná řešení datového úložiště pro grafické objekty (model, textury).
2. Analyzujte funkční a nefunkční požadavky kladené na jádro zadavatelem a ostatními členy realizačního týmu. Zaměřte se především na problematiku komunikačních API.
3. Dle předchozích požadavků navrhněte datové úložiště a příslušející prototypy API, která umožní komunikaci mezi datovým úložištěm a ostatními moduly (moduly blender, mobil, navigace, debug, atd.).
4. Implementujte datové úložiště s ohledem na možnou výkonovou škálovatelnost.
5. Implementujte prototypy REST API rozhraní za použití technologie Node.js, využijte technologie continuous integration.
6. Prototypy rozhraní podrobte automatizovaným a integračním testům.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 20. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Věnná města českých královen - jádro

Daniel Vančura

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Chludil

30. července 2020

Poděkování

V první řadě bych chtěl poděkovat svému vedoucímu práce Ing. Jiřímu Chludilovi za to, že se mnou takovou dobu vydržel, a neztratil příliš trpělivost ani po mých neustálých dotazech.

Dále bych rád poděkoval své rodině za podporu během celé doby studia a svým hlučným sourozencům za neustálé vyrušování při psaní textu této práce.

A v neposlední řadě děkuji své posteli a své přítelkyni za to, že mě neopustily, i když jsem se jim obzvláště v posledních týdnech tvorby této práce neměl téměř vůbec čas věnovat.

Prohlášení

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

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

V Praze dne 30. července 2020 ……

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Vančura, Daniel. *Věnná města českých královen - jádro*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Ve své práci se věnuji analýze a návrhu nové verze stěžejní části projektu *Věnná města českých královen*, tj. datovému úložišti a k němu příslušícím komunikačním API. Jedním z konečných cílů projektu je tvorba a zobrazení historické podoby českých věnných měst v rozšířené realitě.

Tato práce propojuje množství již dříve na toto téma vytvořených modulů a umožňuje jejich vzájemnou komunikaci. Touto prací navazuji na dříve vytvořené serverové části projektu a provádím jejich rozsáhlou optimalizaci.

V rámci práce je provedena analýza předchozích řešení, označeny jejich největší nedostatky a následně je uveden návrh na jejich odstranění. Obsahem práce je i navržení nových procesů nad API, které mají za účel zvětšovat historickou věrohodnost daného 3D modelu budovy. Následně je obsahem práce i tvorba dokumentace nově navrženého API a ta je spolu se zdrojovými kódy uvedena na přiloženém médiu.

Klíčová slova Věnná města českých královen, API, Typescript, TypeORM, Node.js, databáze, PostgreSQL, MongoDB, Express

Abstract

In this thesis I'm mainly focusing on analysis and further developement of the core parts of the project *Dowry Towns of the Queens of Bohemia*, that being it's data storage and communication APIs. One of the final goals of the project is the creation and later display of the historical image of said towns in augmented reality.

This thesis connects multiple other works created in the project before and enables their mutual communication. I'm building on the previously made server-side modules of said project and I'm performing their massive optimalization.

There's a complete analysis of previous solutions of said DB and APIs in this thesis and later there're reccomendations regarding their improvement and removal of their disadvantages. Also a few proposals were made regarding new processes in the APIs, which should be improving the historical credibility of used 3D models. In the later parts of the thesis there's a documentation of the proposed API made and together with the API source code these are present on the attached medium.

Keywords Dowry Towns of the Queens of Bohemia, API, Typescript, TypeORM, Node.js, database, PostgreSQL, MongoDB, Express

Obsah

Úvod **1**

1 Cíle práce **3**

2 Analýza projektu **5**
2.1 Projekt Věnná města českých královen 5
2.2 Rozbor jednotlivých částí (modulů) 6
2.3 Data nutná pro VMČK 11
2.3.1 Zobrazení měst za různých okolností 11
2.3.2 Vývoj ukládaných dat 12
2.4 Stav API a databáze při převzetí vývoje 13
2.5 Použité technologie 16

3 Rozbor zadání **19**
3.1 Hlavní nedostatky předchozího návrhu 19
3.1.1 Schéma databáze 19
3.1.2 Struktura a implementace API 23
3.2 Funkční a nefunkční požadavky 25
3.2.1 Funkční požadavky 25
3.2.2 Nefunkční požadavky 26

4 Návrh řešení **27**
4.1 Databáze metadat 27
4.1.1 Schvalovací proces 30
4.2 MongoDB 34
4.3 Privátní API 34
4.3.1 Tagování 3D objektů 36
4.3.2 Verzování 36
4.3.3 Používané formáty souborů 37

4.4 Vize do budoucna 37
4.4.1 Management uživatelů a přístupových práv 38
4.4.2 Transformace 3D objektů generované na serveru 38
4.4.3 Webové rozhraní pro schvalovací proces 38
4.4.4 Uložení a zobrazení terénu 39
4.4.5 Databáze lokalizačních informací 39
4.4.6 Load balancing a synchronizace databází 39
4.4.7 Vyhledávání 40
4.4.8 Public API 40
4.4.8.1 „Feedback API“ 40
4.4.8.2 Navigace 40
4.4.9 Optimalizace mobilních zařízení 40

5 Realizace **43**
5.1 Instalační příručka 43
5.1.1 Převzetí vývoje 43
5.1.2 Testování 44
5.1.3 Continuous integration 44
5.1.4 Nasazení API na server 45
5.2 Příručka pro vývojáře 45
5.3 Příručka administrátora 49

Závěr **53**

Literatura **55**

A Seznam použitých zkratek **59**

B Obsah přiloženého Média **61**

Seznam obrázků

2.1 Schéma projektu na konci týmových projektů 7
2.2 Big picture - rozsah této BP 10
2.3 Schéma datového úložiště - BP Víta Černého 12
2.4 Schéma DB na konci týmových projektů 14
2.5 Swagger schéma API Bc. Jindřicha Mácy 15
2.6 Struktura GISu 18

3.1 Schéma DB při převzetí - pro přehlednost textu znovu 20
3.2 Swagger schéma API - pro přehlednost textu znovu 24

4.1 Návrh schématu DB 28
4.2 Schvalovací proces modelu I. 31
4.3 Schvalovací proces modelu II. 32
4.4 Schvalovací proces varianty 33
4.5 Swagger dokumentace Private API 35

5.1 Seznam podporovaných formátů 3D objektů 47

Úvod

Virtuální i rozšířená realita (dále VR a AR) se obzvláště v posledních letech velmi rozmohla. Firma BMW používá AR k prohlédnutí vozu před jeho objednáním (aplikace BMW iVisualizer), *ProjectDR*, umožňující lékařům zobrazení snímků přímo na pacientově těle pomocí rozšířené reality, je momentálně ve fázi testování, projekty jako *ClassVR* míří k integraci virtuální i rozšířené reality do školství a další projekty jsou vymýšleny a vytvářeny téměř denně.

V ČR během let vzniklo například několik podnětných projektů a prací zabývajících se digitalizací městské infrastruktury či její historické podoby ([1], Virtuální stará Praha [2, 3, 4], Visit.More…). Jako podobně tématicky zaměřený projekt (podpora vizualizace české historie) vznikla Věnná města českých královen (dále jen VMČK či Věnná Města)[5]. Tento projekt se nezabývá myšlenkou univerzálního zobrazení libovolných dat v příslušných realitách, nýbrž kromě prezentace vzniklých modelů měst či jiných historických artefaktů také optimalizací procesu jejich tvorby. Oproti dříve zmíněným tedy není naším cílem vytvořit jednu aplikaci zobrazující daná města, ale celou infrastrukturu vedoucí k tomuto výsledku.

Realizace tohoto projektu je přibližně ve své polovině, kde velká část dosavadních prací byla spíše rešeršního charakteru a věnovala se sběru dat. Další práce, obzvláště na FIT ČVUT, se pak věnovaly rešerši a vývoji navržených prototypů, službám vyžadovaným v serverové části projektu a případnému datovému úložišti. Během těchto prací vzniklo v projektu mnoho nových funkcionalit a směrů, se kterými se původně nepočítalo, ale které si zaslouží být prozkoumány a implementovány, výsledkem čehož bylo hned několik závěrečných prací nad původní rámec Věnných Měst.

V rámci zkoumání možností dalšího rozvoje projektu pak vznikají práce věnující se například přesnějšímu určení polohy, než jaké nabízí GPS, navigaci po daném městě, automatizované tvorbě textur a podobně. Tyto práce se však prozatím zaměřovaly na jednotlivé funkcionality, spíše než na jejich propojení (zásuvný modul pro Blender [6], navigační modul[7] a další). Dosud bylo vy-

tvořeno několik různých verzí serverové části [8, 9, 10, 11, 12], která propojuje datové úložiště a ostatní moduly. Při neustálém vývoji projektu však vznikla potřeba vytvoření její nové a optimálnější verze. Tato část mi byla nabídnuta jako závěrečná práce a volné pokračování předmětu *Softwarový týmový projekt*, a protože česká historie mě vždy zajímala a projekt mě zaujal, tuto nabídku jsem přijal.

V rámci Věnných Měst jsem ve velké míře spolupracoval hned s několika studenty. Největší zásluhy na návrhu současné podoby API (Application Programming Interface) mají kolegové Bc. Jindřich Máca, Ing. Michal Martinek a Martin Púčala, z předchozích let pak kolegové Bc. Martin Čapek, Ing. Jaroslav Štěpán a Ing. Ondřej Slabý. Ve své práci zmíním naše společné body, konkrétní požadavky kladené na mou práci a můj návrh jejich řešení.

V sekci 2 představím projekt Věnná města Českých královen, jeho schéma, celkovou architekturu a jeho současný stav. Dále představím technologie pro tuto práci klíčové (Postgres, MongoDB, Typescript a další). Ve stejné části práce se budu věnovat jednotlivým požadavkům na úpravu datového úložiště a příslušného API.

V sekci 4 se budu věnovat vytváření modelů a schémat datového úložiště, diagramům vysvětlujícím myšlenky a jednotlivé ideje stojící za Věnnými městy, návrhu samotných API a systému verzování nad datovým úložištěm.

Nakonec v sekci 5 uvedu příručky pro další pracovníky na projektu a návody k použití již vytvořených částí.

Kapitola 1

Cíle práce

Hlavním cílem mé bakalářské práce je přepracovat datové úložiště vzniklé během předmětu *Softwarový týmový projekt* (dále jen týmový projekt) a implementovat k němu API rozhraní komunikující se zbylými částmi projektu. Součástí této práce je i návrh a implementace verzovacího systému nad datovým úložištěm za účelem tvorby jednotlivých grafických objektů a usnadnění hledání chyb.

Výběr samotných technologií cílem mé bakalářské práce není, neboť technologie datového úložiště i API byly zadány vedoucím práce s ohledem na funkční a nefunkční požadavky projektu jako celku a určeny předchozím vývojem.

Jelikož API rozhraní bylo již částečně implementováno během práce [12] na základě požadavků zadavatele práce, cílem této práce je i převzít a dokončit implementaci daného API. Konkrétní podoba se ale bude ve velké míře lišit, a to z důvodů uvedených v kapitole 3.1.

Můj malý osobní cíl je vytvoření souhrnného dokumentu, který bude sloužit k seznámení se studentů dalších ročníků s projektem VMČK. Jedním z hlavních problémů, se kterým jsem se setkal během práce na týmovém projektu, byla právě neexistence čehokoliv podobného, což mělo za následek zdržení práce o několik týdnů, které byly věnovány čistě vysvětlení myšlenek projektu. Vytvořením této bakalářské práce si dávám za cíl alespoň částečné odstranění tohoto problému, a také to, že až se projekt posune do budoucích fází natolik, že tato práce nebude pro vysvětlení stačit, tak se najde student, který se tímto inspiruje a udělá totéž, ať už ve formě své závěrečné práce, či něčeho jako *VMČK wiki*.

Kapitola 2

Analýza projektu

V této kapitole se budu věnovat především představení projektu, kontextu této práce, a jednotlivým technologiím použitým ve VMČK. Nejprve zde rozeberu projekt obecně, jeho rozdělení na jednotlivé moduly (tzv. big picture), poté krátce jeho historii, myšlenky stojící za jeho konkrétní podobou, a poté stav projektu před touto prací. Jelikož součástí mé práce není výběr jednotlivých technologií, uvedu na konci této kapitoly pouze jejich seznam, krátký popis a účel, nikoliv porovnání s ostatními. Na základě myšlenek projektu, jeho požadovaných funkcionalit a stavu při převzetí vývoje, pak ještě vytvořím seznam funkčních a nefunkčních požadavků kladených na tuto práci.

2.1 Projekt Věnná města českých královen

Jedním z cílů projektu je vytvořit komplex spolupracujících aplikací za účelem virtuální prohlídky, tvorby a prezentace historie Věnných měst Českých královen.

„Projekt se zaměřuje na prezentaci tohoto kusu historie široké veřejnosti pomocí nástrojů historické geografie a počítačové grafiky. Hlavním výstupem tohoto snažení má být mobilní aplikace a informační portál s historickým průvodcem včetně vyhotovených zrekonstruovaných 3D modelů budov a dalších artefaktů. Mobilní aplikace by měla nabídnout jedinečný zážitek pomocí rozšířené reality, kdy bude možné si prohlédnout podobu města napříč časem přímo na místě.“ [13]

V pozdějších letech se očekává rozšíření působnosti projektu z pole historické geografie i na obecně jakékoliv artefakty hmotného historického dědictví. Samotné památky (knihy a podobně) mohou pak být i nadále bezpečně uloženy, ale zároveň zkoumány ve virtuálním prostoru bez možnosti poškození. Za spoluúčasti odborníků budou vytvořeny věrné repliky (3D modely) různých artefaktů (v současnosti konkrétně budov, měst, soch, později například 3D scany knih, maleb…), a tyto repliky budou zobrazovány a zkoumány ve

virtuální a rozšířené realitě.

Téměř s každou další prací zkoumající rozšíření či optimalizace projektu se objevují další možnosti jeho využití i v jiných oblastech, než bylo původně zamýšleno. Například vytvořené analýzy, návrhy a implementace databází, či vytvořené zásuvné moduly (*Blender*) budou v příštích letech využívány jako podpora výuky a modelování (Univerzita Hradec Králové, počítačová grafika 3), případně výzkumu [14].

2.2 Rozbor jednotlivých částí (modulů)

Vzhledem k tomu, že v rámci projektu bylo již v různých oborech obhájeno 12 závěrečných prací (+3 v projektu *Virtuální historický průvodce*), zde pro lepší pochopení kontextu, velikosti projektu a dalších částí mé práce uvedu obecný pohled na jednotlivé funkce, moduly a myšlenky projektu, jehož celkové schéma je vidět na Obrázku 2.1

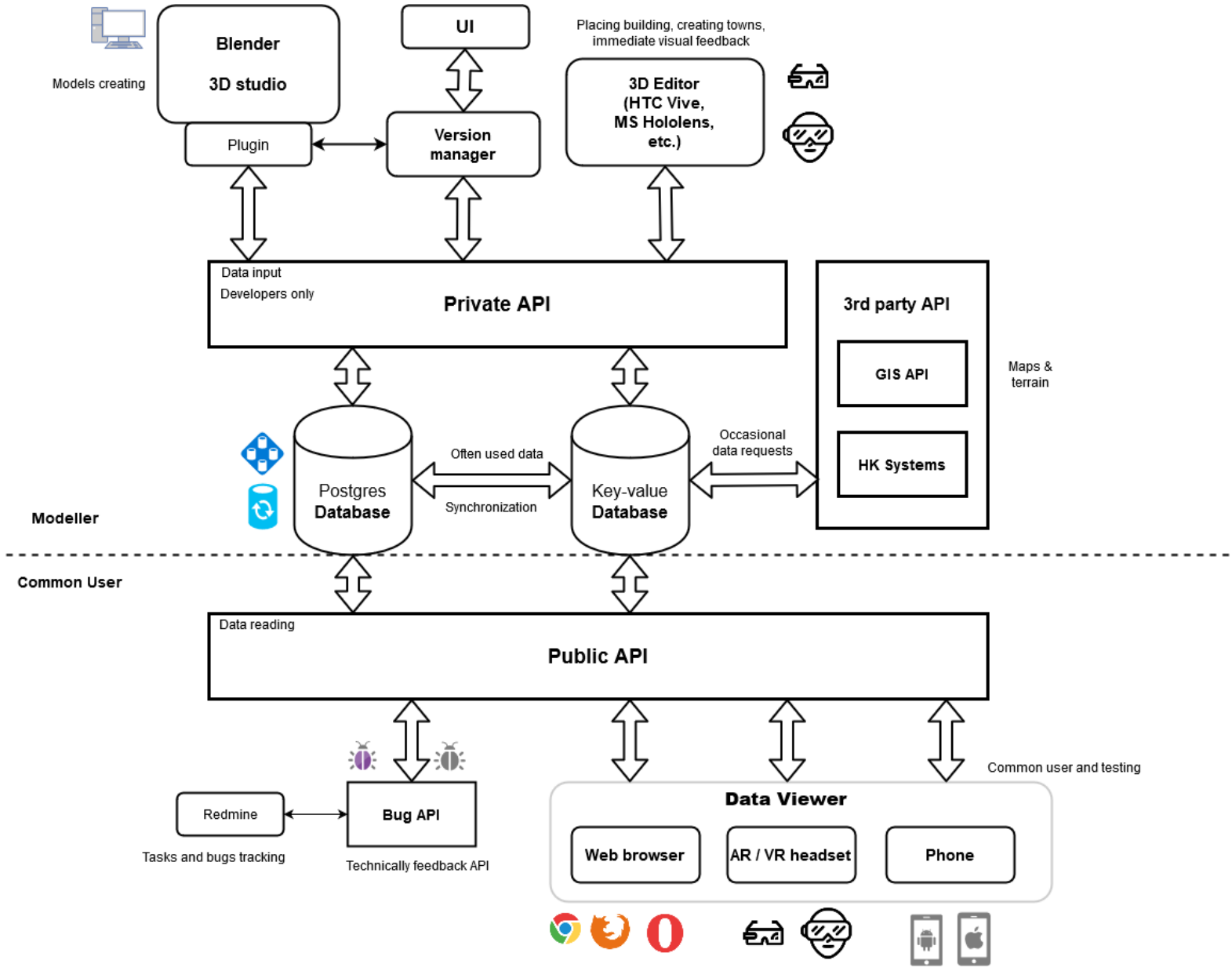
Jak již bylo řečeno dříve, jedním z konečných účelů celého projektu je zobrazení a úprava věrné podoby daného (věnného) města ve VR či AR. Pro tento koncový produkt je nutná celá série kroků. Nejprve musí někdo vytvořit 3D model zadaného historického artefaktu. Tato činnost bude prováděna modelářem, velmi pravděpodobně pracujícím se softwarem *Blender* či *3D Studio MAX*. Po vytvoření daného modelu je třeba jeho uložení na datovém úložišti, ke kterému mají přístup ostatní pracovníci na projektu – v našem případě databáze na serveru. K tomu slouží exportní zásuvný modul [6], který byl vytvořen jako jeden z výstupů projektu VMČK, umožňující přímé ukládání modelů do databáze přes příslušné endpointy *Privátního API*, jehož konkrétní podoba je mimo jiné předmětem této práce.

Po několika experimentech a rozsáhlejším zkoumáním návrhu projektu se ukázalo, že samotný proces modelace nelze v našich rozšířeních projektu brát jako jednorázovou záležitost, kde modelář vytvoří model a ten uživatel po krátkém testování uvidí. Jelikož nelze očekávat, že modelář vytvářející dle podkladů model dané budovy je zároveň i odborník na historii, je nutné výsledek jeho práce před publikací zkontrolovat historikem. Tato kontrola pak musí proběhnout při každé iteraci a přepracování modelu podle nových instrukcí.

Kromě kontroly historické správnosti modelu je pak nutné zkontrolovat na modelu i jeho grafickou správnost. V rámci této nadstavby nad původní návrh VMČK pak vznikla diplomová práce [13] Ing. Michala Martinka, se kterým jsem na tvorbě tohoto *schvalovacího procesu* úzce spolupracoval. Tento proces probíhá na úrovni *Privátního API* a podrobněji bude rozebrán později v kapitole 4.1.1.

Pro usnadnění tvorby modelů a realizaci zpětné vazby v případě chyb či oprav je již delší dobou uvažována existence jistého verzovacího systému.

[1] autor práce, Rostislav Osvald, Juraj Filan, Ondřej Lang, Peter Kanoš, Jakub Neustadt, Michal Zajíc a Jakub Drgoň. Samotný obrázek vytvořil autor práce



Obrázek 2.1: Schéma celého projektu (později jen *Big picture*) vytvořené studenty[1]během týmového projektu

Pro přehlednost a hlavně zdůraznění této skutečnosti je ve schématu oddělen modul *Version Manager* od *Privátního API*, i když bude fakticky veškeré verzování probíhat na úrovni API a databáze. Vzhledem k tomu, že nemůžeme od každého modeláře očekávat znalost a fungování verzování v rámci API, je současně s touto prací Martinem Púčalou vytvářeno webové rozhraní (na schématu označené jako modul UI) umožňující mimo jiné zobrazení jednotlivých verzí modelu.

Samotné zobrazování vytvořených 3D modelů, či později celých měst, uvažujeme v rámci projektu na jakémkoliv zařízení, které umí využívat AR (mobil či tablet s AR core, Microsoft Hololens…). Ve VR očekáváme prozatím pouze tvorbu a úpravu mapy daného města. Procházku městem doma ve VR momentálně nepředpokládáme, a pokud k její implementaci dojde, bude to rozhodně v konečných fázích projektu.

Jelikož je ale téměř nemožné vytvořit jednu univerzální aplikaci zobrazující na jakémkoliv zařízení libovolný formát 3D objektu v AR či VR, byly a stále jsou vyvíjeny aplikace specifické pro jednotlivá zařízení, konkrétně různé varianty modulu *3D Editor*. Kromě naprosto zjevného účelu (vykreslení dat) mají ale jednotlivé moduly *3D Editor* (jak už název napovídá) mít možnost jednotlivé modely upravovat, eliminovat nesrovnalosti a zvyšovat tak jejich historickou věrohodnost. Tím je například míněno posouvání historických objektů po mapě v řádu centimetrů a méně, aby naprosto přesně kopírovaly svou reálnou polohu, případně natáčení objektů, aby byla jejich hrana přesně rovnoběžná s krajem cesty a podobně. Největší výhodou tohoto přístupu (editace v příslušných realitách) je fakt, že modelář okamžitě uvidí, jak se jeho změna projeví uživatelům, a odstraní se tak chyby, které například mohou být z počítače špatně viditelné (model je lehce nakloněn a na jedné jeho straně se tak vznáší nad podlažím, mezi domy byla ulička, ale kvůli současnému umístění modelů je moc úzká, a podobně).

Jedním z dalších problémů při zobrazení modelů v rozšířené realitě je určení přesné polohy uživatele. Použití GPS je v tomto případě intuitivní, ale její nepřesnost, obzvláště při nízkém signálu, vede k *třesení obrazu*, případně k zobrazení úplně jiné oblasti. Alespoň částečné řešení nabízejí práce jako [15] či [16] (lokalizační modul na mobilním zařízení), zkoumající možnost rozpoznat přesnou polohu podle fotografií okolí. Za tímto účelem je třeba vytvořit databázi lokalizačních informací, například 360° fotografií.

Problematika zobrazování či úpravy modelů na jednotlivých zařízeních však není předmětem této práce, a pro více informací vizte závěrečné práce zmíněné výše, či práce jako [17] nebo [18].

Modul *Data Viewer* je v tuto chvíli uvažován jako zreplikovaná forma *3D Editoru* bez možnosti editace jednotlivých objektů. Slouží čistě pro komunikaci s koncovým uživatelem. Jeho hlavní úlohou je tedy zobrazovat data, vypočítávat, která data se mají v současné chvíli do zařízení stáhnout, teoreticky zprostředkovávat navigaci po jednotlivých památkách (více v kapitole 4.4), ale také sbírat od uživatelů zpětnou vazbu. Pro zdůraznění této úlohy

bylo od *Public API*, ze kterého bude tento modul získávat data, odděleno *Bug API*, které bude propojené s trackovacím softwarem (například Redmine). Primárním účelem *Bug API* je sbírání chybových hlášení z aplikace, případně upozornění na občasné nepřesnosti, ale jelikož předpokládáme přidání možnosti subjektivního feedbacku (zpětné vazby) uživatelů, bude přejmenováno na *Feedback API*.

Všechny ostatní části projektu VMČK, které nebyly zmíněny dříve, nebo byly zmíněny jen letmo v případě API, jsou alespoň částečně předmětem této práce. Pro lepší orientaci vizte schéma 2.2. Jelikož na těchto modulech již závěrečné práce postaveny byly, uvádím pro lepší pochopení kontextu v následujících kapitolách nejprve historii předchozího vývoje a až poté analýzu současného stavu.

Analýza požadavků, návrh a implementace (dokončení) *Privátního API* a databáze pod systémem *Postgres* jsou hlavní náplní mé práce. Účelem prvního zmíněného je ve zkratce kompletní management všech dat přítomných v dané databázi a jeho většímu rozboru se budu věnovat v kapitolách 3 a 4.

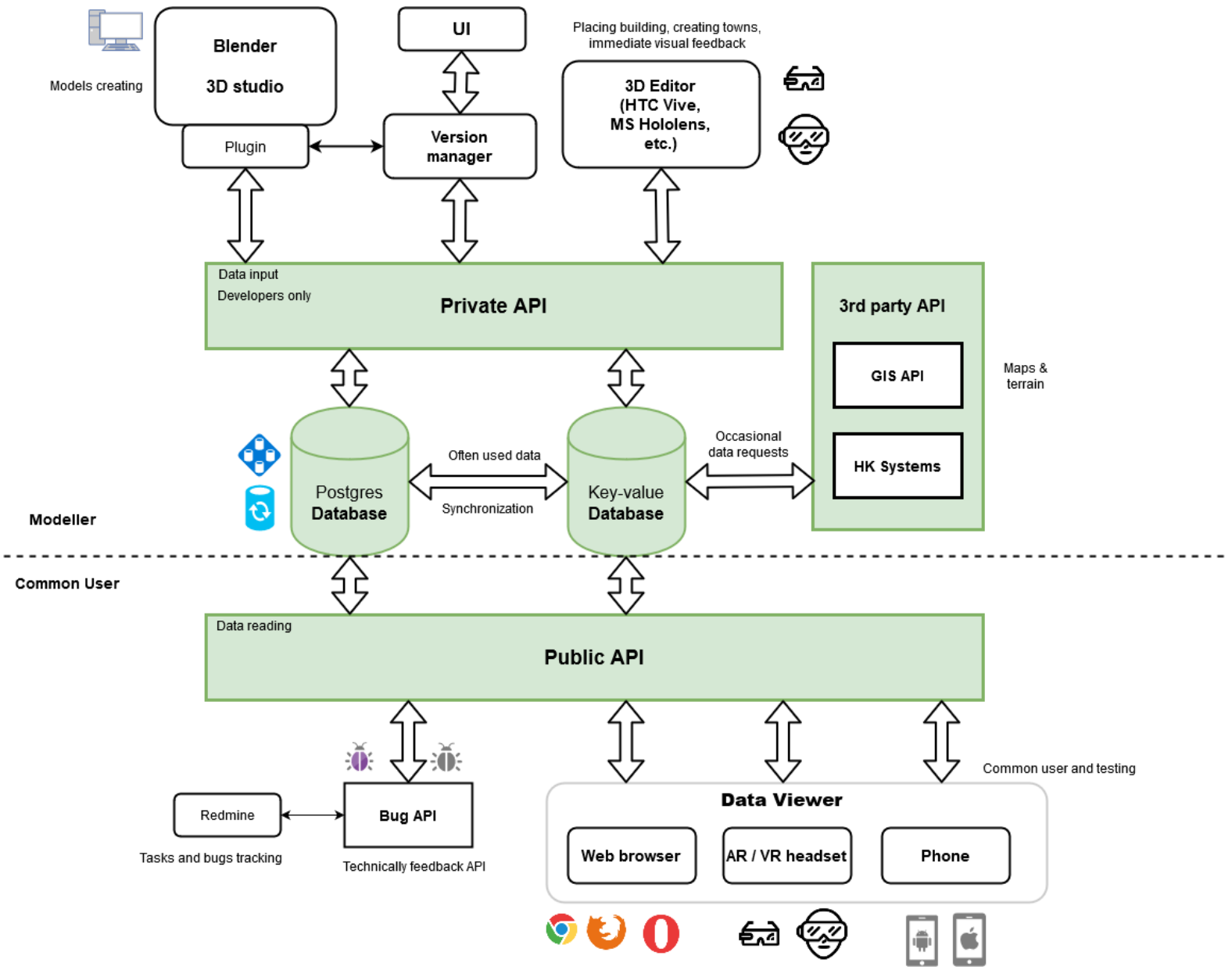
Public API je fakticky upravená část API Privátního, s částečně odstraněnou (úprava dat) a z části novou (např. sběr zpětné vazby – Bug API) funkčností. Oproti tomu *Key-value Database* funguje (jak již název napovídá) na principu klíč-hodnota, tudíž nutnost její úpravy je prakticky nulová.

Hlavní rozdíl modulu *Public API* kromě dříve zmíněných je zaměření se (oproti *Private API*) na výkon a škálovatelnost, kde uvažujeme na několika serverech různé instance API (propojené load balancerem) a DB synchronizované na pozadí. Jedním z důvodů výběru PostgreSQL pro databázi na uložení dat byla právě výborná podpora replikace a synchronizace. Implementace těchto funkcionalit není součástí této práce, jelikož konkrétní kroky byly již zmíněny v práci [11] v kapitole Realizace.

Na první pohled s projektem nesouvisející je *Third party API*, ve kterém jsou zmíněné *GIS API* a *HK Systems*. Tyto moduly mají sloužit primárně pro komunikaci s ostatními pracovišti a univerzitami, konkrétně *HK Systems* slouží pro oboustrannou komunikaci s Univerzitou Hradec Králové (dále jen UHK) a *GIS API* slouží k připojení k systému s mapovými podklady.

Historickou geografií a rekonstrukcemi terénu se zabývá například Historický ústav akademie věd, a jelikož jeho pracovníci již několikrát úspěšně rekonstruovali terén a dnes již neexistující stavby na něm, jejich pomoc byla vyhledána právě kvůli mapám terénu. Existují alternativní způsoby získání podobných dat, například zakoupení leteckých snímků, ty ovšem nejsou nijak levné a neřeší náš problém zobrazení podoby terénu před desítkami či stovkami let. Veškeré tyto údaje se zaznamenávají v podobě GISů (více v sekci 2.5). Konkrétní management terénu, případně rozdělení mapy na menší jednotky a podobně, je opět nad rámec této práce a řešení problémů tohoto tématu se očekává jako téma dalších závěrečných prací či týmových projektů.

Z UHK probíhá spolupráce primárně s vývojovým týmem VMČK sídlícím na katedře pomocných věd historických a archivnictví na filosofické fakultě.



Obrázek 2.2: Big picture projektu - označené části, kterých se týká tato práce

Samotný podnět k projektu VMČK původně vzešel z této katedry a údaje o přesném umístění budov, podobách jednotlivých měst, změnách a podobně budou získávány hlavně odtud. Zároveň jsou zde designovány a implementovány webové stránky, na kterých se budou vyskytovat například 360° fotografie, modely jednotlivých budov, plány města a podobně. V rámci spolupráce se předpokládá vzájemné dodávání modelů, plánů, map atd.

Původně měla být k ukládání a distribuci modelů v projektu využívána databáze [8] (a případné API [9]) vytvořená jako jeden z výstupů projektu Ing. Monikou Borkovcovou, Ph.D z UHK FF. Řešení jejího týmu je plně dostačující pro projekt jak byl původně navržen, ale pro naše plánované rozšíření a optimalizace se objevila potřeba vlastního a komplexnějšího řešení dané DB kvůli nově navrženým modulům a procesům, se kterými její práce nemohla počítat. Příkladem konkrétních opatření, která v její práci chybí, a která chceme v projektu implementovat je například dříve zmíněný systém schvalování modelů, podpora procesu modelování a zavedení verzování, úpravy zobrazených modelů v AR v závislosti na časových, povětrnostních a dalších podmínkách a snadné škálování celého systému pro dosažení co nejlepšího výkonu.

V rámci nadstavby nad původní podobu VMČK bylo tedy po dohodě s vedoucím práce rozhodnuto o vytvoření databáze a API, které budou mít univerzálnější a škálovatelnější použití, výsledkem čehož jsou práce jako [10], [11], [12] a tato. Moje práce tedy vzniká kromě pokračování zmíněných prací také jako nadstavba řešení pí Borkovcové a synchronizace našich databází bude v budoucnu probíhat právě přes modul nazvaný HK Systems.

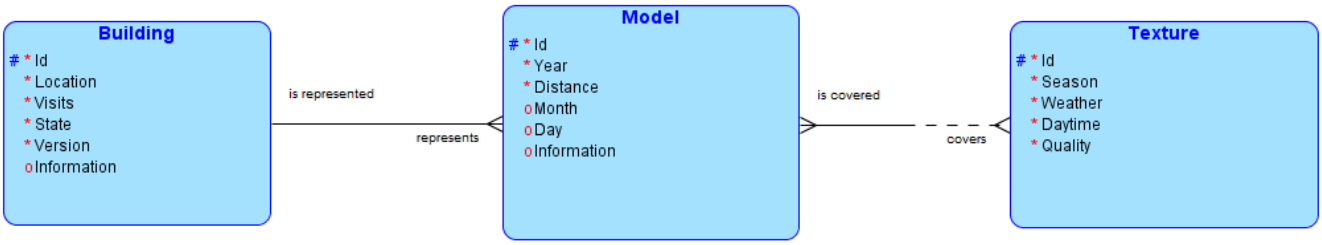
2.3 Data nutná pro VMČK

2.3.1 Zobrazení měst za různých okolností

Při tvorbě a následném zobrazení objektů v AR či VR máme hned několik možností, jak přistupovat k zobrazení jeho variant. Prvním a dodnes často používaným řešením je vytvoření objektu v jeho jediné podobě a tu zobrazovat uživatelům nezávisle na okolnostech. Vzhledem k tomu, že se v projektu zaměřujeme na co největší historickou věrohodnost koncového produktu, se nám tato možnost jevila jako naprosto nepřijatelná – uprostřed zasněženého města nedává smysl zobrazovat dům zalitý sluncem a okna plná rozkvetlých květin.

Další možností, která se jeví už jako přijatelnější, je vytvoření jediného modelu a tvoření jeho variant v závislosti na okolnostech koncového zařízení (počasí, denní a roční doba…) v momentě jeho zobrazení. Z této varianty bylo upuštěno poté, co jsme zjistili, že některé operace v závislosti na použitém hardwaru trvají desítky až stovky minut, pokud mají být věrohodné.

Pro co nejlepší zážitek koncového uživatele bylo vybráno poslední uvažované řešení – model a veškeré jeho varianty jsou vyrenderovány předem a koncovému uživateli pouze zobrazeny. Uživatel tedy při prohlídce daného



Obrázek 2.3: Návrh struktury datového úložiště v [17] a [11]

města bude mít skutečně pocit, že se ve stejné chvíli nachází v jeho minulosti, jelikož vše, co v AR uvidí, bude nasvícené/zasněžené/...úplně stejně jako realita kolem něj.

2.3.2 Vývoj ukládaných dat

Jednou z prvních prací, která položila základ samotnému projektu VMČK, je práce [11] od Bc. Víta Černého, ve které se zabývá výběrem jednotlivých technologií pro tehdy ještě *Virtuálního historického průvodce*. Právě z jeho práce poprvé jasně vyplynulo směřování projektu k použití PostgreSQL a technologie *Node.js*. Současně s jeho prací vznikla práce [17] Bc. Heleny Pavlíkové, která se sice primárně zaměřovala na vizualizaci městské infrastruktury, ale i tak poskytla velmi užitečný náhled na požadavky, které klade na API a databázi klientská aplikace. Již v této práci se ukázalo, že pro navržení optimálního rozhraní je velmi užitečná spolupráce právě klientské (*front-end*) části se serverovou (*back-end*), a tato spolupráce při vytváření API se přenesla i do dalších let.

Jak je však vidět na obrázku 2.3, tyto závěrečné práce jsou skutečně pouze počátkem práce na projektu a velikost i struktura navrženého datového úložiště tomu značně odpovídají. Jejich primárním účelem a výsledkem byl výběr technologií použitých později ve VMČK [11], čímž alespoň částečně určili směřování projektu v následujících letech.

Obě tyto práce vznikly v době, kdy konkrétní podoba VMČK stále ještě nebyla úplně určena. Jako prozatímní jméno bylo tehdy vybráno právě *Virtuální historický průvodce*, a až po ukončení činnosti Víta Černého a Heleny Pavlíkové došlo ke konečnému schválení projektu a k určení jména VMČK. Přibližně v té samé chvíli byla VMČK zadána jako týmový projekt následujícím ročníkům.

Primárním účelem prací v těchto týmových projektech byla právě analýza datových úložišť, projektu jako celku a mimo jiné i problematiky zobrazení a manipulace s 3D modely v rozšířené realitě, spíše než jakákoliv forma implementace návrhů řešení. Hlavním výsledkem těchto analýz byl návrh datového úložiště (Obrázek 2.4) a dříve uvedené schéma projektu jako celku (*big picture*

– Obrázek 2.1). Přímým vyústěním zkoumání možností a potenciálu projektu v tomto stádiu byly závěrečné práce jako prototyp jádra mobilního klienta [19], modul rozpoznávání obrazu [15], různé úpravy textur [20, 21, 22] a v neposlední řadě druhá iterace vývoje *Privátního API* [10].

Poslední zmíněná práce přímo navazovala na schéma datového úložiště 2.4, a zaměřila se na jeho vylepšení, tvorbu API a technologie použité k jeho vývoji. Bohužel byla tato práce úzce zaměřena na problematiku editoru virtuální reality, a nebrala v potaz účel projektu jako celku, tudíž další vývoj ve VMČK je touto prací pouze inspirován, ale nestojí na ní. Důležitým výsledkem této práce byl ale výběr technologií (z nichž část byla určena už v práci [11]), které jsou v projektu v současnosti použité, a to navzdory tomu, že autor v práci pouze uvádí seznam použitých technologií, ale nikde neuvádí srovnání s ostatními technologiemi či konkrétní důvody pro jejich použití. Po konzultaci Bc. Jindřicha Mácy s vedoucím práce byly tyto technologie finálně schváleny jako vhodné pro pokračování ve vývoji (více v kapitole 2.5).

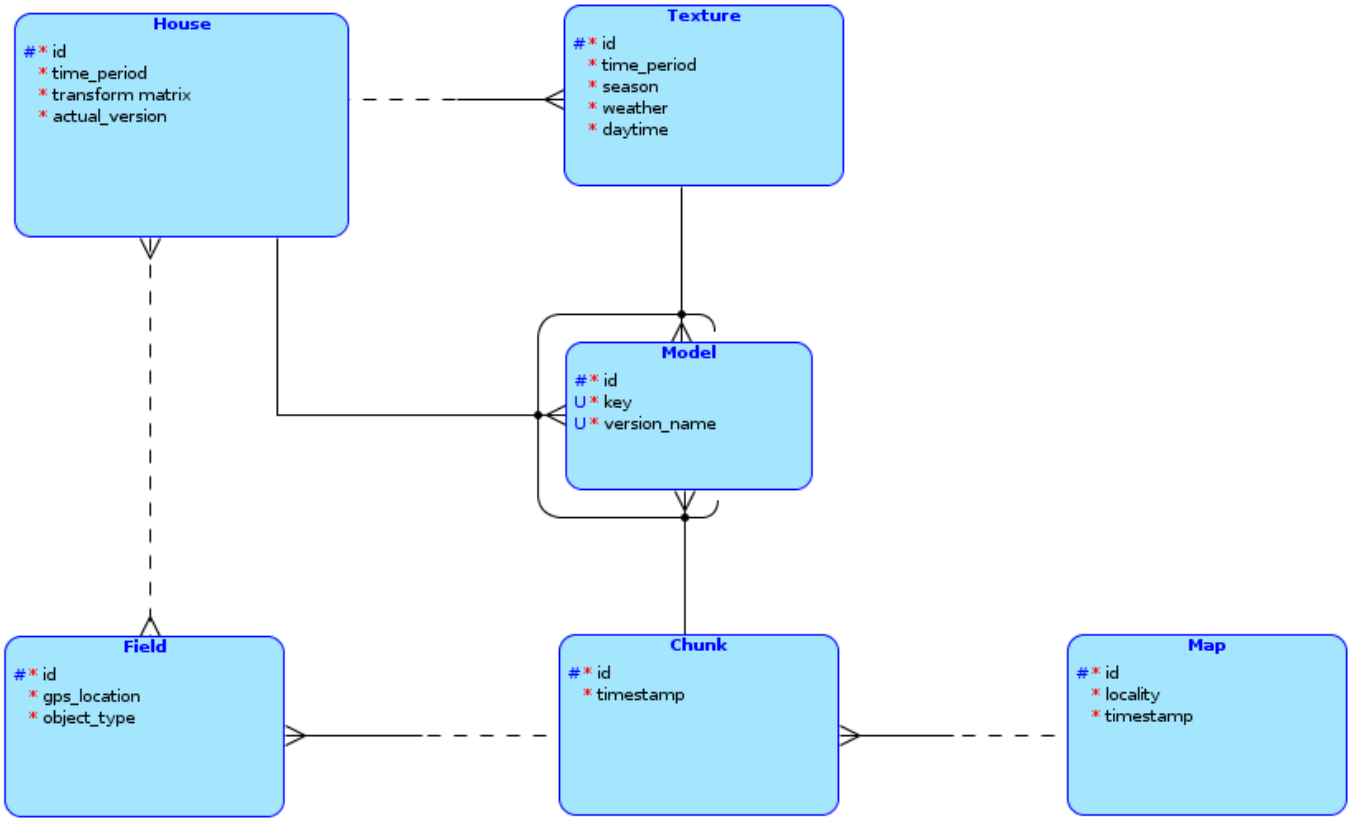
Jako poslední práci, která přímo ovlivnila podobu projektu, je potřeba uvést práci [12]. Ta přímo navazovala na schéma 2.4 a ve spolupráci s touto prací se zaměřila na jeho vylepšení a implementaci samotného API.

2.4 Stav API a databáze při převzetí vývoje

Základní návrh API v práci [12] se dostal přibližně do stavu uvedeném na schématu 2.5, tj. umožňovalo vytváření a načtení záznamů o 3D objektech (dříve entita *House*) a jejich modelech či texturách, bez možnosti úpravy dat či jejich mazání. Konkrétní stav (kód) implementace při převzetí je uveden na přiloženém médiu ve složce */src/api-start/*. Kromě zjevných možností uvedených výše stojí za zmínku ještě vytvoření verzování jednotlivých 3D objektů, v dané chvíli řešené pouze (celým) číslem verze a odkazem na předchozí objekt, dále vytvoření aplikace za účelem velmi snadné instalace a převzetí vývoje (více v kapitole 5.1) a implementace rozdělení dat do dvou databází.

První zmíněnou databází je *MongoDB*. Zde jsou uloženy samotné modely historických objektů, jejich textury a podobně. Tato databáze je založena čistě na principu *key-value*, kde klíčem pro vyhledávání byl vybrán md5 hash a hodnotou samotný soubor. Původně bylo uvažováno uložení těchto souborů na file systému, ale tato možnost se při analýzách v rámci týmových projektů ukázala jako velmi pomalá. S elegantním řešením formou *key-value* databáze přišel Ing. Michal Valenta, Ph.D.

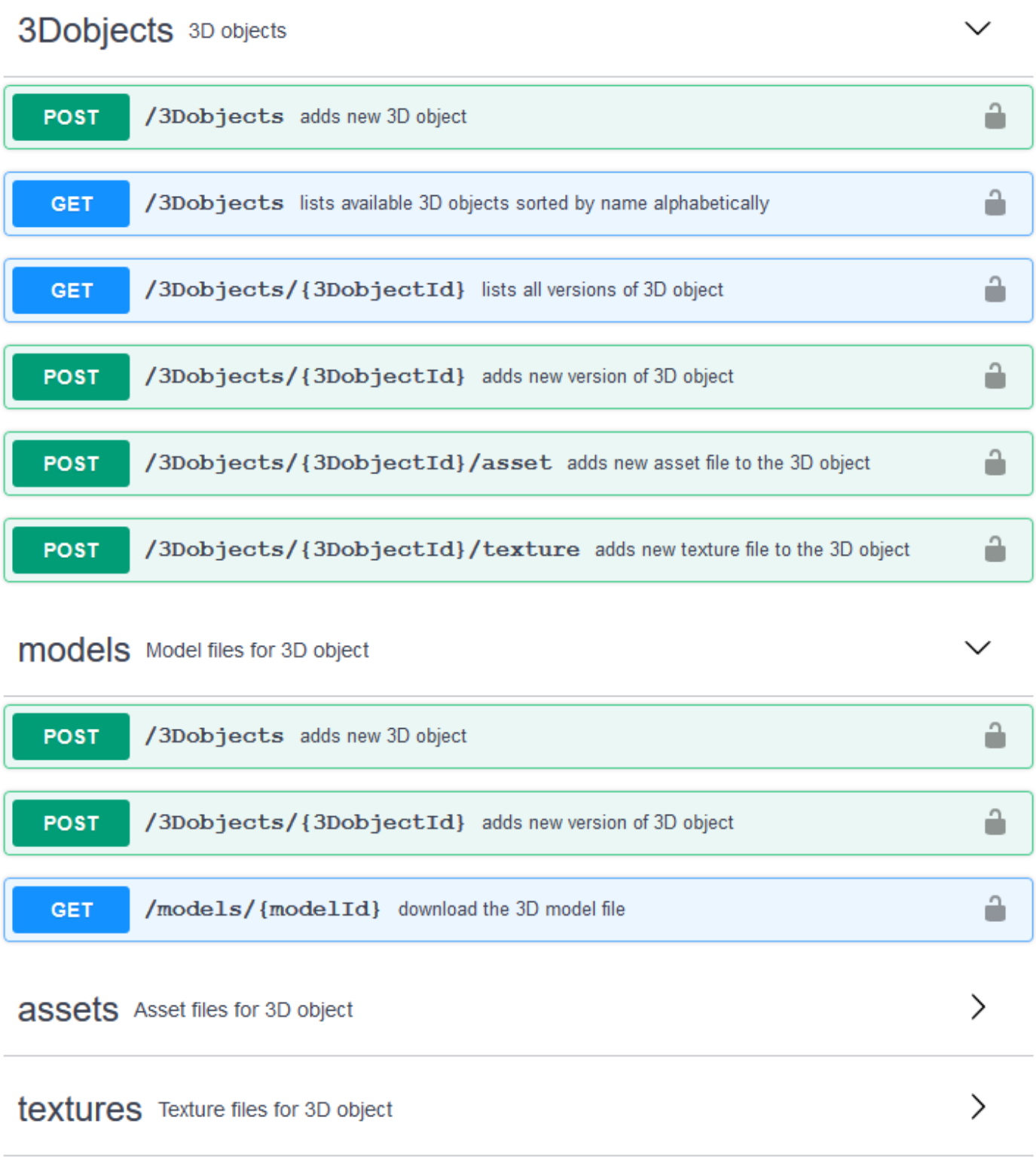
Druhou částí je databáze metadat pod *PostgreSQL*. Zde jsou uložené veškeré údaje o objektech vystupujících v projektu, a tato databáze je kromě *Privátního API* hlavní náplní této práce. Zde bude probíhat veškeré vyhledávání a filtrování. Až bude třeba fakticky zobrazit samotné modely, získají se z první databáze na základě jejich md5 hashe.



Obrázek 2.4: Návrh struktury datového úložiště vytvořený Jurajem Filanem v rámci týmového projektu, použito v práci [10]

Rozdělení dat do těchto dvou databází bylo vybráno z důvodu, že projekt VMČK situován pro provoz na mobilních datech, a tudíž je třeba minimalizovat jejich využití, kdekoliv to není nutné. Zároveň je ale v projektu nutné zobrazovat mnohdy netriviální objekty, a to v reálném čase, tudíž je zároveň potřeba minimalizovat odezvu serveru.

Bylo tedy rozhodnuto o zmíněném oddělení samotných modelů k zobrazení od jejich metadat. Zmenšením záznamů v databázi metadat na mnohdy až tisíciny jejich původních velikostí bude dosaženo rychlejšího filtrování a hlavně menšího využití dat. Klientská aplikace si ze serveru stáhne metadata o modelech, ale ne modely samotné, poté si sama vybere, které je potřeba zobrazit, tj. v první řadě stáhnout. Jelikož se budou stahovat pouze modely, které klientská aplikace v dané chvíli potřebuje, rapidně tím klesne i doba nutná pro jejich načtení i využití mobilních dat.



Obrázek 2.5: Podoba API vytvořená na swaggeru Bc. Jindřichem Mácou [23]

2.5 Použité technologie

Předchozí závěrečné práce, týmové projekty a zadavatel práce společně jednoznačně určili technologie použité ve VMČK. Jelikož součástí této práce není jejich výběr či porovnání, uvádím zde pouze jejich seznam a krátký popis. Kvůli velikosti projektu a rozmanitosti modulů (od automatických úprav textur po administrativní rozhraní) zde uvádím pouze technologie týkající se této práce. Jelikož se ale od práce [10] seznam technologií prakticky nezměnil, tak se tato kapitola téměř zcela překrývá s totožně pojmenovanou částí dané práce.

Docker: Nástroj zabývající se *kontejnerizací* aplikací, neboli uzavření aplikace do *kontejneru* (tj. jejich oddělení od ostatního prostředí počítače), kde v každém z nich jsou veškeré závislosti, které aplikace potřebuje. Tím je umožněn snadný vývoj na jakékoliv platformě, která docker podporuje, protože v konečném důsledku pak často pro pokračování ve vývoji stačí nainstalovat docker, vytvořit soubory jako *DockerFile* či *docker-compose.yml* a spustit příkaz *docker-compose up*, více v kapitole 5.1.

Node.js: *„Node.js je open-source multiplatformní JavaScriptové prostředí postaveno na Chrome V8 JavaScript enginu.“* [10] Oproti dřívějším variantám, kdy byl Javascript (dále jen JS) spouštěn v internetovém prohlížeči, byl *Node.js* od prohlížeče oddělen, a nebylo pak už potřeba JS kódy upravovat a testovat na jednotlivých prohlížečích. Jeho další výhodou je, že byl už od začátku navržen s myšlenkou velkého výkonu a dobré rozšiřitelnosti. [24]

Express: Nejpopulárnější framework pro *Node.js*, který umožňuje velmi jednoduše napsat webovou aplikaci. *„I když je velmi minimalistický, vývojáři vytvořili kompatibilní middleware balíčky na téměř každý problém týkající se vývoje webových aplikací.“* [24]

Typescript: Open-source typová nadstavba k původně netypovanému jazyku Javascript, vyvíjená firmou Microsoft. JS byl vybrán pro výbornou výkonnost při zpracování velkého množství požadavků na API, oproti jazykům jako například Java (Spring) nebo Python (Django). Konkrétně Typescript(dále TS) byl vybrán pro usnadnění vývoje, kde čistě ověření typu jednotlivých proměnných zabrání množství chyb, ke kterým je JS náchylný. Pro statickou analýzu kódu a jeho formátování je využíván **TSLint**.

TypeORM: ORM vyjadřuje objektově relační mapování objektů do databáze, a konkrétně tento framework může pracovat s velkým počtem JS platforem, jmenovitě pak například *Node.js*, React Native, NativeScript či TypeScript. [25] Hlavní výhodou oproti například *Sequelize* je výborná spolupráce s TS a snadnější úpravy databáze pomocí migrací.

JSON Web Token (JWT): Označení standardu sloužícího k bezpečnému přesunu informací ve formátu JSON (JavaScript Object Notation), v projektu VMČK používaný k autentizaci uživatele. Skládá se ze tří částí: Header (většinou obsahuje typ tokenu a šifrovací algoritmus), Payload (samotná data, občas dodatečné informace jako expirace tokenu a podobně) a Signature (podepsaná, tj. zakódovaná podoba předešlých 2, slouží k ověření, že se zprávou nebylo manipulováno). Oproti ostatním standardům používaným k témuž účelu, např. SWT (Simple Web Token) nebo SAML (Security Assertion Markup Language Tokens) byl vybrán z důvodu jednodušší syntaxe (oproti SAML) a větší bezpečnosti (v porovnání s SWT). [26]

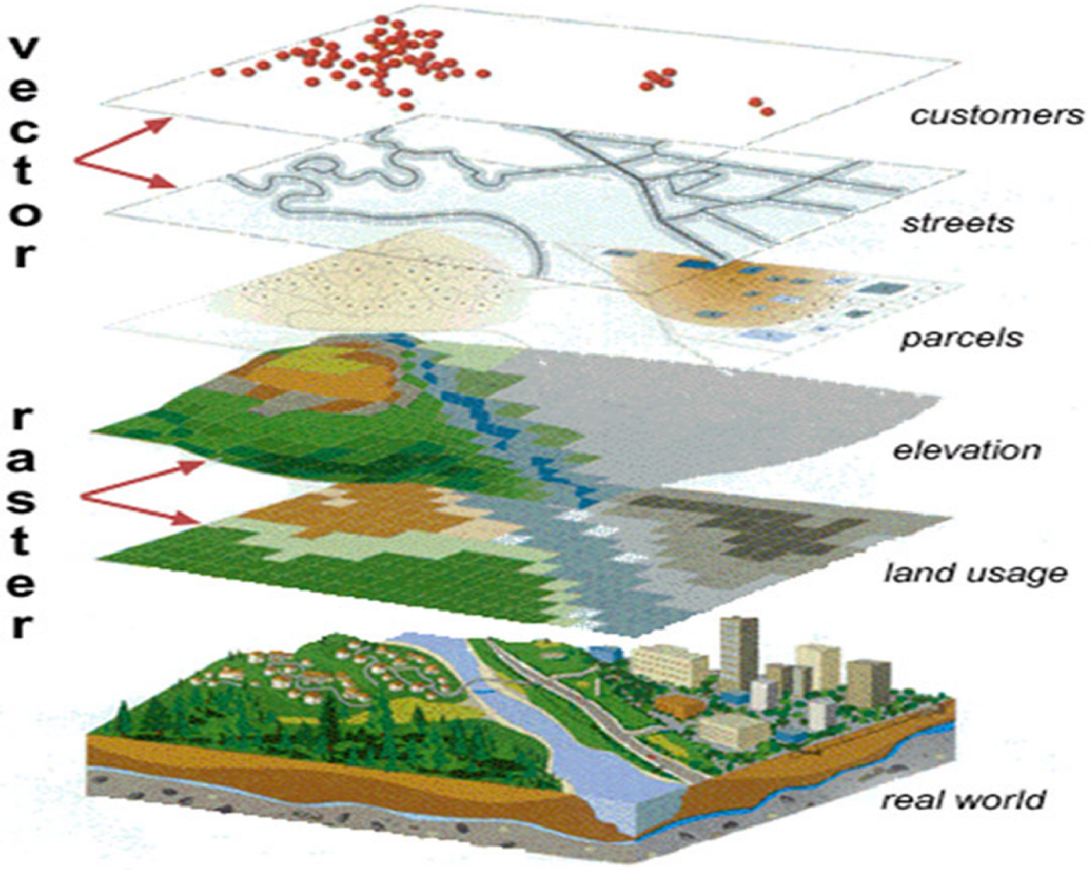
PostgreSQL: Open-source objektově relační databázový systém, použitý pro uložení veškerých (meta)dat nutných pro chod projektu. *„Je pověstný svou spolehlivostí a vysokou bezpečností. Na PostgreSQL wiki* [2] *lze nalézt rozsáhlý seznam dostupného open-source software, sloužícího k administraci a monitoringu PostgreSQL databází. Nejrozšířenější a velmi přehledný je pgAdmin.“* [10]

MongoDB: Dnes jedna z vedoucích open-source NoSQL technologií. Slibuje flexibilní strukturu dat, široké možnosti dotazování a model architektury, který zajišťuje škálovatelnost a vysokou dostupnost. Pro ukládání záznamů používá formát dat BSON (Binary JSON) a jelikož nemá vlastní dotazovací jazyk, používá k přístupu k jednotlivým záznamům (dokumentům) styl klíč-hodnota.

Geografický Informační Systém: Komplex různých zařízení, programů, lidí a metod použitých k získávání, správě a analýze *„dat o geografické poloze prvků či jevů v území“*. [27] Zároveň se ale stejným názvem (zkráceně GIS) nazývá i výstup tohoto komplexu a sice samotný souhrn všech těchto dat.

Typově mezi tato data patří například mapa silnic a řek, struktura a rozvržení obydlení, nadmořská výška, satelitní/letecké fotky atd. Spojením všech těchto dat dostaneme celkem věrný obraz skutečného světa, viz obrázek 2.6. Právě tento model bude projekt Věnná města využívat k zobrazení terénu. Konkrétní zpracování těchto dat a jejich ukládání bude předmětem dalších závěrečných prací či týmových projektů, tato práce pouze počítá s budoucí možností jejich využití.

[2] `https://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools`



Obrázek 2.6: Struktura GISu – převzato z prezentace během [27]

Kapitola 3

Rozbor zadání

Jelikož podstata zadání této práce je *de facto* převzetí a dokončení prací [10] a [12], rozeberu zde nejprve stav projektu při jeho převzetí, respektive jeho nedostatky a chybějící části. Tím získáme jasnější přehled o stavu projektu bez nutnosti studovat samotný kód (v tomto ohledu je tato kapitola podrobnější pokračování kapitoly 2.4) a alespoň částečně seznam funkčních a nefunkčních požadavků aplikace.

3.1 Hlavní nedostatky předchozího návrhu

3.1.1 Schéma databáze

Základní myšlenka předchozího návrhu databáze (pro lepší přehled znovu uvedeného jako Obrázek 3.1) vypadala přibližně následovně:

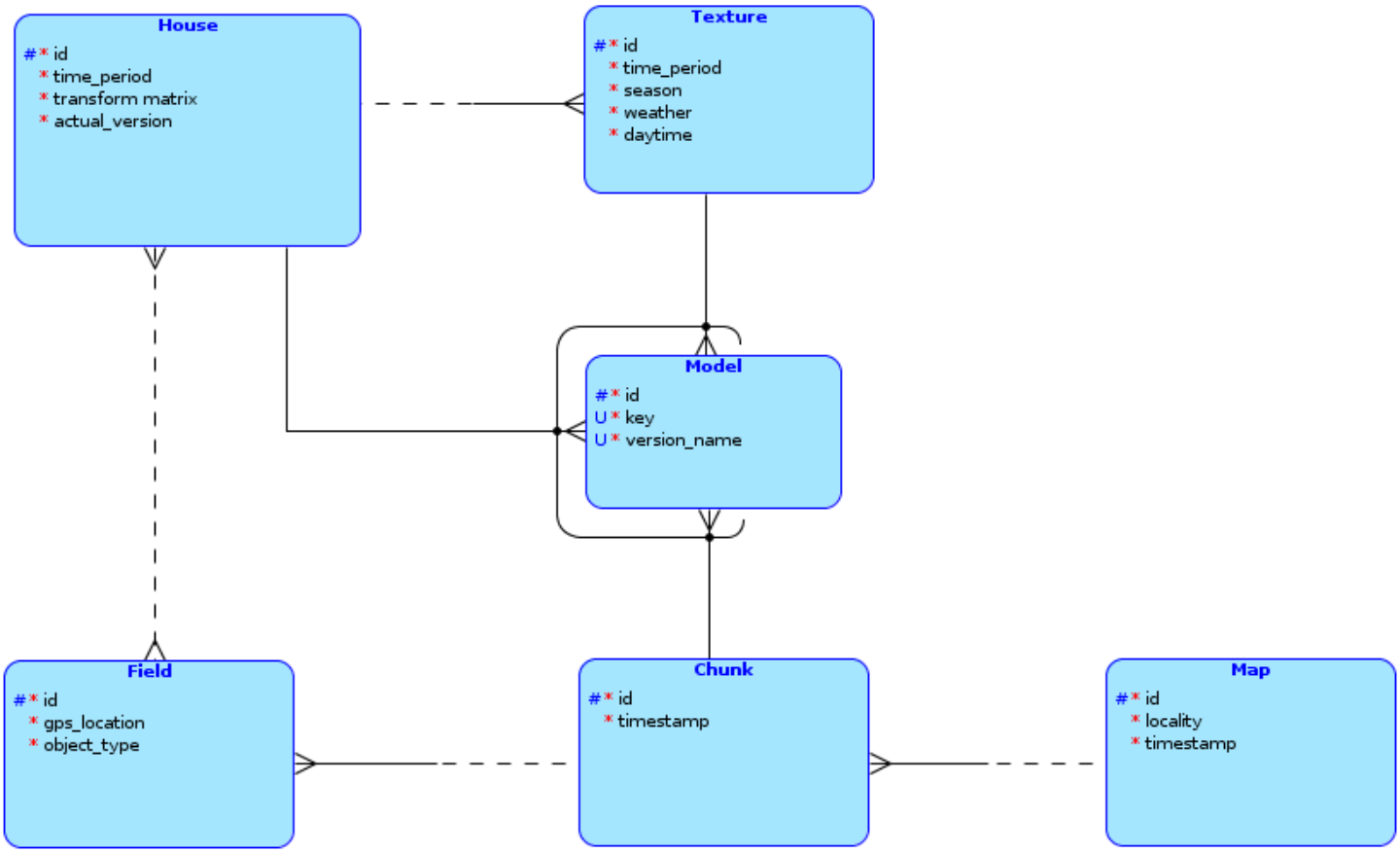
V databázi je uložená mapa města (záznam v tabulce *Map*). Jak je vidět na schématu, tabulka je to značně jednoduchá s jedinými atributy *lokalita* a platnost záznamu.

Každá mapa má být rozdělena na menší části (*Chunky*) pro ušetření mobilních dat při načítání. Každý *Chunk* by měl mít v DB uložený svůj 3D model terénu k zobrazení, proto má v DB vazbu na tabulku *Model*, která je ve schématu uvažována čistě jako spojení s *key-value* databází, tj. místa uložení samotných souborů.

Field byl v projektu uvažován jako půdorys, na který by byl umístěn daný 3D model budovy, kde každý *Field* musí být umístěný na nějakém *Chunku*. Atribut *object_type* napovídá rozdělení těchto objektů do kategorií podle toho, jaký objekt je na nich položen (dům, socha, fontána, cesta…) – pravděpodobně kvůli vyhledávání.

Absence téměř jakýchkoliv dat v těchto tabulkách napovídá, že neproběhl podrobný výzkum jejich optimální (či vůbec nějaké) podoby či načítání.

Tabulky *House*, *Texture* a *Model* nápadně připomínají schéma 2.3 z počátků projektu, kde v tomto případě *House* je samotný 3D model budovy,



Obrázek 3.1: Pro přehlednost znovu uvedené schéma DB při převzetí vývoje

který má u sebe uvedeny modely (tj. soubory ke stažení z *key-value databáze*) a blíže neurčitý počet textur, kde každá má opět vazbu na tabulku *Model*, kvůli stažení souborů.

Atributy u tabulky *House*: *time_period* – označení doby platnosti dat a náznak úmyslu budoucího filtrování podle historických období. *transform matrix* – návrh částečného řešení ukládání budovy na mapě. Samotná poloha (GPS) je uložena ve *Fieldu*, na kterém je budova umístěna, ale jednotlivé drobné úpravy, vzniklé např. nepřesností *Fieldu*, jsou zaznamenány ve transformační matici. *actual_version* – náznak úmyslu řešení verzování jednotlivých 3D modelů.

Atributy u tabulky *Texture*: *time_period* – stejné jako u *House*. *season, weather, daytime* – náznak úmyslu filtrování 3D modelů podle jejich vlastností. Například v zimě se budou (pokud tomu počasí bude odpovídat) načítat pouze

zasněžené textury a modely.

Atributy u tabulky *Model* jsou opět velmi jednoduché: *key* – klíč použitý v *key-value* databázi pro přístup k uloženému souboru. *version_name* – navržené řešení verzování.

Již z této krátké analýzy navrženého schématu je patrné poměrně velké množství nedostatků (první uvedené se týkají věcí obsažených ve schématu, věci chybějící případně vyžadující přepracování uvedu až poté):

Map, Chunk a Field jsou nedomyšlené: Tyto tabulky slouží čistě jako *placeholder*. Je potřeba provést poměrně rozsáhlý průzkum jejich ukládání a podoby (stačí čtvercová síť, nebo je ideální nějaká forma grafu s variabilním tvarem *Chunku*?). Dále pokud má *Field* sloužit jako půdorys, je nutné uložení dat jeho o tvaru, ale v navrženém schématu cokoliv podobného chybí.

Field se jeví jako nedůležitá tabulka: Jediné využití, které si dokážu představit, je spolupráce více historiků při tvorbě města, kdy nejprve skupina vytvoří půdorysy všech budov podle starých map, a poté jiná skupina bude budovy na jednotlivé půdorysy pokládat. Toto řešení je však vyvažováno pomalejší odezvou serveru (je nutný JOIN další tabulky), lehce zvýšenou spotřebou dat (uvážíme-li další data v tabulce Field určující třeba jeho tvar a podobně), ale hlavně dvojnásobným množství práce historiků tvořících danou mapu. Poslední zmíněné je podle mně dostatečný důvod k odstranění dané tabulky, už jen kvůli tomu, že způsobuje zbytečné nepohodlí uživatele a v počátcích projektu tím zpomaluje jeho šíření. Pozice dané budovy může být uložená u ní a atribut object_type se jeví jako zbytečný, obzvlášť zvážíme-li pojmenování tabulky *House*.

GPS je na mobilních zařízeních nepřesný údaj: Je třeba vylepšit způsob určení přesné polohy zařízení pro lepší zobrazení modelů v AR, například založený na pracích [15] a [16].

House a Texture – další atributy a vyhledávání: Jelikož uvažujeme projekt jako průvodce historií, očekával bych v *House* alespoň atributy jako *name* a *description*, či jinou formu podrobnějších informací pro koncového uživatele. Jejich absencí návrh téměř znemožňuje jakékoliv vyhledávání nad danou tabulkou (až na časové období). U *Texture* je sice náznak atributů pro vyhledávání, ale očekával bych například i údaje o složitosti textury (Level of Detail, dále jen LOD).

Absence jakýchkoliv informací o stahovaném Modelu: Podobně jako výše bych očekával údaje jako LOD, velikost souboru, formát…

Nemožnost uložit konfigurační soubory: Některé formáty textur vyžadují konfigurační soubor pro jejich správné sestavení (například formát

.sfb, využívaný na mobilních zařízeních, vyžaduje přítomnost .sfa souboru). Daný návrh s ničím takovým nepočítá a velmi tak omezuje využití projektu na daných zařízeních.

Poněkud jednoduchý návrh verzování: Dosavadně navržená varianta podporuje pouze verzování jednotlivých souborů, neboli záznamů v tabulce Model. Tudíž ať chce modelář vytvořit novou verzi textury nebo chunku, vytvoří se tím nový záznam v tabulce *Model*. Takovýto přístup může být pro modeláře matoucí, a povede k přeplnění jedné tabulky, kdežto ostatní budou v porovnání s ní prakticky prázdné. Zároveň navržené schéma neobsahuje žádný atribut využitelný například ke zjištění celé historie jednoho objektu. Intuitivní řešení by bylo vyfiltrovat veškeré modely k danému záznamu podle ID a pak je seřadit podle data vytvoření, ale tabulka *Model* neobsahuje ani to.

Úprava vyhledávání vede k úpravě databáze: Jakákoliv úprava vyhledávacích parametrů (například *season* a *weather* v *Texture*), tj. jejich zrušení či přidání nových, vede nutně k úpravě dané tabulky. Zároveň pokud bychom chtěli vyhledávat podle dat, které mají smysl u vícero tabulek (např. LOD, time_period…), tak nám vznikají duplicitní sloupce v několika tabulkách. Řešení tohoto problému nabízí například zavedení tabulky *properties*, kde přidání nového vyhledávacího parametru vyžaduje pouze INSERT do této tabulky.

Absence uživatelů: Ve schématu DB nejsou vůbec bráni v potaz uživatelé a tím pádem ani vyhledávání modelů podle autorství. Přitom jeden z prvních *use-cases* modeláře, který mě napadl, byl: *Vrať modely, které jsem vyrobil já.*

Absence stavu modelu: Jak již bylo řečeno dříve, v tomto API se počítá s verzováním modelů. API by mělo ukládat různé stavy 3D objektů a to ať už nedokončené (rozdělané) práce či různé varianty téhož domu vyrobené z důvodu nových historických skutečností. Za tímto účelem je třeba přidat k příslušným záznamům údaj o stavu 3D objektu (na první pohled zřejmé *unfinished* oproti *published*).

Téměř veškerý výběr dat je nutný dělat manuálně: Představme si *use-case*, kdy chceme získat veškeré možné podoby jedné konkrétní budovy. Získáme záznam z tabulky *House* a k němu desítky různých souborů (textur), kde je automatizovaně problém zjistit, které z nich se mají použít na vykreslení které transformace (zasněžení, různé osvětlení…). Jedním z řešeních by bylo například přidat si ke každému záznamu vícero konfiguračních souborů s údaji, který soubor slouží k jaké transformaci. Jinou, a podle mě *čistší*, možností by bylo vytvořit další objekt (například *Variant*) zastřešující jednotlivé transformace modelů. *House* pak

bude obsahovat seznam variant, kde každá bude u sebe mít všechny soubory, které k sestavení potřebuje (a případné .config soubory) a žádné jiné.

3.1.2 Struktura a implementace API

Jak je vidět na navrženém schématu API (pro přehlednost znovu uvedeném na Obrázku 3.2), implementované endpointy mají na starosti pouze tvorbu a získávání záznamů, nikoliv však jejich úpravu či mazání. Části *assets* a *textures* například obsahují pouze endpointy *assets/{id}* respektive *textures/{id}* obsluhující GET request, jehož zaslání zahájí stahování daného souboru.

Tato podoba API byla vytvořena a nasazena na server i v takto nedokončené formě, jelikož bylo potřeba v rámci ostatních týmových a závěrečných prací začít pracovat na konkrétní implementaci komunikace s *Privátním API* (např. na mobilním klientovi). Díky tomuto nasazení bylo možné v některých pracích pokračovat a dříve zmíněný plugin do *Blenderu* [6] napojit na příslušné endpointy. Díky tomu bylo (je) tedy možné začít plnit databázi vytvářenými 3D modely budov.

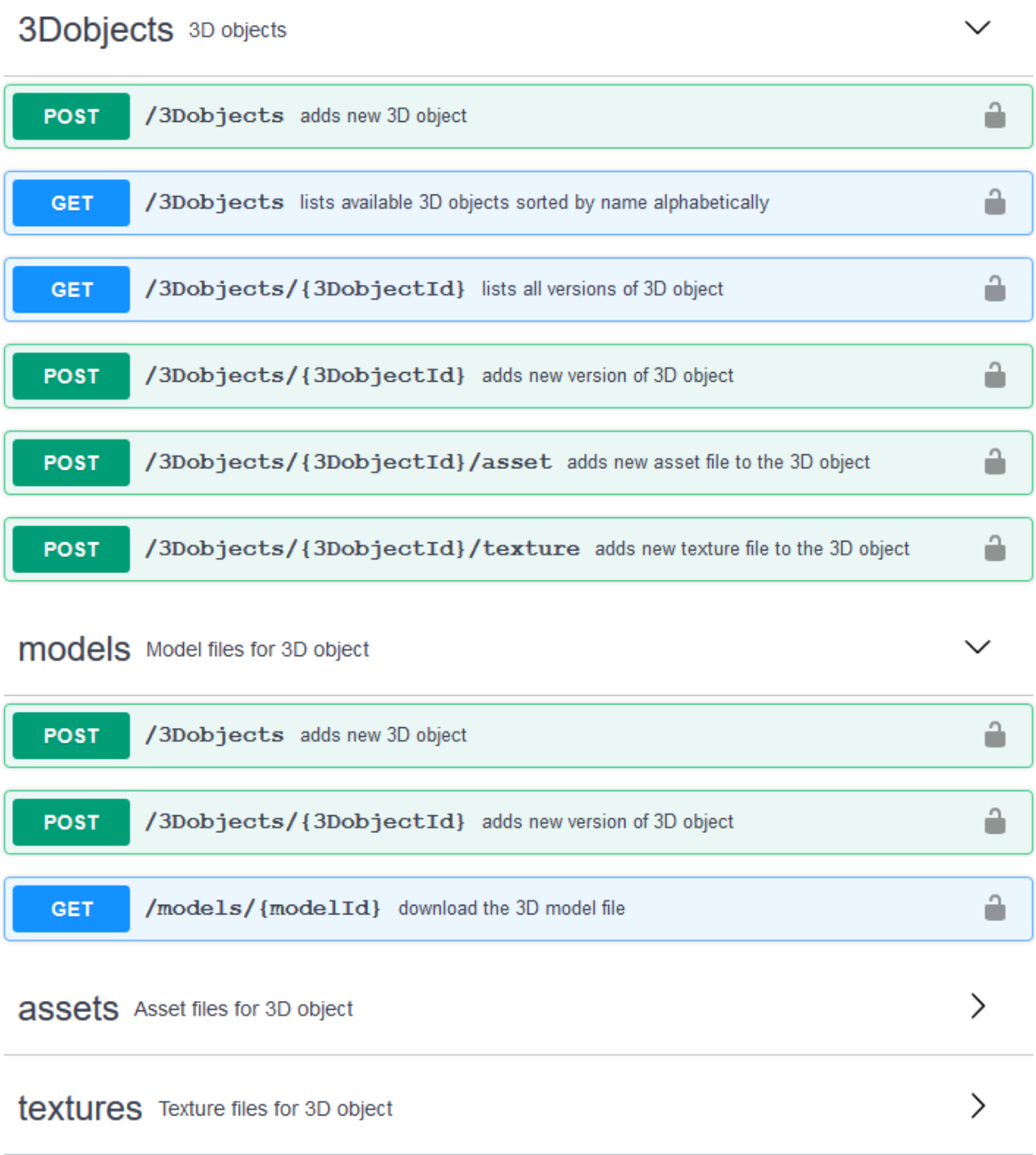
Za hlavní nedostatky tohoto návrhu API považuji následující:

Nemožnost úpravy dříve vytvořených dat: API umožňuje čistě operace Create a Read, ale už ne Update či Delete

Nedostatek dokumentace: Nebylo vytvořeno žádné schéma databáze, a poslední, se kterým tedy můžeme pracovat, je schéma uvedené dříve. Vývojář pak musí stav DB odhadovat z kódu a ze swaggeru . Jedinou výhodou je přítomnost Create scriptu, jakožto první migrace, ze kterého se stav dá vyčíst. V samotném kódu je také minimum komentářů a žádná jiná forma dokumentace neexistuje.

Minimum DB tabulek: API implementuje pouze přístup k tabulkám *3DObject* (dříve *House*), *Texture* a *Model*. Odstraňuje oproti dříve uvedenému schématu DB nedostatky jako absence důležitých dat v daných tabulkách, propojuje obě databáze a umožňuje faktické ukládání souborů a jejich následné stažení, ale ostatním problémům (jako například nutnosti existence endpointů pro schvalovací proces) se tato implementace (jakožto nedokončená) nevěnuje.

Neexistence správy uživatelů a přístupových práv: Je vytvořená autentizace uživatele pomocí JWT tokenu, který má pro sebe v DB každý uživatel vygenerovaný, ale neexistuje zde žádný způsob, jak uživatele vytvořit přes API. Jediný způsob je manuální vytvoření záznamu v DB, stejně tak i veškeré úpravy či mazání uživatelů. Zároveň každý uživatel může všechno, přístupová práva nejsou ani náznakem řešena, ale to je vzhledem k pokročilosti implementace pochopitelné.



Obrázek 3.2: Pro přehlednost znovu uvedené schéma návrhu API vytvořené během práce [12]

.

Jednostranné udržování historie: Dotaz na historii objektu vrátí pouze starší verze. Neexistuje způsob, jak najít novější verzi 3D objektu (*use-case*: Modelář vytvoří model, uloží ho, jeho kolega převezme práci a vytvoří novou verzi, a první uvedený nemá jak se k této nové verzi dostat, maximálně tak přes vyhledávání nad všemi 3D objekty).

3.2 Funkční a nefunkční požadavky

Na základě předchozí kapitoly, dříve vytvořených prací, konzultací s vedoucím práce a ostatními spolupracovníky a analýzy projektu jako celku předpokládám následující požadavky na tuto práci:

3.2.1 Funkční požadavky

F1: Management 3D Objektů včetně modelů a textur: API umožní uživateli kompletní CRUD (Create, Read, Update, Delete) operace nad zmíněnými tabulkami.

F1.1: Verzování 3D Objektů: API umožní uživateli zjistit si celou historii daného 3D Objektu, a to jak starší tak novější verze, dále pak vytvářet nové verze daného 3D Objektu, či je naopak mazat nebo upravovat.

F1.2: Ukládání různých podob 3D objektů: API musí být schopno uložit jeden 3D objekt v desítkách až stovkách podob v závislosti na vytvořených texturách, úrovních detailu a podobně.

F2: Management uživatelů: Uživateli bude umožněno se přes API zaregistrovat. Po registraci bude uživatel čekat na schválení účtu adminem a od toho momentu bude mít uživatel přístup k datům dle jeho přidělených přístupových práv.

F2.1: Přístupová práva a uživatelské role: Uživateli budou v rámci API přidělena přístupová práva a uživatelské role. Tato data může jinak uživateli změnit pouze uživatel s rolí admin.

F3: Properties a jejich management: Veškeré sloupečky, podle kterých se předpokládá vyhledávání nad tabulkami, z nich budou odebrány a přeměněny na záznamy v tabulce *Properties*. Opět bude vytvořena kompletní obsluha všech CRUD požadavků nad danou tabulkou.

F3.1: Podpora tagování 3D objektů: API umožní uživatelům přidávat k jednotlivým 3D objektům různé tagy. Ty budou u každého záznamu uloženy ve formě *properties* a ukládány budou ve stejně pojmenované tabulce.

F4: Filtrování a vyhledávání: API umožní uživateli vyhledávání 3D Objektů, textur a podobně nad sloupečky v daných tabulkách. Pokud nejsou v GET requestu uvedeny vyhledávací parametry, vrací API uživateli seznam všech modelů, kterých je autorem.

F4.1: Vyhledávání modelů podle geografie, počasí...: Veškeré tyto informace budou u záznamů v DB ukládány ve formě zmíněných tagů a API bude podle nich umožňovat vyhledávání.

F5: Schvalování modelů: Nad API bude vytvořen schvalovací proces jednotlivých 3D objektů a příslušné endpointy k tomu náležící. Schválit či odmítnout 3D Objektu jeho historickou správnost může pouze uživatel s rolí *historik*. Schválit či odmítnout 3D Objektu jeho grafickou správnost může pouze uživatel s rolí *grafik*. Pro úspěšné schválení modelu musí model odsouhlasit i *historik* i *grafik*. Pokud alespoň jeden neschválí, jde model s připomínkami na přepracování zpět k modeláři. Uživatel, který má obě zmíněné role nemůže schválit za obě u téhož modelu. Uživatel nesmí schvalovat své vlastní modely.

3.2.2 Nefunkční požadavky

NF1: Minimalizace využití mobilních dat: API bude vytvořeno s ohledem na velkou paměťovou efektivitu z pohledu koncového uživatele.

NF2: Optimalizace ukládaných dat: Ukládání dat v DB musí být implementováno s ohledem na minimalizaci využití úložného prostoru. Za předpokladu stejných dat v několika verzích 3D objektu bude šetřeno místo v databázi technikami počítané reference a *copy-on-write*.

NF3: Minimalizace odezvy serveru: Struktura dat bude v práci zvolena tak, aby odezva serveru byla co nejmenší, a uživatel tak na jejich načtení nečekal dlouho.

NF3.1: Rozšířitelnost, replikace a load balancing: Systém musí být dobře rozšířitelný, zreplikovatelný a synchronizovatelný, aby umožnil rozdělení zátěže na několik serverů pomocí technologie load balancing, a tím zmenšil odezvu serveru na nezbytné minimum.

NF4: Technologie: Práce bude vypracována podle zadaných technologií v kapitole 2.5.

NF5: Dokumentace: API bude ve swaggeru vytvořena dokumentace s jeho aktuálním stavem a kód API bude náležitě okomentován.

Kapitola 4

Návrh řešení

V této části se budu věnovat návrhu nové podoby datového úložiště. V případě *MongoDB* pouze uvedu strukturu ukládaných dat určenou v práci [12] – od dané práce se neliší. V případě *Postgresu* zde uvedu a vysvětlím mnou navržené nové schéma. Poté zdokumentuji a navrhnu, respektive rozšířím dříve navržené, *Privátní API*. Ke konci kapitoly pak uvedu *vize do budoucna*, tj. části projektu, které se této práce přímo týkají, nebo na ní navazují, a **s větším časovým rozpočtem** by se některé i teoreticky daly do této práce zakomponovat.

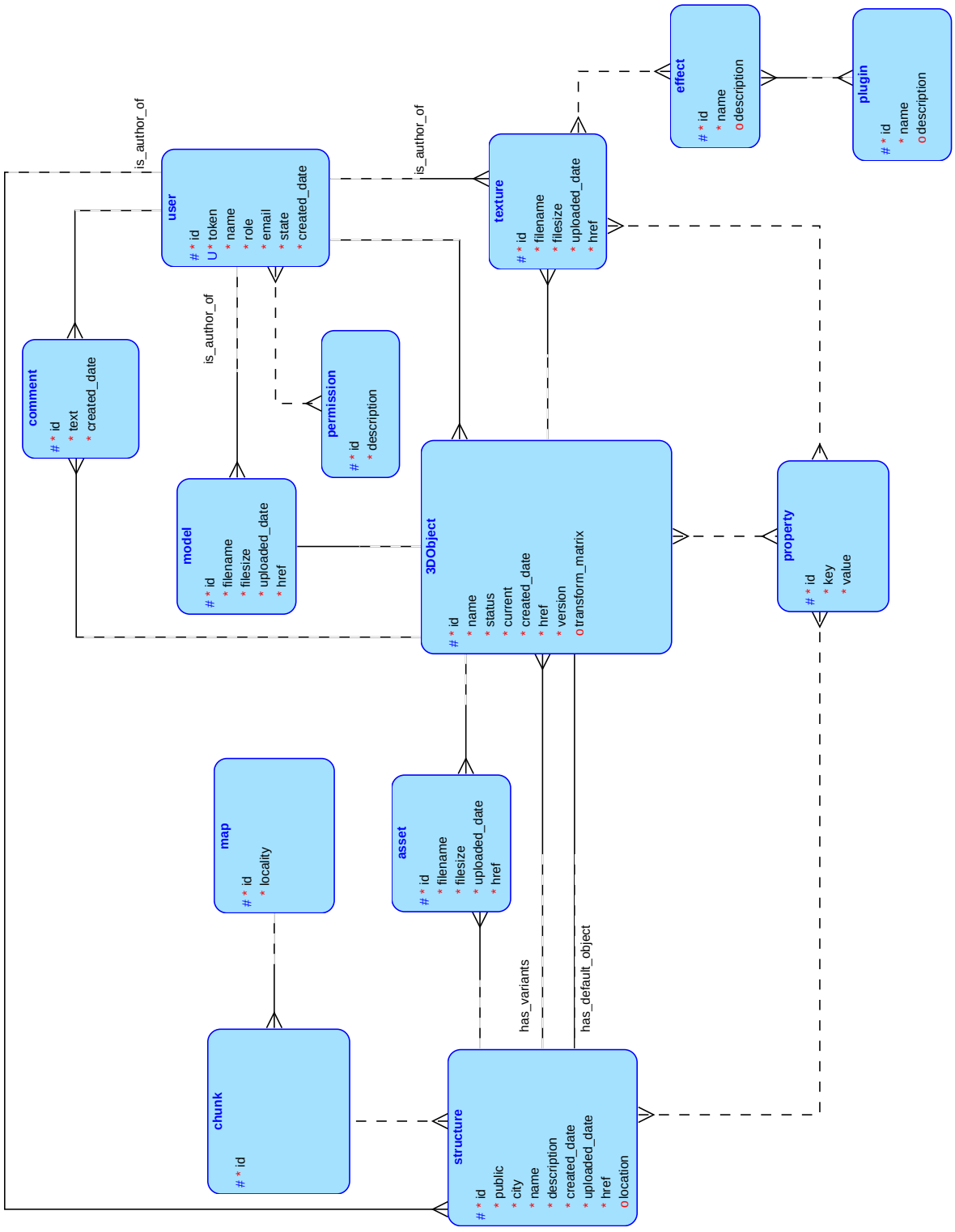
4.1 Databáze metadat

Po analýze předchozího řešení jsem vytvořil nové schéma datového úložiště (viz Obrázek 4.1). Hlavní rozdíly zde uvádím pro lepší čitelnost v bodech:

Odstranění tabulky *Field*: Jak již bylo zmíněno v předchozí kapitole, takto navržené pokládání budov na mapu pouze způsobuje větší nepohodlí uživatele. Po dohodě se zadavatelem práce byla tabulka *půdorysu* odstraněna.

Změna *House* na *3DObject*: Původní návrh předpokládal pouze pokládání modelů **budov** na mapu. V projektu předpokládáme i tvorbu jiných objektů, například soch, fontán a podobně. V neposlední řadě musí být v databázi ukládány i modely terénu, proto bylo pojmenování *House* zavádějící. Fakticky však bude tato tabulka v databázi uložena pod názvem *TDObject*, jelikož jmenné konvence v Postgresu neumožňují název tabulky začínající číslovkou.

Doplnění chybějících atributů a zavedení *properties* Do tabulek sloužících pro spojení s *MongoDB* byly přidány informace o daném souboru, a do tabulek jako *3DObject* a *structure* byly přidány názvy, popisy a další



Obrázek 4.1: Logické schéma navržené databáze

(vizte schéma 4.1). Pro všechny ostatní vyhledávací parametry byly vytvořeny *properties*, kde budou udržovány informace jako například LOD, roční období, počasí…Pro zavedení nových vyhledávacích parametrů pak stačí vytvořit nový záznam v tabulce *property* a jeho přiřazení k záznamům v jednotlivých tabulkách.

Nově vytvořené tabulky Nejdůležitější takovou je tabulka *structure*, sloužící pro „obalení“ *3DObjectu*. Vytvořením této tabulky byl odstraněn nedostatek „manuálního výběru dat“ – zde bude vytvořen záznam o daném historickém artefaktu a v tabulce *3DObject* pak budou jeho jednotlivé transformace, tj. stejná budova za různého počasí, nasvětlení, LOD a podobně. U každé této transformace pak budou přítomny jen soubory nutné pro její sestavení, a informace o tom, o jakou transformaci se jedná, bude uložena v přiřazených *properties*.

Dále byla vytvořena tabulka *asset* za účelem ukládání konfiguračních souborů a upravena tabulka *texture*. Obě slouží stejně jako *model* pro spojení s *MongoDB*, ale z důvodu rozdílného účelu souborů byly od dané tabulky odděleny. Jednotlivé sloupce jsou na první pohled identické, a i když se dají *filename* a *filesize* také považovat za vyhledávací parametry, tj. lze je přesunout do *properties*, byly pro jednodušší manipulaci s daty ponechány v jednotlivých tabulkách.

Účel tabulek *user* a *permission* je zřejmý z jejich pojmenování, kde obě jsou zatím ve značně minimalistické formě. U naprosté většiny vytvářených dat očekáváme uložení údajů o autorství – užitečné pro realizaci zpětné vazby a vyhledávání. Uživatel může mít prozatím pro jednoduchost jen jednu roli a případně mu můžou být pro lepší přizpůsobení dané situaci dodána jednotlivá oprávnění (*permissions*).

Oproti nim tabulka *comment* na první pohled se zobrazením 3D objektů nesouvisí. Možnost komentovat jednotlivé 3D objekty bude využita ve schvalovacím procesu řečených objektů, kde historik/grafik do komentáře uvede modeláři požadavky na jejich úpravu. Prozatím tabulka obsahuje pouze záznam o autorovi, čas vytvoření a samotný text komentáře, ale v budoucnu očekáváme její rozšíření – více v sekci 4.4.

Poslední nově vytvořené jsou *effect* a *plugin*. Tyto tabulky slouží pro uložení informace o tom, jak daná textura vznikla – který zásuvný modul (např. do *Blenderu*) byl použit a jaká sekvence transformací vedla ke konkrétní podobě textury. Tyto informace mají sloužit modelářům k usnadnění práce a k porovnávání jednotlivých pluginů. Prozatím jsou tyto tabulky (podobně jako *chunk* a *map*) ve formě *placeholderů* a jejich rozšíření očekávám v dalších pracích.

Verzování Oproti původnímu návrhu, kde šlo vytvářet pouze nové verze *Modelů*, tyto verze byly pojmenovány a *House* obsahoval odkaz na aktu-

ální verzi, bude nově umožněno vytvářet nové verze celého *3DObjectu* (např. použitý různý set textur, či rozpracované verze během modelování). Každý *3DObject* bude obsahovat odkaz na jeho starší i novější verzi, aktuální verze bude označena příznakem *current=true* a pojmenování verzí bude ve formátu „*major.minor.patch*“. Verzování bude více rozebráno v sekci 4.3.2.

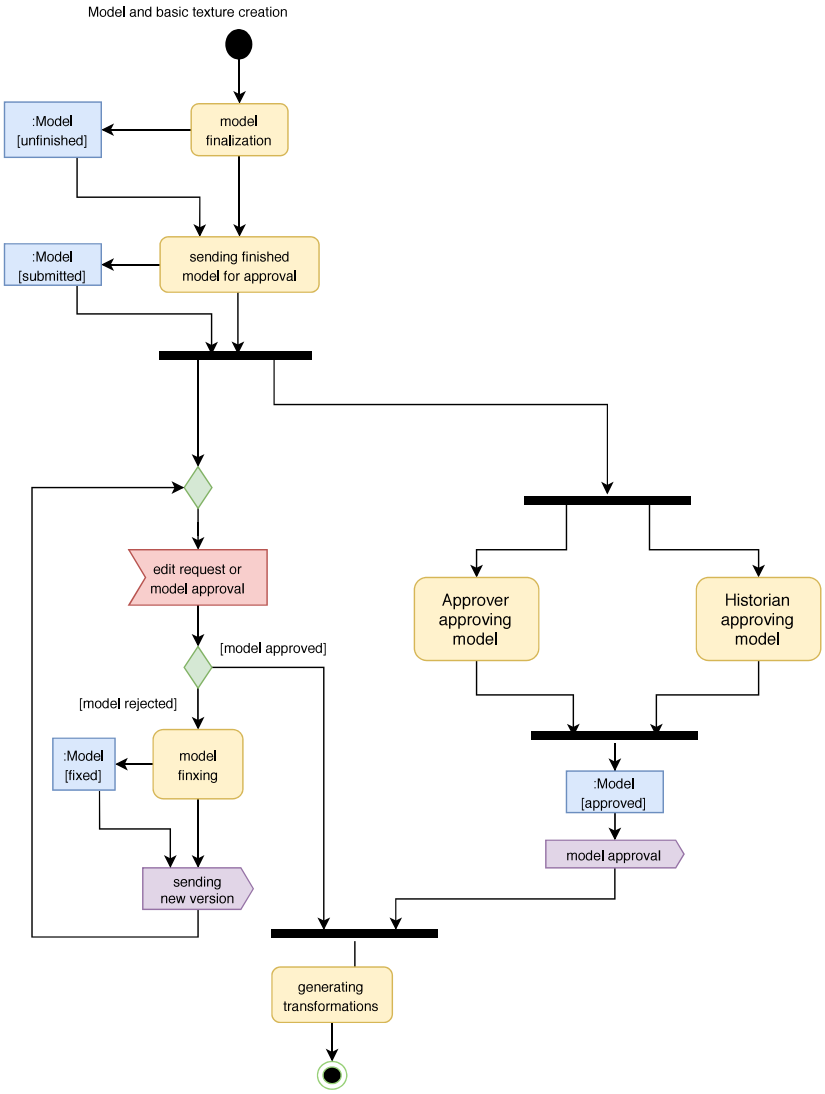
4.1.1 Schvalovací proces

Po vytvoření 3D modelů daných historických artefaktů je třeba je zkontrolovat pro jejich ať už historickou či grafickou správnost předtím, než budou zobrazovány koncovým uživatelům. Obsah této sekce byl vytvořen ve spolupráci s Ing. Michalem Martinkem, který vytvářel k projektu administrační rozhraní, jehož podstatná část je právě schvalovací proces [13], a konzultován s Ing. Jiřím Chludilem.

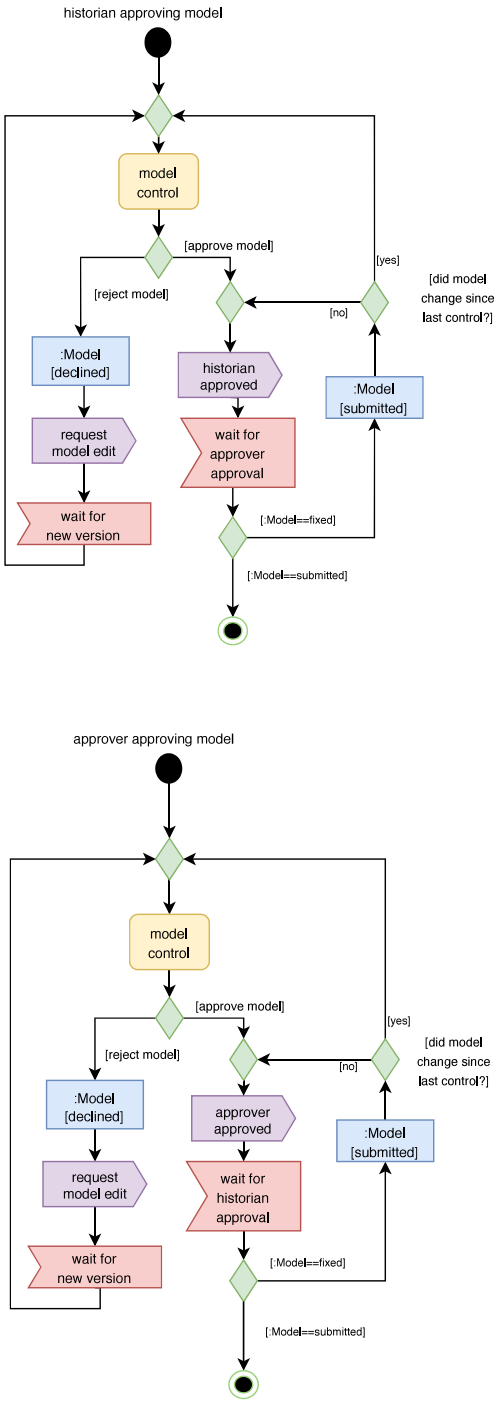
Nejprve *historik* vytvoří záznam v tabulce *structure*, se všemi potřebnými detaily, popisem, podklady pro modelování a podobně. Poté, co je s výsledkem spokojen, zadá *modeláři* za úkol daný objekt na základě požadavků vypracovat. Tímto se spouští modelování a poté schvalovací proces základní verze modelu, tj. obrázky 4.2 a 4.3.

Modelář daný objekt po jeho vytvoření odešle ke schválení. Zde vstupuje do procesu *grafik*, který kontroluje model z hlediska technické správnosti. Současně s ním *historik* provádí kontrolu, zda model odpovídá zadání a je historicky přesný. Model (3DObject) je považován za schválený pouze pokud oba zmínění model schválí – jakmile alespoň jeden z nich není s prací spokojen, je model s komentářem vrácen *modeláři* k přepracování. *„Nastane-li situace, kdy je jeden s výsledkem spokojen a druhý ne, tak ten, co je spokojen, nemusí provést opětovné schválení, a čeká se, až se spraví daná chyba mezi tím, co je nespokojen, a Modelářem, a až po opravě daných chyb je ten druhý vyzván k finálnímu schvalování. Toto má za úkol redukovat pracnost schvalovacího procesu.“* [13]

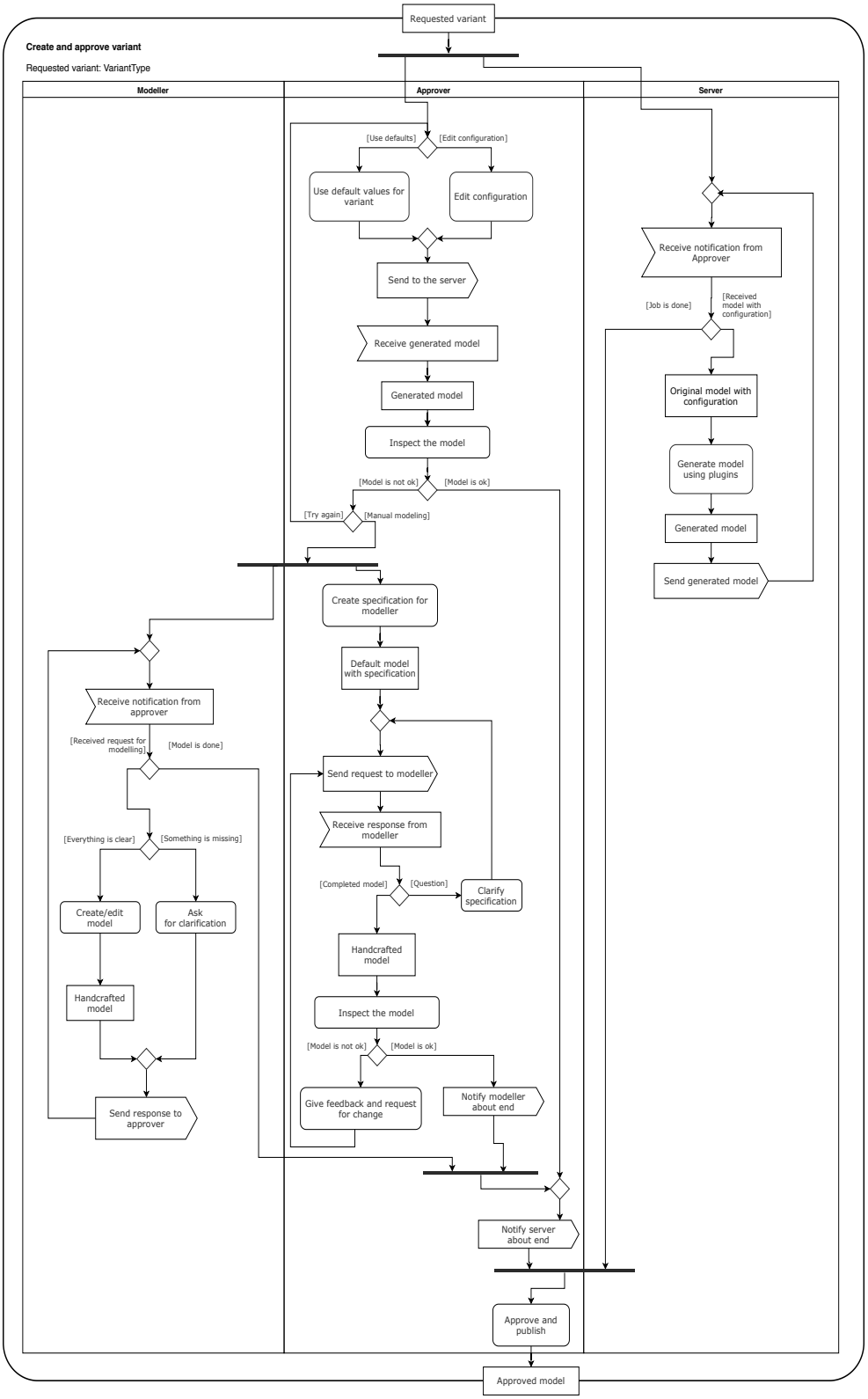
Jakmile je tento základní model schválen, je zveřejněn a spouští se fáze generování a následné schvalování jeho jednotlivých transformací (textury za různého počasí, nasvícení apod). Tuto část návrhu vytvářel Ing. Michal Martinek a schéma tohoto schvalování je vidět na obrázku 4.4. Generování daných transformací bude řešeno poloautomaticky – *grafik* vybírá vstupní parametry pro Blender pluginy a na serveru budou tyto pluginy z danými parametry spuštěny. Tvorba některých takových pluginů byla předmětem prací [20] a [21]. V případě, že *grafik* není s vygenerovanými variantami spokojen, pozmění parametry dříve zmíněných pluginů, a takto generuje znovu dokud není řešení v pořádku. Pokud usoudí, že daný plugin není schopen vygenerovat požadovaný výsledek, může znovu oslovit *modeláře*, který vytvoří daný model ručně. Po vytvoření dané varianty bude následovat zjednodušená varianta výše zmíněného základního schvalovacího procesu (schvaluje pouze *grafik*).



Obrázek 4.2: Schvalovací proces modelu I.



Obrázek 4.3: Schvalovací proces modelu II.



Obrázek 4.4: Schvalovací proces jednotlivé varianty [13]

Jednotlivé transformace budou schvalovány (a publikovány) nezávisle na sobě, nebudou tedy čekat na schválení ostatních.

4.2 MongoDB

V této databázi jsou ukládány jednotlivé soubory – tj. modely historických artefaktů, jejich textury, případné konfigurační soubory a v budoucnu očekáváme i další (GISy, zobrazitelné modely terénu, něco jako profilové obrázky uživatelů…). Kromě samotných souborů jsou zde ukládána metadata jako jméno a velikost souboru, datum jeho nahrání a md5 hash. Ukládaná data jsou a budou na schématu uvedena v příslušných tabulkách pro komunikaci s MongoDB, tj v tomto případě *model*, *texture* a *asset* na schématu 4.1. Jako přístupový klíč k jednotlivým záznamům jsou použity právě md5 hashe ukládaných souborů.

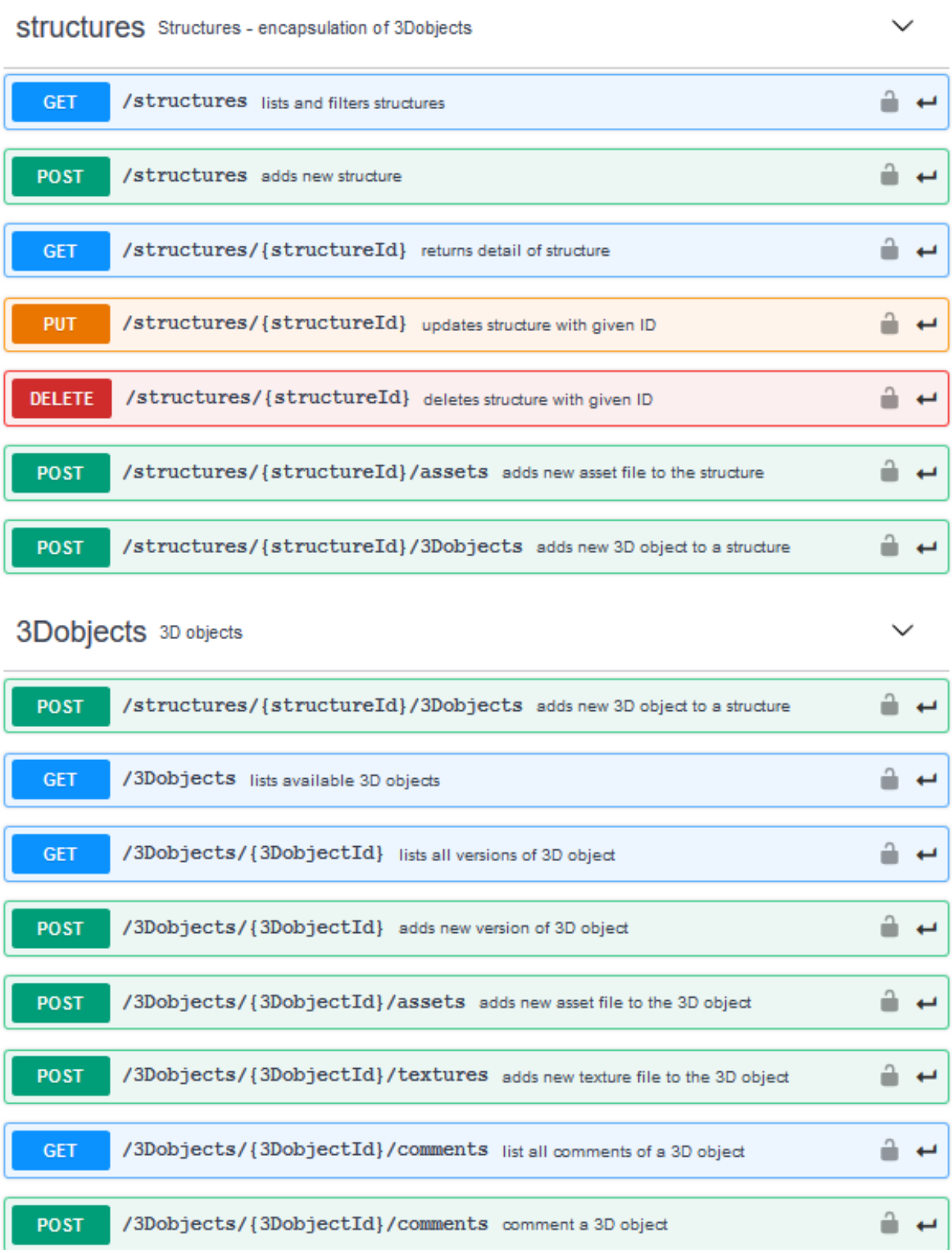
4.3 Privátní API

Hlavní (implementační) částí této práce je vytvoření *Privátního API* k dříve navrženému schématu databáze metadat. Toto API musí umožňovat základní CRUD operace nad navrženými tabulkami, obsahovat endpointy nutné pro dříve zmíněný schvalovací proces, převzít a dokončit předchozí vývoj a hlavně pokrýt požadavky zmíněné v kapitole 3.

Základní podoba tohoto API byla vytvořena a nasazena na server již během práce [12], protože bylo nutno umožnit modelářům alespoň základní ukládání dat. Má práce se tedy primárně snaží o převzetí a dokončení jak návrhu tak implementace práce [12]. Konkrétní stav při převzetí uvádím v kapitole 2.4 a v příslušných složkách na přiloženém médiu.

Při návrhu jsem se podobně jako předchozí práce nejvíce věnoval částem *structure* a *3DObject*, jelikož minimálně ze začátku projektu budou právě tyto nejvíce potřebné a využívané – *modelář* zde ukládá vytvářené modely a jejich verze, *historik* a *grafik* se věnují jejich schvalování či požadavkům na přepracování a následně jsou zde generovány a ukládány transformace jednotlivých modelů.

Na základě výše zmíněného jsem ve *swaggeru* vytvořil návrh struktury *Privátního API*, který podle mého odpovídá zmíněným požadavkům na tuto práci. Ukázku konkrétní podoby uvádím na schématu 4.5 – jako ukázka byly vybrány právě *endpointy* týkající se *3DObjectů* a *struktur*. Celá dokumentace a seznam a popis všech *endpointů* jsou uvedeny na přiloženém médiu ve složce */docs/api-end*.



Obrázek 4.5: Ukázka swagger dokumentace Privátního API

4.3.1 Tagování 3D objektů

V rámci lepší úpravy vyhledávacích parametrů jednotlivých objektů bez nutnosti při každé této příležitosti zasahovat do struktury API byla vytvořena tabulka *Property*. Tato tabulka uchovává jednotlivé *tagy*, které jsou uživatelem (a některé automaticky) přidělovány k záznamu v ostatních tabulkách. Mezi uživatelem přidělované tagy patří například údaje o počasí, denní či roční době, úrovni detailu a podobně. 3D objektu budou automaticky přidělovány tagy jako například formát uložených souborů.

Dále API umožňuje kompletní management těchto tagů, kde konkrétní způsob jejich přidávání či úprav zmiňuji v kapitole 5.3. Jedním z hlavních důvodů tvorby tohoto systému byla právě podpora vyhledávání, ale po dohodě s vedoucím práce byla implementace optimálního vyhledávacího algoritmu určena jako téma pro samostatnou závěrečnou práci, a tudíž vyřazena z rozsahu této.

4.3.2 Verzování

Pro podporu modelování a snadnější realizaci zpětné vazby byl vytvořen systém verzování 3D objektů. V API jsou tedy navrženy příslušné *endpointy* pro úpravu a vytváření nových verzí, případně pro získání celé historie daného záznamu. Konkrétní způsob tvoření a úpravy verzí, tj. jaký http request poslat na který *endpoint* popisuji podrobněji v sekci 5.3. Zde uvádím obecné myšlenky a způsob, jakým verzování 3D objektů funguje.

Jelikož nebylo žádoucí v rámci projektu znovu tvořit celý Git, mnou navržené verzování je poněkud jednoduššího charakteru. Navržená struktura udržovaných verzí má momentálně pro jednoduchost podobu spojového seznamu, jelikož stromovitá struktura historie objektu může při špatném použití značně snížit její srozumitelnost. Pokud se modelář pokusí vytvořit novou verzi na neaktuálním objektu, API pro zabránění tvorby stromovité struktury historie vrátí chybu (http status 400).

V případě, že by modelář chtěl vytvořit novou „větev“, je třeba vytvořit si odděleně nový samostatný 3D objekt, a veškeré další verze vytvářet na něm. Dále v případě, že více 3D objektů (ať už různých verzí téhož nebo spolu vůbec nesouvisejících) sdílí tytéž soubory (například textury), jsou v databázi v rámci minimalizace využití úložného prostoru tyto soubory sdíleny s využitím techniky počítané reference. Jelikož jako klíč k záznamům v MongoDB byl použit md5 hash, povede snaha o znovuuložení nezměněného souboru pouze k navrácení údajů o něm (a odeslání statusu „Úspěch operace“). V databázi metadat v tabulkách *model*, *texture* a *asset* jsou jako primární klíče použity stejné klíče jako v MongoDB (tzn. md5 hashe souborů), a tudíž jsou sdíleny mezi verzemi (a 3D objekty) i záznamy v těchto tabulkách, nejen záznamy v MongoDB.

Běžnému uživateli (v tuto chvíli modeláři) je v API povoleno upravovat a

mazat pro zjednodušení pouze poslední, tj. aktuální verzi, a to ještě pouze u záznamů, které sám vytvořil. Pouze uživatel s rolí *admin* má povoleno upravovat i starší verze 3D objektu. Dále byl u 3D objektu zaveden status *deleted* v případě jeho smazání namísto jeho fyzického odstranění z databáze. Tímto se zabrání scénáři, kdy je například modeláři ukončena spolupráce, a tak se rozhodne smazat veškerou svojí práci. Záznamy v daném stavu budou periodicky procházeny *daemonem*, který je bude fyzicky odstraňovat, a do té doby mohou být jejich autorem či *adminem* obnoveny.

4.3.3 Používané formáty souborů

Jak již bylo řečeno dříve, v rámci projektu předpokládáme ukládání objektu v jeho různých podobách, tj. například formátech. Jelikož ale nemůžeme předpokládat, že formáty souborů se nebudou periodicky měnit, bylo API navrženo nezávisle na nich. Konkrétní způsob, jak jednoduše změnit seznam podporovaných formátů uvádím v sekci 5.2.

V současnosti je API navržené tak, že podporuje následující formáty souborů:

blend: Formát používaný softwarem *Blender*. V tomto formátu předpokládáme primárně ukládání rozpracovaných verzí 3D objektů.

3ds a max: Nejčastější formáty využívané softwarem *3D Studio MAX*. Opět předpokládáme primárně pro ukládání nehotové práce.

obj a fbx: Nejběžnější a nejčastěji podporované formáty 3D souborů. V těchto formátech předpokládáme uložení již hotových prací a právě tyto jsou používány pro zobrazení modelu v AR či VR.

sfb: Předrenderovaná forma souboru (.fbx nebo .obj), kterou vytvoří systém android (za použití současných technologií na mobilním klientovi – Sceneform) při kompilaci aplikace a následně ji za chodu zobrazuje. V projektu předpokládáme v budoucnu kompilaci do těchto formátů již na serveru a koncové zařízení pak model už jen zobrazí.

sfa: Konfigurační soubor nutný pro kompilaci souboru do formátu **sfb**, opět úzce svázaný s aktuálně použitou technologií na mobilním klientovi.

jpg, jpeg a png: Prozatím zvolené formáty pro ukládání textur.

txt a config: Prozatím zvolené pro ukládání konfiguračních souborů (*assetů*).

4.4 Vize do budoucna

V této sekci uvádím funkcionality, se kterými se v projektu počítá, ale jejichž rozsáhlejší analýza je nad časové možnosti této práce. Tato sekce tedy slouží

jako inspirace pro navazující závěrečné práce či týmové projekty a snaží se ještě více přiblížit v projektu dříve provedené kroky a jejich úmysl.

4.4.1 Management uživatelů a přístupových práv

V projektu jsem navrhl pro začátek a jednoduchost jen pár endpointů pro správu uživatelů a přístupových práv. Momentálně se při vytvoření nového uživatele nastaví jeho status na „new“ a dokud není „schválen“ (admin pošle PUT request na příslušný endpoint se změněným stavem a/nebo seznamem práv) tak mu není přidělena ani role ani žádná přístupová práva.

Zároveň se během práce na projektu několikrát projednávaly myšlenky jako lokalizovaná tabulka uživatelů proti googlu (provázání práce na projektu s google účtem). V tomto účtu (administraci) si pak uživatel uloží svůj ssh klíč (podobně jako je řešen přístup na školním Gitlabu) a autentizace uživatele by probíhala přes tento klíč. V současnosti je pro jednoduchost použita autentizace přes JWT vygenerovaný při vytvoření uživatele. Případné jiné a robustnější řešení nechávám na dalších iteracích práce na API.

4.4.2 Transformace 3D objektů generované na serveru

Po vytvoření modelu očekáváme vygenerování jeho transformací (různá úroveň zasněžení, stárnutí budovy…) buď pomocí Blender pluginů (závěrečné práce [20] a [21]) nebo ručně *modelářem* (nástin v sekci 4.1.1). Prozatím jsou tyto transformace vytvářeny lokálně a nahrávány uživatelem.

V budoucnu očekáváme uložení daných pluginů a jejich konfigurací na serveru, kde si uživatel (například v administračním rozhraní [13]) navolí / změní danou konfiguraci, a server si tyto pluginy zavolá sám a jejich výsledek uloží jako novou verzi 3D objektu. Zároveň je třeba si kromě pluginu ukládat i seznam efektů, které byly na vygenerování dané varianty použity, a to včetně jejich pořadí.

Konkrétní podobu ukládaných dat, ukládání výše zmíněných pluginů, způsob jejich volání či ukládání výsledků předpokládám v budoucích závěrečných či týmových pracích.

4.4.3 Webové rozhraní pro schvalovací proces

Tato část projektu byla z velké části navržena a vytvořena Ing. Michalem Martinkem během jeho diplomové práce [13]. Zde časem očekávám kompletní rozhraní pro výše zmíněný schvalovací proces. Jakožto jeho rozšíření očekáváme v projektu možnost si v řečeném rozhraní daný 3D objekt zobrazit, a při nespokojenosti označit přímo nevyhovující místa na modelu. Za tímto účelem bude také nutné upravit dosavadní formu komentáře a příslušné endpointy v závislosti na konkrétní implementaci této funkcionality v administraci.

4.4.4 Uložení a zobrazení terénu

V projektu očekáváme mimo modelů různých historických artefaktů i ukládání 3D modelu terénu v dané oblasti (viz sekci 2.2). Pro uložení těchto informací byla navázána spolupráce s Historickým ústavem akademie věd, kteří slíbili tato data dodávat. Uložení těchto dat očekáváme v budoucnu ve formátu GISů v databázi MongoDB, a jejich obraz v *Postgresu* v tabulkách *map* (základní informace o dané lokalitě) a *chunk* (samotný GIS „rozřezaný“ na menší části). Rozdělení GISu na menší části očekáváme **například** formou na serveru uloženým Python scriptem (například software *Quantum GIS* využívaný na PřF UK výborně spolupracuje s řečenými scripty [27]), který se spustí při nahrání GISu celé oblasti, a sám uloží jednotlivé *chunky*.

Alternativním řešením problému ukládání GISů by bylo využít přímo rozšíření *Postgresu* – PostGIS, ale analýzu optimálního řešení tohoto problému ponechávám na budoucích závěrečných či týmových pracích.

4.4.5 Databáze lokalizačních informací

Všechna zařízení, ať už využívající AR či VR, si budou muset z databáze stahovat objekty na základě toho, kde se zrovna nacházejí. Řekněme uprostřed procházky městem se stahují jen objekty co jsou kolem, ne na druhé straně města. Tyto objekty je nutno zobrazit tak, aby naprosto přesně kopírovaly svou reálnou podobu, ale s nepřesnými údaji z GPS je to téměř nemožné.

Za tímto účelem je potřeba na serveru vytvořit modul umožňující přesnou lokalizaci zařízení na základě i jiných údajů než čistě GPS, například na základě fotografií. V projektu očekáváme funkcionalitu, kdy uživatel pošle fotografii s přibližnou polohou (GPS), a na základě triangulace z obrazu a přibližné polohy se dopočítá poloha přesná (nutné pro správné zobrazení objektů v AR) – pokud je daná fotografie nedostatečná, uživatel bude vyzván k odeslání další, například s instrukcemi, co má přesně vyfotit.

Aby takový modul mohl fungovat, je třeba vytvořit databázi například jednotlivých fotografií, metadat k nim a podobně. Tímto směrem ale rešerše zatím neprobíhala, neboť podobná funkcionalita se počítá spíše do závěrů projektu. Zároveň ale nebude příliš těžké potřebná data získat, protože téměř přesně toto mají kolegové z UHK na dříve zmíněné webové stránce.

4.4.6 Load balancing a synchronizace databází

Ve své práci se soustředím primárně na samotnou podobu databáze a příslušných API. Jak jsem již zmiňoval dříve, předpokládáme v projektu více instancí jak DB tak API běžících na samostatných serverech a synchronizovaných na pozadí. Po dohodě s vedoucím práce uvádím v kapitole 5.2 pouze návod na zprovoznění replikace Postgresové databáze vytvořený Bc. Vítem Černým během jeho bakalářské práce [11]. Reálné nasazení těchto funkcionalit očekávám v pozdějších iteracích, až bude API ve více vyvinutém stavu.

4.4.7 Vyhledávání

V předchozích kapitolách jsem zmínil požadavek na tvorbu vyhledávacího algoritmu, který by pracoval jak se sloupečky v příslušných tabulkách, tak s tagy jim přidělenými. Ve své práci tvořím pouze velmi jednoduchou formu filtrování daných objektů (autorství, status…) a po dohodě s vedoucím práce byla tvorba optimálního vyhledávání určena jako teoretické téma na samostatnou závěrečnou práci.

4.4.8 Public API

Public API je míněno jako částečná kopie *Privátního API* s odstraněním funkcionalit jako úprava dat a přidáním nových jako je sběr zpětné vazby, výpočet, které modely se mají stáhnout, či teoreticky navigace po daném městě. Tato práce se návrhem a implementací soustředí na *Privátní API*, a z prvního zmíněného pak fakticky vytváří jen část pro stažení konkrétních modelů.

4.4.8.1 „Feedback API“

Tato část Public API je pouze abstraktní a její oddělení je čistě pro přehlednost a zvýraznění úmyslu jejího vytvoření. Očekáváme propojení Public API s nějakým *trackovacím* softwarem (například redmine), kde zpráva od koncového uživatele vytvoří (a přiřadí) k danému 3D objektu nové *issue*.

4.4.8.2 Navigace

Po vytvoření map vybraných věnných měst budou tyto mapy včetně všech vytvořených budov, cest, soch a podobně zobrazeny koncovým uživatelům. Při procházce daným městem očekáváme zájem uživatelů navštívit konkrétní místa či památky. Za tímto účelem začal již v počátcích projektu Ivo Strejc vytvářet ve své BP navigační modul [7], jehož rozšíření a konkrétní nasazení předpokládáme ke konci projektu.

Zároveň uvažujeme možnost předpřipravit si doma trasu, kterou si uživatel bude chtít projít (a zároveň tedy například na wi-fi stáhnout nutné 3D objekty předem), v administraci vytvářet trasu doporučenou, případně dávat v aplikaci návrhy trasy na základě tras vytvořených ostatními uživateli dříve.

Tyto funkcionality očekáváme ale skutečně až v závěru projektu, a dříve zmíněné mají z tohoto pohledu rozhodně přednost.

4.4.9 Optimalizace mobilních zařízení

Jak jsem již psal na začátku práce, v rámci projektu jsem spolupracoval s kolegy Štěpánem a Slabým, kteří se zabývají mobilním klientem.

Protože mobilní zařízení obecně nemají takový výkon jako počítače a servery, snažíme se na nich minimalizovat výpočetní úkony. Zároveň na mobilních

zařízeních existuje více modulů, které by aplikace mohla používat na umisťování objektů v AR. Každý takový modul má různé parametry, a zjištění optimálního modulu pro dané zařízení není triviální.

Kolega Ing. Ondřej Slabý pak přišel s návrhem, že jakmile se pro dané zařízení najde optimální nastavení, uloží se do databáze společně s identifikátorem modelu (string, například „Honor H30-U3C“). Každé další zařízení pak nemusí toto nastavení počítat, ale stačí ho zjistit přes API.

Kapitola 5

Realizace

V této části práce uvedu příručky pro jednotlivé uživatele v projektu. Nejprve zde uvedu návod pro vývojáře, jak si lokálně „rozchodit“ projekt, poté vybrané případy, jak jednoduše upravit chování API a nakonec návrh pro uživatele ostatních modulů, jak tyto moduly napojit na mnou vytvářené API.

5.1 Instalační příručka

5.1.1 Převzetí vývoje

Struktura *Privátního API* byla již během práce [12] navržena s ohledem na co nejsnazší převzetí jeho vývoje. Za tímto účelem bylo pro vývoj vybráno prostředí docker (popis v kapitole 2.5) a až na drobné opravy je návod na zprovoznění stejný jako ve zmíněné práci.

Nejprve nainstalujeme samotný docker podle kroků zmíněných na [28]. Pro jednoduchost předpokládáme vývoj na OS Linux a instalaci provádíme příkazem:

```
sudo apt-get install docker docker-compose
```

Po instalaci dockeru (a jeho případné aktualizaci na nejnovější verzi pokud se stáhla nějaká stará) si do námi zvolené složky naklonujeme repozitář se zdrojovými kódy API:

```
git clone git@gitlab.fit.cvut.cz:vmck/core/private-api.git
(případně https://gitlab.fit.cvut.cz/vmck/core/private-api.git
a vyplnění přístupových údajů, pokud nechceme řešit ssh klíče)
```

Vstoupíme do naklonované složky (*private-api*) a následně ve složce */config* vytvoříme soubory *.env* a *.env.test* podle instrukcí v *.env.example*. Zde uvedeme jednotlivé proměnné prostředí, jako například přihlašovací údaje do databází (MongoDB a Postgres) – pro konkrétní hodnoty kontaktujeme ostatní vývojáře nebo vedoucího projektu (v tuto chvíli zadavatel této práce).

Po korektním vytvoření daných souborů již stačí v kořenovém adresáři *private-api* spustit příkaz:

```
docker-compose up
(pokud chceme v terminálu vidět výpisy dockeru)
případně
docker-compose up -d
(pokud chceme, aby docker běžel na pozadí)
```

Docker si v tuto chvíli do daného *kontejneru* stáhne potřebné závislosti (uvedené v souboru *package.json*) a po (automatickém) spuštění kontejneru je prostředí připravené pro vývoj. V případě, že tento příkaz zahlásí chybu *„Coundn't connect to docker daemon at http+docker://localhost…“*, zkusíme spustit příkaz pod admin přístupem (*sudo*) – jednou z možných příčin této chyby jsou nedostatečná oprávnění uživatele.

5.1.2 Testování

V rámci API byly vytvářeny už během práce [12] i automatizované testy. Pro tvorbu těchto testů byl již v předchozích letech vybrán velmi jednoduchý Javascript framework – Jest.

Veškeré soubory s vytvořenými testy jsou v repozitáři uloženy v adresáři */tests*, a podobně jako většina vytvořené implementace se soustředí primárně na endpointy, které se přímo týkají ukládaných 3D objektů, modelů a textur.

Pro spuštění jednotlivých testů pak používáme:

```
docker-compose exex private-api yarn test {testFile}
```

kde *testFile* je název souboru (ve formátu XYZController) ze složky */tests* a bez přípon *.test.ts*.

Před začátkem samotného vývoje výrazně doporučuji si skutečně přečíst soubor *README.md*, kde uvádím problémy, na které jsem v rámci vývoje narazil (část „Known Issues“) – jako příklad uvedených „good-to-knows“ je nutnost spouštění testů po jednom, jelikož spuštění všech testů najednou (tzn. neuvedení názvu souboru s testy) vede k *failu* všech.

5.1.3 Continuous integration

„*Průběžná integrace* (dále jen CI) *je metodika využívaná při vývoji software, kdy vývojáři pravidelně provádějí merge do centrálního repozitáře, následkem čehož se spustí automatické buildy a testování. …Hlavním účelem a výsledkem tohoto přístupu je snazší a rychlejší hledání chyb, lepší stabilita kvality softwaru a jeho rychlejší validace a vydání nových verzí.*“ [29] Součástí zadání mé práce je mimo jiné i použití zmíněné technologie.

Na světě existuje velké množství nástrojů obstarávajících CI. Například stránka [30] jich uvádí přes 50. Výběr pro projekt optimálního nástroje byl

původně zamýšlen v rozsahu práce [12] a v tomto se skutečně spoléhalo na její dokončení. Analýza a srovnání jednotlivých nástrojů nebyla z časových a jiných důvodů zařazena do rozsahu této práce, a vývoj v tomto směru se tedy skutečně zasekl.

Základní předpoklady pro případné snadné zavedení CI jsou v projektu již splněny – existuje centrální repozitář, fungování projektu předpokládáme na více serverech a pro vývoj byl využit Docker (určeno v práci [12]). Například pro zprovoznění jednoho z nástrojů (GitLab CI/CD) pak stačí téměř jen krátká konfigurace daného repozitáře a vytvoření konfiguračního souboru *.gitlab-ci.yml* podle příslušné dokumentace [31].

Jedním z hlavních důvodů nevytvoření fungujícího CI v mé práci je právě v předchozí kapitole zmíněná chyba, kdy je nutné testy spouštět jednotlivě, jinak dojde k neúspěchu všech, kterou se mi během vývoje nepodařilo odstranit. Ve chvíli, kdy testování nefunguje bez chyb ani při lokálním spuštění, mi přišlo spouštění testů na vzdáleném serveru zbytečné.

Navzdory tomu, že jdou testy spouštět pouze jednotlivě, byla vytvořena jejich rozsáhlá sada ověřující funkčnosti API a jednotlivých *endpointů*. Více o testování je uvedeno v předchozí kapitole.

5.1.4 Nasazení API na server

Jak již bylo zmíněno v sekci 5.1.1, práce [12] tvořila podobné procesy tak, aby byly co nejjednodušší, a moje práce se nezabývala jejich přetvořením.

Díky těmto skutečnostem je samotná instalace (předpokládáme-li manuální nakopírování dat na server) velmi jednoduchou záležitostí. Konkrétně pak pro nasazení API stačí úplně stejné kroky jako v dané kapitole, tj.

```
sudo apt-get install docker docker-compose; (plus případný update)
git clone {repozitář};
cd private-api;
docker-compose up (-d);
```

a API je zprovozněno a reaguje na http requesty (testováno pomocí softwaru *RESTer*).

5.2 Příručka pro vývojáře

Zde pro usnadnění budoucího vývoje uvádím pár vybraných *use-cases*, respektive předpokládaných úkolů pro vývojáře. Nepředpokládám, že tato sekce připraví budoucího vývojáře na vše (vzhledem k tomu, že určité části jsou poněkud obecné), ale uvádím alespoň základ.

U1 Vytvoření / Upravení tabulky a endpointů: Pro snazší úpravy databáze a ukládání historie těchto změn byly již během práce [12] zpro-

vozněny *migrace*. Po vytvoření či úpravě souboru ve složce */src/entity* po vzoru dalších zde uvedených souborů stačí spustit příkaz

```
docker-compose exec yarn migration:generate {nameOfMigration}
```

výsledkem čehož je vytvoření nové migrace ve složce */src/migration*. Po její kontrole (doporučuji) spustíme příkaz

```
docker-compose exec yarn migration:run
```

a změny budou uloženy do databáze. Pro navrácení DB do předchozího stavu pak slouží příkaz

```
docker-compose exec yarn migration:revert.
```

Pro uložení nové či upravení stávající doménové logiky použijeme soubory ve složce */src/models*, a pro definice jednotlivých endpointů slouží */src/controllers*. Jednotlivé url jsou uloženy ve */src/routes/Urls.ts* a v *Controllerech* jsou jen importovány. Hlavním důvodem volby této struktury bylo odstranění kruhové závislosti mezi *Controllery*, kde byly dané url jako konstanty uloženy původně.

U2 Změna podporovaných formátů: Tato část byla již v práci [12] navržena velmi jednoduše a má práce ji téměř neupravovala. Seznam podporovaných souborů je k nalezení v souborech */src/utils/Extensions.ts* a *mimetype* do hlavičky requestů v */src/utils/MimeType.ts*. Pro přidání případně odebrání podporovaného formátu pak stačí v příslušných *enumech* (TDObjectFormat a Extension) upravit jejich seznam a následně mapování v MimeTypeTDFormatsRelations a ExtensionTDFormatsRelations. Pro lepší vizualizaci jednoduchosti této úpravy uvádím ukázku kódu 5.1.

U3 Manuální úprava dat v DB: Pro případ, kdy je při vývoji třeba zasáhnout do databáze manuálně, tj. nestačí logika obsažená v API, byly pro obě databáze připraveny admin přístupy. Obě tyto „administrace“ se spouští automaticky současně s celým API (sekce 5.1.1) a jejich detaily jsou uvedeny v souboru *docker-compose.yaml*.

Pro Postgres se spustí na portu 3001 *dockette/adminer:dg* a pro Mongo na portu 3002 *mongoclient*. Pro přístupové údaje k administracím je opět nutno kontaktovat ostatní pracovníky či vedoucího projektu.

Případně lze do obou příslušných kontejnerů vstoupit přes příkazovou řádku (např pro Postgres: *docker-compose exec postgres psql* za předpokladu, že username i heslo k dané databázi je uložené v .env souboru), ale přístup přes dané webové rozhraní mi přišel podstatně intuitivnější.

MimeType.ts 717 Bytes

```ts
export enum MimeType {
    TDS = 'application/x-3ds',
    BINARY = 'application/octet-stream',
    BLENDER = 'application/x-blender',
    JPG_IMAGE = 'image/jpeg',
    MULTIPART_FORM_DATA = 'multipart/form-data',
    OBJ = 'application/x-tgif',
    PNG_IMAGE = 'image/png',
}

export enum TDObjectFormat {
    TDS = '3DS',
    BLEND = 'BLEND',
    FBX = 'FBX',
    MAX = 'MAX',
    OBJ = 'OBJ',
    SFB = 'SFB',
}

export const MimeTypeTDFormatsRelations = {
    [TDObjectFormat.TDS]: MimeType.TDS,
    [TDObjectFormat.BLEND]: MimeType.BLENDER,
    [TDObjectFormat.FBX]: MimeType.BINARY,
    [TDObjectFormat.MAX]: MimeType.BINARY,
    [TDObjectFormat.OBJ]: MimeType.OBJ,
    [TDObjectFormat.SFB]: MimeType.BINARY,
};
```

Obrázek 5.1: Způsob uložení podporovaných formátů 3D objektů

U4 replikace / duplikace: Tato sekce byla kompletně vytvořena Bc. Vítem Černým během jeho Bakalářské práce [11] a zde ji uvádím jen pro úplnost textu.

Po nainstalování databáze je třeba zavést replikaci. Všechny příkazy bychom měli vykonávat jako postgres uživatel. Začneme tím, že na master serveru vytvoříme uživatele replication, který bude mít REPLICATION práva.

```
psql -c "CREATE ROLE replication WITH REPLICATION PASSWORD 'heslo '
LOGIN"
```

Nyní upravíme soubor pg_hba.conf, a přidáme někam (jinam než na konec) řádek s IP adresou otroka.

```
host replication rep IP_adresa_otroka /32 md5
```

a opět také soubor postgresql.conf. Odkomentujeme a přepíšeme následující řádky:

```
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'
max_wal_senders = 1
hot_standby = on
```

A restartujeme službu:

```
service postgresql restart
```

Nyní se přesuneme na otroka. Nejprve otroka zastavíme:

```
service postgresql stop
```

Stejně jako v předchozím případe je třeba přidat tento řádek do pg_hba.conf

```
host replication rep IP_adresa_mastera /32 md5
```

a znovu upravit postgresql.conf

```
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'
max_wal_senders = 1
hot_standby = on
```

Konfiguraci již máme téměř hotovou, zbývá už jen prvotní nakopírování hlavní databáze z mastera na otroka. To uděláme následujícími příkazy vykonanými na master serveru. Nezapomeneme přepsat IP adresu otroka:

```
psql -c "select pg_start_backup('initial_backup');"
rsync -cva --inplace --exclude=*pg_xlog* /etc/postgresql/9.4/main/
IP_adresa_otroka :/etc/postgresql/9.4/main/
psql -c "select pg_stop_backup();"
```

Nakonec už jen zbývá vytvořit recovery složku na otrokovi. Přejdeme tedy do složky postgres

```
cd /etc/postgresql/9.4/main
```

a zde vytvoříme nový soubor s názvem recovery.conf a obsahem:

```
standby_mode = 'on'
primary_conninfo = 'host=IP_adresa_mastera port=5432
user=replication password=heslo '
trigger_file = '/tmp/postgresql.trigger.5432'
```

Nyní již zbývá jen spustit otroka:

```
service postgresql start
```

A replikace je zprovozněna. [11]

5.3 Příručka administrátora

Zde uvádím návod pro vývojáře ostatních částí projektu na to, jak mají své moduly napojit na *Privátní API*. Podobně jako v předchozí sekci zde uvádím pouze z mého pohledu nejdůležitější *use-cases*. Pro podrobnější informace o chování API vizte swagger dokumentaci na přiloženém médiu.

U1 Založení nového uživatele: Jako intuitivní začátek veškeré práce na projektu uvádím jako první způsob tvorby nového uživatele. Oproti předchozím iteracím práce na API, kdy bylo nutné tuto aktivitu provádět manuálním zásahem do databáze, byly vytvořeny příslušné *endpointy*, a pro samotné vytvoření nového uživatele pak stačí poslat POST request na endpoint */user* obsahující požadované *username* a *email*. Vytvořený uživatel je ve stavu *new* a dokud nebude manuálně schválen *adminem* (PATCH na */user/{username}* s přidělenou rolí a právy), nemůže daný uživatel v projektu dělat nic.

V budoucnu předpokládám automatizovanější proces tvorby nového uživatele, například formou potvrzovacího emailu s odkazem na aktivaci účtu. S tímto výhledem je u uživatelů *email* vyžadován, ale případná implementace této *featury* je skutečně až v daleké budoucnosti.

U2 Napojení Blender pluginu [6]: Následující *use-cases* slouží modeláři k propojení jeho vývojového prostředí s *Privátním API*. Opět zde uvádím pouze nejdůležitější kroky, pro ostatní je zde swagger dokumentace.

U2.1 Vytvoření nového 3D objektu: V první řadě je třeba si uvědomit, že všechny 3D objekty jsou zastřešené tabulkou *Structure*. Pokud tedy neexistuje záznam o struktuře, kterou *modelář* vytváří, je nejprve potřeba tento záznam vytvořit odesláním POST requestu na endpoint */structures*.

Požadovaná data pro vytvoření struktury jsou následující:

```
{
"public": boolean, - údaj o tom, jestli má být model viditelný,
nebo jde pouze o testovací model
"city": string, - umístění struktury
"name": string, - název, např. "Věž Kropáčka"
"description": string,
"location": JSON obsahující GPS souřadnice struktury
ve formátu: { "lat": number, "lon": number, "alt": number },
"properties": pole tagů, které chceme struktuře přidělit
}
```

Po vytvoření struktury pak stačí pro tvorbu nového 3D objektu odeslat POST request na endpoint */structures/{structureId}/3Dobjects*, obsahující název vytvářeného 3D objektu a samotný model. Jako nepovinné je zde pak opět pole přidělených tagů, textur či konfiguračních souborů (*assets*).

První 3D objekt takto přidělený ke struktuře je brán jako defaultní a s nejvyšší úrovní detailu. Z něj budou pak vytvářeny jeho další transformace a tento model je také jediný, který bude kontrolován z hlediska historické správnosti. Ostatní 3D objekty přiřazované k dané struktuře jsou brány pouze jako transformace tohoto základního, a budou tedy kontrolovány už jen grafikem.

U2.2 Vytvoření nové verze: Samotný akt vytvoření nové verze je dost jednoduchý – stačí poslat POST request na endpoint */3Dobjects/{3DobjectId}* se stejnými daty jako při založení nového.

Samotnému verzování jsem se věnoval již dříve v kapitole 4.3.2 a před napojením a použitím výrazně doporučuji (znovu) přečíst.

U2.3 Výpis historie: Tato *featura* byla vytvořena právě pro podporu vývoje a umožnění modelářům navrácení se k předchozí práci v případě chyby.

Pro získání celé historie stačí poslat GET request na *endpoint* zmíněný v předchozím *use-case*. Tato historie obsahuje i novější i starší verze daného objektu, vždy seřazeny podle data jejich tvorby. Pro získání pouze konkrétní verze pak slouží GET na */3Dobjects/{3DobjectId}/{version}*, kde *version* je string ve formátu „major.minor.patch“.

U2.4 Pokračování ve vývoji modelu: V případě, že u sebe modelář nemá soubor (model, texturu…) potřebný k vývoji, ale předtím si ho uložil do DB (neuložil si ho lokálně, případně ho již smazal), stačí si zjistit z historie 3D objektu id požadovaného souboru.

Zaslání GET requestu na */models/{id}*, případně */textures/{id}* či */assets/{id}* v závislosti na jeho typu, zahájí stahování požadovaného souboru.

Tímto způsobem lze i navázat na práci v předchozích verzích souboru, stačí použít id souboru z verze, na kterou chce modelář navázat. Případné další verze souboru pak ale musí být navázány na nejnovější 3D objekt, zdůvodnění v kapitole 4.3.2.

U2.5 Přiřazení tagů: API umožňuje k záznamům v tabulce 3DObject přidávat různé tagy. Primární účel této featury má být vyhledávání, jak jsem již v této práci popisoval dříve.

Samotný akt úpravy přiřazených tagů se provádí stejně, jako úprava ostatních údajů u 3D objektu, tj. zasláním PUT requestu na */3Dobjects/{3DobjectId}/{version}*, kde *version* musí odpovídat nejnovějšímu 3D objektu, jelikož pouze *admin* má oprávnění upravovat neaktuální verze souborů. Tento request musí obsahovat veškeré údaje, které jsou o 3D objektu k dispozici – absence jakýchkoliv dat povede k chybě (http status 400).

Samotné tagy, které chce modelář k 3D objektu přidat, musí být uvedeny v poli *properties*, a pokud chce modelář vytvořit nové, musí tak nejprve učinit přes POST na */properties*, protože neexistující tagy (z pohledu databáze) budou ve zmíněném PUT requestu ignorovány.

U3 Schválení 3D objektu: V současnosti veškerá obsluha schvalovacího procesu (větší rozbor v kapitole 4.1.1) je řešena zasláním PATCH requestu na */3objects/3DobjectId/approval* obsahujícím roli, za kterou chce daný uživatel model ohodnotit (schválit či odmítnout), boolean s údajem, jestli uživatel model schvaluje a případný text s komentářem.

Historik může schvalovat pouze aktuální verze dříve zmíněných defaultních 3D objektů a to jen když jsou ve stavu *submitted* případně *fixed*. Volání příslušných endpointů při jiných stavech 3D objektů (případně při schvalování neaktuální verze) vede k chybě.

Grafik může oproti *historikovi* schvalovat i 3D objekty ve stavu *generated*, tj. byly vygenerovány transformace a čeká se na jejich schválení, jinak mají tito uživatelé podmínky totožné.

V případě neprázdné položky *text* v daném PATCH requestu bude s tímto textem vytvořen komentář k danému 3D objektu.

U3.1 Komentáře k 3D objektu: Kromě možnosti vytvoření komentáře během (ne)schválení 3D objektu zmíněné v předchozím *use-case* jsou tvorba (POST request) a úpravy (PUT request) komentářů obsluhovány enpointy */3Dobjects/{3DobjectId}/comments* případně *.../comments/{commentId}*.

Pro získání všech komentářů (všech verzí) 3Dobjektu stačí zaslat GET request na první zmíněný *endpoint*.

Závěr

Cílem mé práce bylo analyzovat použité řešení datového úložiště pro grafické modely a následně ho dle požadavků kladených na projekt rozšířit a optimalizovat. Dále bylo mým cílem převzít vývoj API zprostředkovávající komunikaci tohoto úložiště s jednotlivými moduly a navrhnout a implementovat v něm nové procesy nutné pro další vývoj projektu. Vedlejším cílem mé práce bylo vytvořit rozsáhlejší dokument vysvětlující hlavní myšlenky a směry projektu a tím usnadňující zapojení nových pracovníků.

Z důvodu posledního zmíněného cíle jsem se v analytické části své práce věnoval nejprve celému projektu obecně, vysvětlil jsem jeho hlavní myšlenky a směry, a uvedl jsem práce, které se projektu dosud věnovaly. Následně jsem v téže části rozepsal stav projektu v době počátku psaní této práce. Po tomto popisu jsem provedl rozbor stávajícího řešení a poukázal jsem na jeho zásadní nedostatky.

Na základě zmíněné analýzy jsem pak úspěšně navrhl nový prototyp komunikačního API odstraňující většinu nedostatků předchozího. Některé části návrhu (např. rozdělení mapy na menší kusy) si však zaslouží natolik složitější analýzu problému, že byly po dohodě se zadavatelem práce vyčleněny jako samostatné téma závěrečné či týmové práce a v mém návrhu jsou tedy pouze zmíněny. Zároveň jsem ve spolupráci s Martinem Púčalou, Ing. Michalem Martinkem a Ing. Jiří Chludilem navrhl v API nové procesy, jako například automatické generování transformací 3D objektu na serveru či několikrát zmiňovaný proces schvalování 3D objektů. K mnou vytvořenému návrhu komunikačního API jsem následně vytvořil swagger dokumentaci s jeho zamýšleným stavem.

Po dokončení výše zmíněného jsem částečně implementoval navržený prototyp komunikačního API. Při implementaci a optimalizaci jednotlivých funkcionalit jsem se kvůli zamýšlené podpoře modelářů primárně zaměřil na management ukládaných 3D objektů a jejich verzování. Z důvodu neočekávatelných okolností a tudíž menšího časového rozpočtu jsem bohužel nebyl schopen dokončit implementaci API do ideálního stavu a například obsluha přístupo-

vých práv by si zasloužila další rozvoj.

Závěrem bych rád řekl, že jsem cíle své práce splnil, i když nebyla samotná implementace z různých důvodů dotažena do ideálního stavu – došlo k rozsáhlé optimalizaci předchozích návrhů API i databází, modelářům je již umožněno plnit databázi daty, byly navrženy nové procesy a bylo určeno hned několik směrů a témat, kterými se projekt může v budoucnu zabírat. Zda-li je tento dokument dostatečným seznámením s projektem VMČK, a tudíž jestli jsem splnil i „svůj malý osobní cíl“ se pravděpodobně dozvím až v následujících letech podle toho, jak rychle vznikne potřeba vytvořit na začátku práce zmíněnou *VMČK wiki*.

Literatura

[1] Žára, J.: *Web-Based Presentations of Large Urban Scenes*. Czech Technical Univerzity – Faculty of Electrical Engineering, 2004, prezentace dostupná z: https://old.cescg.org/CESCG-2004/presentations/ZaraCESCG2004.pdf. Dostupné z: `https://old.cescg.org/CESCG-2004/papers/01_ZaraJiri.pdf`

[2] Bouchner, P.: *Data Acquisition for VIRTUAL OLD PRAGUE*. Czech Technical Univerzity – Faculty of Electrical Engineering, 2004. Dostupné z: `https://old.cescg.org/CESCG-2002/PBouchner/index.html`

[3] Smrček, A.: *Visualization of Prague Castle*. Diplomová práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, Praha.

[4] Žoha, V.: *Finalizace modelů ve Virtuální Staré Praze*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, Praha.

[5] Věnná města českých královen [online]. [cit. 29. 07. 2020]. Dostupné z: `https://www.kralovskavennamesta.cz/`

[6] Chludil, J.: *Exportní zásuvné moduly pro 2D a 3D grafická studia [online]*. 2019. Dostupné z: `https://kralovskavennamesta.cz/vystupy2019/dokumentace_exportni_system_2019.pdf`

[7] Strejc, I.: *Virtuální historický průvodce - navigační modul*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

[8] Borkovcová, M.: *Databáze pro 2D a 3D grafické objekty [online]*. 2019, [cit. 29. 07. 2020]. Dostupné z: `https://www.rvvi.cz/riv?ss=detail&h=RIV%2F62690094%3A18460%2F19%3A50016030`

[9] Borkovcová, M.; Majerík, F.: *API rozhraní a konektor mezi webovými a mobilními aplikacemi [online]*. 2019, [cit. 29. 07. 2020]. Dostupné

z: https://www.rvvi.cz/riv?ss=detail&h=RIV%2F62690094%3A18460%2F19%3A50016029

[10] Čapek, M.: *Věnná města českých královen - API pro předávání grafických modelů.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

[11] Černý, V.: *Virtuální historický prúvodce - jádro.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2018.

[12] Máca, J.: *Škálovatelná architektura mikroslužeb pro AR/VR aplikace – (nedokončeno).* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha.

[13] Martinek, M.: *Administrační rozhraní pro projekt Věnná města českých královen.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2020.

[14] Chludil, J.: *Optimalizace rekonstrukčních technik historických objektů s využitím metod softwarového inženýrství a nástrojů počítačové grafiky – (v realizaci).* Dizertační práce, Univerzita Hradec Králové, Ústav historie a archivnictví, Hradec Králové, 2020.

[15] Štěpán, J.: *Věnná města českých královen - Modul rozpoznání obrazu.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

[16] Melcher, L.: *Věnná města českých královen - Lokalizační modul.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2020.

[17] Pavlíková, H.: *Virtuální historický prvodce - modul virtuální reality.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2018.

[18] Křepínský, P.: *Věnná města českých královen - Úprava 3D modelu ve virtuální realitě.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

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

[20] Zajíc, M.: *Věnná města českých královen - úprava textur I.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

[21] Sůvová, D.: *Věnná města českých královen - úprava textur I.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2019.

[22] Tislický, J.: *Věnná města českých královen - úprava textur II.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2020.

[23] SwaggerHub API specification - Private API for the project Věnná Města Českých Královen [online]. [cit. 12. 07. 2020]. Dostupné z: `https://app.swaggerhub.com/apis-docs/VMCK/private-api/1.0.0`

[24] Express/Node introduction [online]. [cit. 12. 07. 2020]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction`

[25] TypeORM documentation [online]. [cit. 12. 07. 2020]. Dostupné z: `https://typeorm.io/`

[26] JWT documentation [online]. [cit. 12. 07. 2020]. Dostupné z: `https://jwt.io/introduction/`

[27] Laštovička, M. J.: *Úvod do GIS, QGIS 3D Modelace.* školení na PřF UK, 23. 3. 2018.

[28] Docker installation manual [online]. [cit. 25. 07. 2020]. Dostupné z: `https://docs.docker.com/compose/install/`

[29] Continuous Integration Explained [online]. [cit. 29. 07. 2020]. Dostupné z: `https://aws.amazon.com/devops/continuous-integration/`

[30] Top Continuous Integration Tools [online]. [cit. 29. 07. 2020]. Dostupné z: `https://stackify.com/top-continuous-integration-tools/`

[31] Getting started with GitLab CI/CD [online]. [cit. 29. 07. 2020]. Dostupné z: `https://docs.gitlab.com/ee/ci/quick_start/`

Dodatek A

Seznam použitých zkratek

API Application Programming Interface

VMČK Věnná města Českých královen

DB Databáze

BP Bakalářská práce

GIS Geografický informační systém

VR Virtual Reality / Virtuální realita

AR Augmented Reality / Rozšířená realita

UHK (FF) Univerzita Hradce Králové (Filosofická Fakulta)

JS Javascript

TS Typescript

JSON JavaScript Object Notation

JWT JSON Web Token

SWT Simple Web Token

SAML Security Assertion Markup Language Tokens

BSON Binary JSON

LOD Level of detail

CRUD Create, Read, Update, Delete

OS Operační systém

CI / CD Continuous Integration / Continuous Deployment

Dodatek B

Obsah přiloženého Média

- `readme.txt` ..... stručný popis obsahu média
- `docs` ..... adresář se Swagger dokumentací
  - `api-start` ..... stav při převzetí vývoje
  - `api-end` ..... zamýšlený stav při odevzdání BP
- `src`
  - `api-start` ..... zdrojové kódy Privátního API při převzetí vývoje
  - `private-api` ..... zdrojové kódy Privátního API při odevzdání
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF