FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Věnná města českých královen - Schvalovací systém pro 3D modelu ve virtuální realitě
Student:	Marek Klofáč
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Věnná města českých královen je projekt zabývající se zobrazením historického prostředí ve virtuální realitě.

1. Analyzujte dříve vytvořený schvalovací model v rámci projektu VMČK.
2. Analyzujte systémy virtuální reality a nástroje pro vývoj ve virtuální realitně v Unity3D.
3. Pomocí metod softwarových inženýrství navrhněte sady vhodných interakcí s historickými modely ve virtuální realitě se zaměřením na uživatelskou skupinu modelářů, grafiků a historiků.
4. Implementujte prototyp aplikace s navrženými interakcemi v Unity3D ve virtuální realitě.
5. Podrobte a vyhodnoťte prototyp vhodným uživatelským testům.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 4. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Věnná města českých královen - Schvalovací systém pro 3D modely ve virtuální realitě

Marek Klofáč

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Chludil

22. června 2021

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Jiřímu Chludilovi za pravidelné konzultace, skvělé nápady a zkušené rady. Poděkování patří také mé rodině za pevnou podporu během celého studia a při psaní této práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla 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 22. června 2021 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2021 Marek Klofáč. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Klofáč, Marek. *Věnná města českých královen - Schvalovací systém pro 3D modely ve virtuální realitě*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato bakalářská práce se zabývá analýzou, implementací a následným testováním dříve navrženého schvalovacího procesu pro kontrolu kvality 3D modelu od Ing. Michala Martínka. Vzhledem k tomu, že výsledkem je aplikace ve virtuální realitě, analýza nezbytně sestává z rozboru dostupných hardware zařízení podporující právě virtuální realitu a krátké shrnutí software možností, pro její realizaci, včetně nakonec zvoleného prostředí pro vývoj Unity 3D Engine a dodatečných SDKs (Software development toolkit).

Aplikace je součástí projektu Věnná města českých královen. Bude tedy využívat již dříve implementované privátní API, pomocí kterého bude načítat uložené historické modely z databáze a zobrazovat je uživatelům ve virtuálním prostoru. Ti budou schopni objektivně posoudit jeho kvalitu a autentičnost díky navrženým interakcím a buďto jej schválit nebo odevzdat na přepracování s poznámkami. Tato aplikace bude následně podrobena předpřipraveným uživatelským testům, které odhalí případné nepřesnosti v návrhu funkcionalit.

Klíčová slova Věnná města českých královen, Virtuální realita, Unity 3D, HTC Vive, historické modely, kvalita 3D modelu

Abstract

This bachelor thesis deals with the analysis, implementation and subsequent testing of a previously designed approval process for quality control of a 3D model from Ing. Michal Martínek. Given that the result is a virtual reality application, the analysis necessarily consists of available hardware devices supporting virtual reality and a brief summary of software options for its implementation, including the chosen environment for development Unity 3D Engine and additional SDKs (Software development toolkit).

The application is part of the Dowry Cities of Czech Queens project. It will therefore use the previously implemented private API, with which it will load stored historical models from the database and display them to users in virtual space. They will be able to objectively assess its quality and authenticity through the proposed interactions and either approve it or submit it for revision with comments. This application will then be subjected to pre-prepared user tests, which will reveal any inaccuracies in the design of functionalities.

Keywords Dowry Cities of Czech Queens, Virtual reality, Unity 3D, HTC Vive, historical models, 3D model quality

Obsah

Seznam obrázků

Úvod

Virtuální realita je technologie, která se poslední dobou dostává do podvědomí více a více lidem a díky lepším grafickým kartám a procesorům v novodobých počítačích si může i běžný uživatel vychutnat VR zážitek doma. Vývoj aplikací pro VR se postupně odklání od herního průmyslu k ostatním odvětvím. Jsou to právě velké korporátní firmy, které vidí případy užití napříč svým portfoliem a proto dnes vidíme mnoho projektů, jejichž výsledkem jsou specifické programy na míru.

Jedním z takových projektů jsou i *Věnná města českých královen* (dále VMČK). Jde o kolaborativní projekt, na kterém se podílí ČVUT FIT (České vysoké učení technické v Praze - Fakulta informačních technologií), UHK FF (Univerzita Hradec Králové - Filozofická fakulta), Ministerstvo kultury a další instituce. Mezi jeho hlavní cíle patří prezentace českého národního dědictví široké věřejnosti za pomoci historických podkladů a pokročilé počítačové grafiky (skenování a vizualizace v rozšířené a virtuální realitě). Výstupem tohoto projektu má být mimo jiné i historický průvodce věnnými městy.

Má práce je součástí už třetí iterace tohoto projektu a z částí navazuje na své předchůdce. Věnuje se rozboru předchozích řešení a návrhu nutných funkcionalit pro implementaci schvalovacího procesu 3D modelů, jenž má za úkol poskytnout validní model pro produkční použití v ostatních klientských aplikacích v rámci projektu VMČK.

Největší motivací pro výběr tohoto tématu je pro mě chuť posunout projekt věnných měst o krok dále a usnadnit budoucí vývoj podobných aplikací ve VR. Na projektu jsem působil v předchozích letech během předmětu *Softwarový projekt I a II* a v developerském týmu po dopsání této práce zůstávám. Ke zvolení tématu přispěla také skutečnost, že jsem měl předchozí zkušenosti se zvoleným vývojovým prostředím Unity Engine 3D.

Kapitola 1

Cíl práce

Hlavním cílem této bakalářské práce je úspěšně navhrnout a implementovat funkční aplikaci ve virtuální realitě, která bude umožňovat grafikům a historikům zkoumat kvalitu vytvořeného 3D modelu od modeláře a zároveň jim umožnit hodnotit jeho provedení či případné vrácení na přepracování.

V rámci projektu VMČK vznikají převážně modely historických staveb a artefaktů z pozdního středověku, z tehdejších českých věnných měst, jako byl například Hradec Králové nebo Polička. Ve výsledku se klade nejvetší důraz na věrohodnost a přesnost vytvořeného modelu. Je tedy nutné navrhnout vhodné interakce s těmito objekty ve virtuální realitě, které dovolí uživateli detailně analyzovat jeho samotnou strukturu a validitu použitých textur. Tyto interakce musejí být navrženy s ohledem na požadavky cílových skupin grafik a historik a zároveň na limitace hardware. Při jejich návrhu musí být dbáno na to, že pro některé uživatele jde o první setkání s VR, musejí být tedy co nejvíce intuitivní a nezpůsobovat žádné obtíže.

Pro vlastní návrh a implementaci bude prvně nutné analyzovat aktuálně dostupný hardware, který zpřístupňuje zážitek ve VR a rozbor softwarových řešení, který použiji pro vlastní realizaci. K samotné funkčnosti aplikace bude zapotřebí prozkoumat aktuální stav privátního API a schvalovací proces, což jsou výsledky předchozích prací v rámci projektu VMČK.

Po úspěšném postavení první funkční verze udělám její uživatelské testování, jehož výsledky ukáží, jak jsou jednotlivé cílové skupiny spokojeny s aplikací. Všechny případné nedostatky se pokusím zapracovat a opravit.

Kapitola 2

Analýza

V této kapitole se především zaměřím na rozbor aktuálního stavu privátního API (*Application Programming Interface*) již zmiňovaného projektu VMČK, které zpřístupňuje uložená data mé aplikaci, která bude výstupem této práce. Prozkoumám stávající navržené *endpoints* [1], abych si ujasnil, jak mohu s uloženými daty nakládat a podle toho navrhnout metody a interakce pro schvalování 3D modelů. Vzhledem k tomu, že privátní API je postavené nad serverovou databází, detailně rozeberu její interní reprezentaci dat a jednotlivé tabulky.

Součástí této kapitoly bude také průzkum již navrženého schvalovacího procesu, na kterém pracoval Ing. Michal Martínek [1]. Tento proces bude stěžejním bodem mé implementační části.

Vzhledem k tomu, že je tato práce zaměřena na tvorbu aplikace ve VR, analýza musí nutně obsahovat rozbor dostupného software pro vývoj a určení koncového zařízení pro uživatele. Nesmí samozřejmě chybět shrnutí doposud vymyšlených interakcí s 3D modely ve virtuálním prostoru, které jsou výstupem předchozích prací v rámci projektu VMČK. Tyto interakce využiji nadále v kapitole Návrh, kdy je doplním o další mé nápady a odpovídajícím způsobem je upravím, aby splnili všechna očekávání cílové skupiny.

2.1 Backend projektu VMČK

Na vývoji backendu v rámci projektu VMČK se podílelo historicky mnoho lidí, jako nejduležitější bych rád zmínil například Jindřicha Mácu s jeho diplomovou prací [2], Daniela Vančuru s jeho bakalářskou prací [3] nebo Dominika Siváka, který aktuálně pracuje na rozvoji backendu a privatního API v rámci diplomové práce [4] v době, kdy píši tuto práci.

[1] Endpoint je jeden konec komunikačního kanálu mezi klientem a serverem, kde na druhé straně je daný zdroj

2.1.1 Privátní API

API je rozhraní, pomocí něhož dokáží vzájemně komunikovat dva různé programy bez závisloti na tom, kde zrovna běží. V současnosti se využívá napříč všemi oblastmi, například v mobilních aplikacích, operačních systémech nebo na webových stránkách. Funguje na principu takzvaných požadavků a odpovědí, kde požadavek se vytvoří na straně klienta a odešle se přes HTTP-HTTPS protokol na server, který vrátí zpět příslušnou odpověď. Existuje samozřejmě mnoho specifikací, které blíže určují, jak takový požadavek a odpověď mají vypadat. Mezi nejznámější se řadí RPC (Remote procedure call), SOAP (Service object access protocol) nebo REST (Representational state transfer). Právě s příchodem RESTu se na přelomu tisíciletí rozmohli webové APIs a staly se velmi populárními. Na rozdíl od ostatních implementačních standardů nedbá tolik na detaily a strukturu prenášených dat, ale snaží se být především bezestavové, vrstevnaté, cachovatelné [2] a jasně odděluje hranici mezi klientem a serverem [5]. Přístup k uloženým datům poté umožňuje pomocí takzvaných zdrojů, kde každý má své unikatní URL (Uniform resource locator), na který se odesílají požadavky nějakou specifickou metodou (mezi nejznámější patří například GET, POST, PUT, PATCH...). Po zpracování našeho požadavku obdržíme od vzdáleného serveru odpověď, jejíž součástí mimo vlastní data jsou i metadata, například statusový kód signalizující, jak komunikace dopadla [6].

Jak jsem již zmiňoval v úvodu této kapitoly, v rámci projektu VMČK vzniklo privátní REST API zpřístupňující důležitá data uložena v databázi, jako například vlastní 3D modely, historie úprav, komentáře k úpravám, textury a uživatelé, včetně jejich přidělených rolí. Během návrhu byla využita OpenAPI specifikace, která jasně udává formát a strukturu souborů, povolené datové typy, vlastní schéma požadavků a odpovědí nebo návratových kódů[7]. Pro vlastní tvorbu byl využit nástroj Swagger, který mimo jiné dokáže automaticky generovat dokumentaci na základě kódu, vytvářet testovací scénáře pro ověření funkčnosti a hostovat webové rozhraní, pomocí kterého je možné provolávat jednotlivé implementované *endpoints*[8]. Samotná dokumentace celého API, respektive její nejnovější verze v době psaní této práce, se nachází na této webové stránce [3].

Na základě práce Daniela Vančury, která k tomu poskytuje stručný manuál, je možné si rozchodit vlastní verzi API serveru na svém lokalním počítači. Nevýhoda této varianty je, že chybí data, která by naplňovala databázi a je nejprve nutné vygenerovat sadu testovacích dat, které je možné později použít. Tato okolnost vedla Ing. Jiřího Chludila, vedoucího této práce, k nasazení API serveru na dedikovaný virtuální server v Cloudovém prostředí, který je dostupný odkudkoliv a po autorizaci pomocí autorizačního tokenu je možné

[2] Z anglického slova *cache*, umožňuje ukládání opakovaných požadavků

[3] Dokumentace API ke dni 16.6.2021: https://app.swaggerhub.com/apis/sivakdom/private-api/3.0.5/

přistoupit k uloženým datům. V budoucích iteracích projektu bude nejspíše upraven přístup k tomuto API serveru a poběží na jiné adrese, protože se stále jedná pouze o testovací prostředí [4]. Popis důležitých *endpoints*, které má aplikace z pohledu schvalování používá, najdete v kapitole návrhu 3.5, kde mimo jiné ukazuji, v jaké formě obdržíme uložená data.

2.1.2 Reprezentace dat

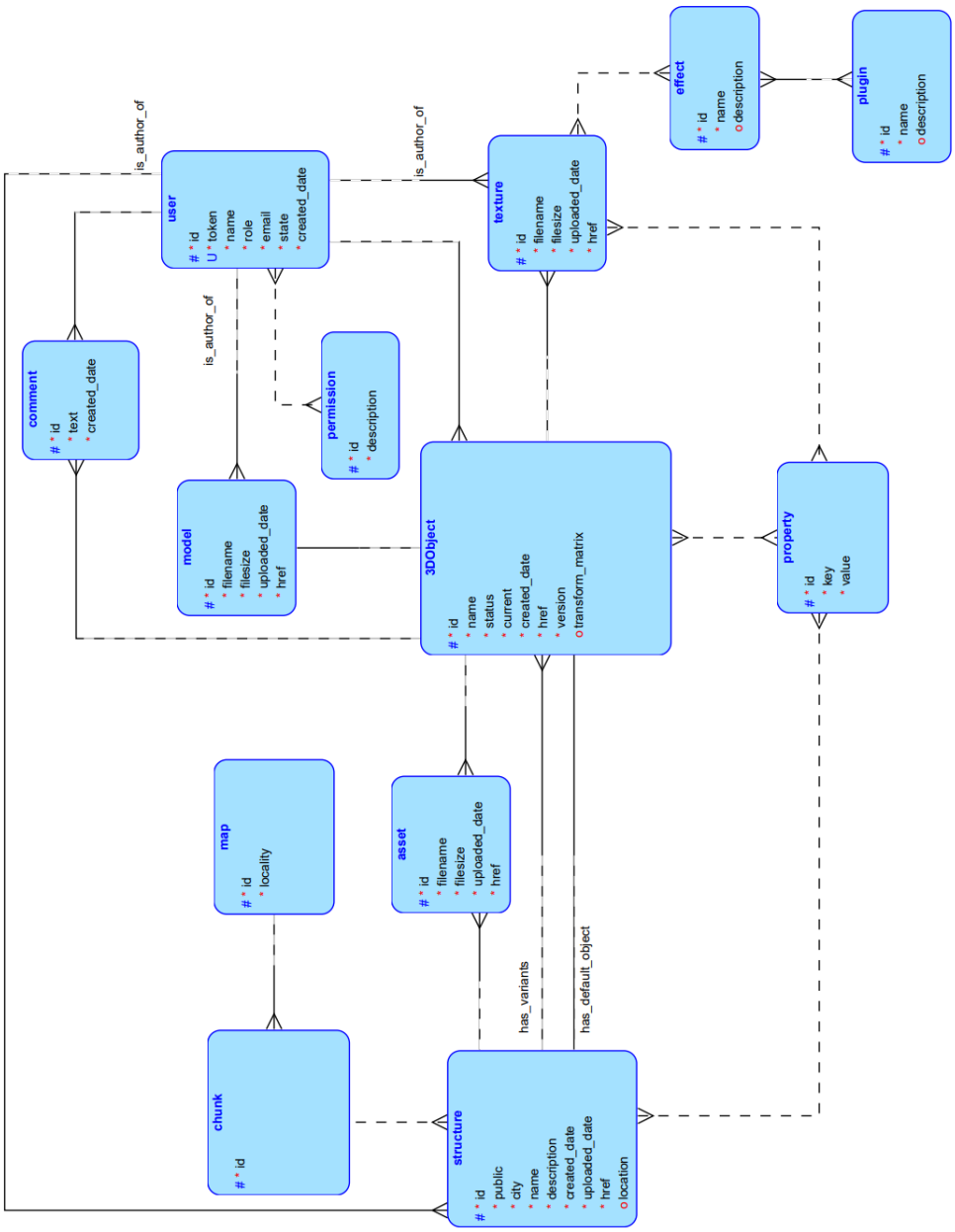
Vzhledem k počtu iterací během projektu VMČK došlo již k několika změnám a upravám stávající databáze, kde jsou uložena všechna potřebná data (od uživatelských dat až po vlastní 3D modely). Nejvetší změny udělal nejspíše kolega Daniel Vančura, který navrhl a realizoval potřebné databázové změny pro adaptaci schvalovacího procesu modelů (více na obrázku 2.1).

Jak bylo však později zjištěno, tento návrh nebyl dokonalý a bylo nalezeno hned několik nedostatků, mezi které se například řadí inkluze specifických dat schvalovacího procesu přímo v tabulce ***3DObject***, jako je třeba uživatelské jméno schvalovatele. Mezi další takové chyby návrhu patřila nedostatečná reprezentace předchozích verzí daného modelu pro snadnější zobrazení jeho historie úprav. Více je možné se dočíst v diplomové práci Dominika Siváka [4], kde mimo jiné navrhuje upravený databázový model, řešící tuto problematiku přidáním nových databázových tabulek jako je ***ApprovalProcess***, ***ApprovalIteration***, ***ApprovalComment***, ***ApprovalAction*** a ***ApprovalVariant*** (viz obrázek 2.2). Právě tyto nově přidané tabulky jsou pílířem pro návrh a implementaci schvalovacího procesu modelů, což je jeden z hlavních cílů mé práce. Pro zjednodušení vývoje jsme společně s kolegou Sivákem naplnili databázi vygenerovanými daty. Snáze se poté implementuje komunikace s API, protože server vrací v odpovědích reálně strukturovaná data. Očekávám, že před přechodem na ostrou verzi projektu budou tato data smazána a nahrazena již existujícími a postupně vznikajícími produkčními modely.

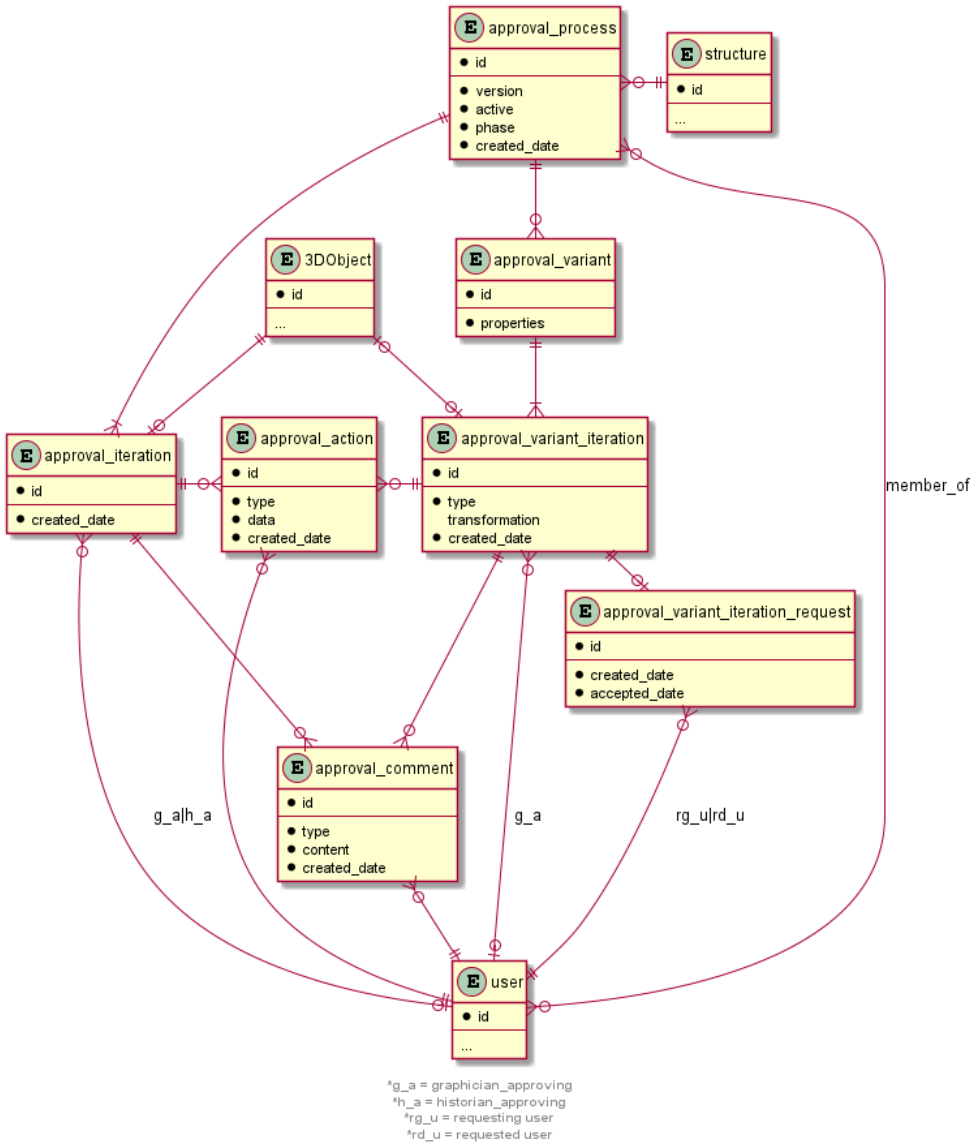
Data jsou reprezentována v relačním databázovém systému **PostreSQL** a díky nainstalovanému webovému rozhraní **Adminer**[5] se dá k datům přistoupit napřímo. Tato funkcionalita je velmi mocná a může být i dvousečná. Vývojářům, kteří s projektem VMČK teprve začínají a dostali k databázi přístup, nedoporučuji upravovat ručně žádné záznamy, především ty, které se týkají schvalovacího procesu. Data jsou sofistikovaně provázaná na více místech a externí zásah, bez použití příslušných API, může znamenat kritickou chybu.

[4] Adresa API serveru ke dni 16.6.2021: http://109.123.202.213:3000

[5] Adresa webového rozhraní databáze ke dni 16.6.2021: http://109.123.202.213:3001



Obrázek 2.1: Schéma databáze navrhnuté Danielem Vančurou [3].



Obrázek 2.2: Rozšíření původního databázového schématu o prvky schvalovacího procesu Dominikem Sivákem [4].

2.2 Schvalovací proces 3D modelů

Základní kámen schvalovacího procesu pro 3D modely položil kolega Martinek společně s Vančurou a Chludilem. Hlavním cílem této práce je adaptovat tento návrh a implementovat jej v aplikaci ve virtuální realitě. Než se pustím do samotné analýzy procesu, bylo by dobré si vyjasnit uživatelské skupiny, které jsou jeho nedílnou součástí a budou využívat tuto aplikaci.

Historik Zadává poptávku na tvorbu daného 3D modelu a vyplňuje k němu historicky přesné informace. Po zpracování tohoto požadavku modelářem, kontroluje vytvořený 3D model dle historické věrohodnosti ve virtuální realitě a v případě nepřesností vrací model na přepracování.

Modelář Využívá samotnou aplikaci pouze k vylistování aktuálně otevřených poptávek od historiků, které může akceptovat. Vlastní model poté vytváří v jiném programu, který je k tomu určen (například Blender, 3Ds Max...). Rozhraní aplikace mu umožňuje svou hotovou práci nahrát a odevzdat. Jsem si vědom toho, že pro Modeláře je aplikace ve virtuální realitě celkem zbytečná. Pro samotné přijimaní požadavků a nahrávání finální práce proto tato role spíše využije webové rozhraní, implementující také tento schvalovací proces, čímž se zabývá bakalářská práce Pavla Antoše [9].

Grafik Člověk znalý počítačové grafiky a s jistou zkušeností s modelovaním. Pomocí navrhnutých interakcí s 3D modely ve virtuální realitě (kapitola 3.6), jako je například zobrazení Wireframe[6] modelu, pečlivě zkoumá kvalitu zaslané práce od modeláře. Stejně jako historik muže poslat model zpět na přepracování společně s komentářem, co je nutno opravit.

Samotný životní cyklus 3D modelu je rozdělen do dvou fází, předtím než je připraven pro produkční použití nebo zpřístupněn veřejnosti. Produkčním použitím mám na mysli ostatní aplikace vyvíjené v rámci projektu VMČK, například zobrazení historických modelů na mobilním telefonu.

První fází tohoto procesu je tvorba modelu. Jak už jsem naznačoval výše, vše začíná u historika, který pomocí aplikace vytvoří požadavek na realizaci tvorby věrohodného modelu. Pro usnadnění modelování, doplní historik do zadání práce potřebná metadata, jako například kde se artefakt nacházel, z jakého je století a pokud možno nějaké vizuální podklady. Na pozadí, aplikace využívá privátní API a odesílá POST[7] požadavek na zdroj */structures*, čímž se vytváří v databázi nový záznam se zadáním. Následně modelář skrze aplikaci přijimá vybrané zadání a pouští se do tvorby. Pro usnadnění

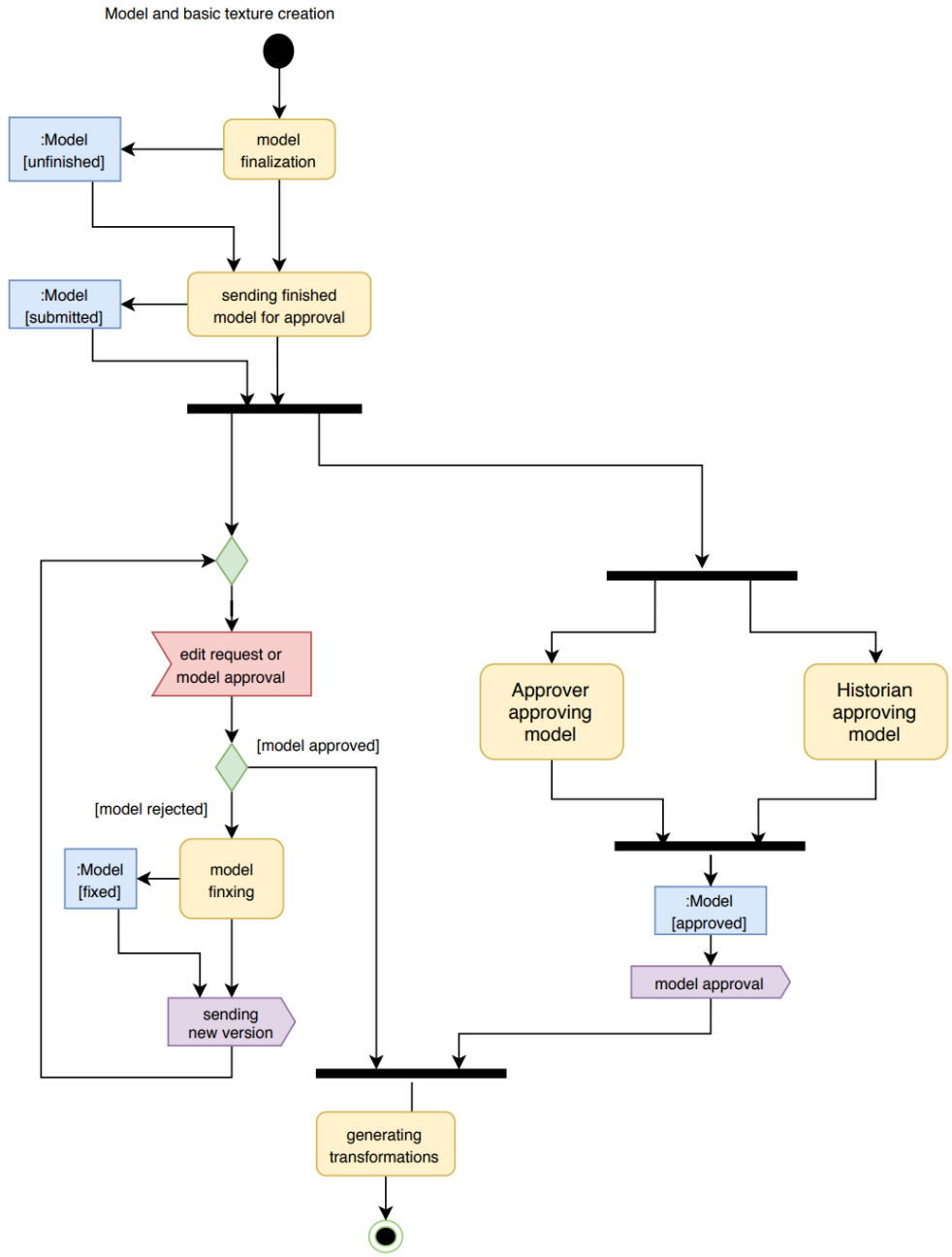
[6] Wireframe je zobrazení modelu pouze pomocí jeho hran a vrcholů. Je tak snadnější nahlédnout dovnitř modelu a zjistit, jak byl vytvořen [10].

[7] POST je jedna z metod posílání požadavků po síti. Je typická tím, že společně s hlavičkou posílá neprázdné tělo obsahující data a upravuje cílový zdroj [11].

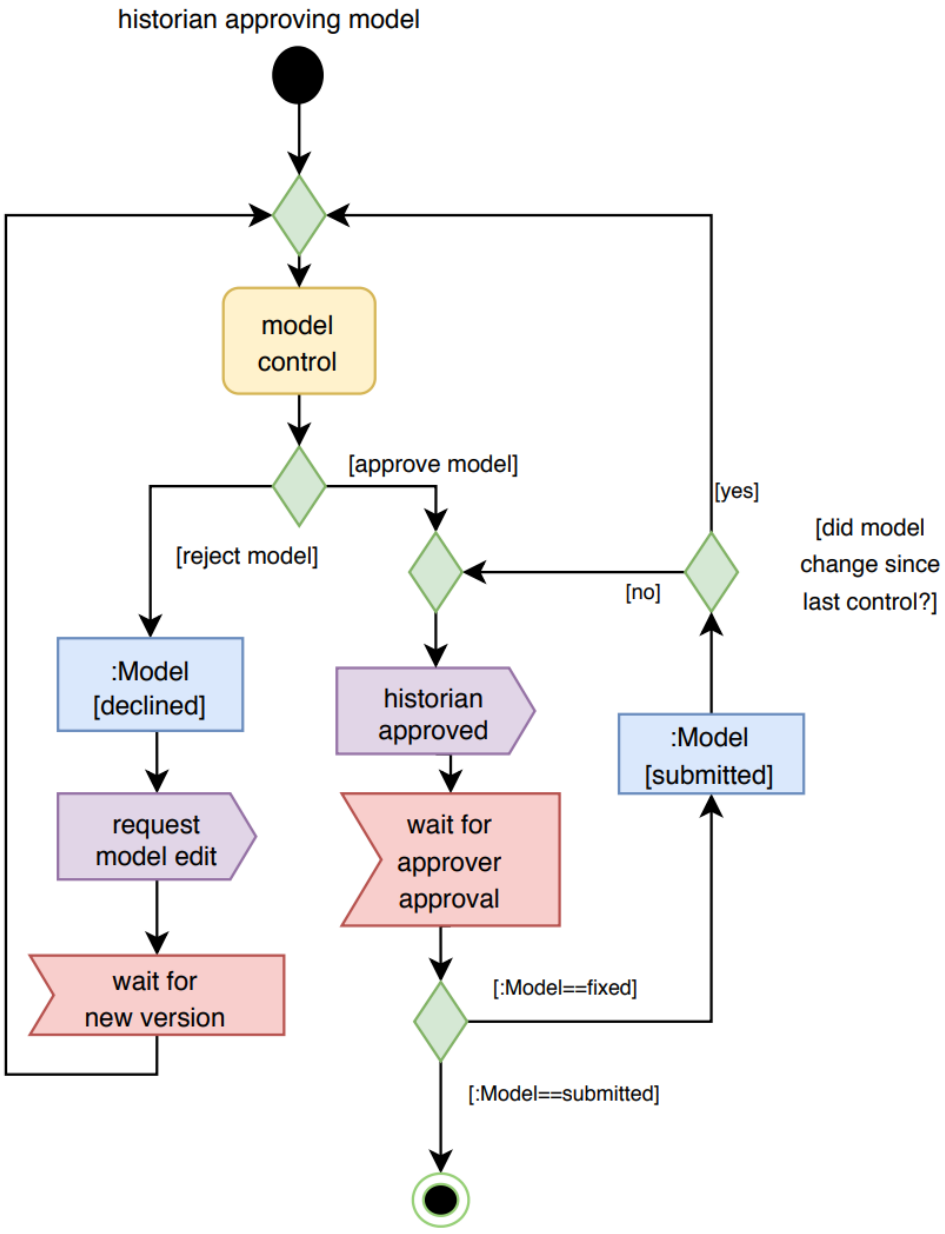
práce a také pro zachycení historie vývoje, modelář může postupně nahrávat jednotlivé verze modelu a v momentě kdy je spokojen, hotový čistý model odevzdává. Interně aplikace opět volá POST požadavek, tentokrát však na cílový zdroj */3Dobjects*. Po nahrání finální verze modelu jsou upozorněni a přiřazeni do procesu další dva uživatelé, přesněji historik a grafik. Ti mají za úkol ve virtuální realitě detailně prozkoumat vytvořený model a buďto ho schválit nebo vrátit s připomínkami na přepracování. V případě odmítnutí se práce vrací zpět modeláři, který pomocí zadaných poznámek model předělá. V případě, že jeden z uživatelů odevzdanou práci schválí a druhý ne, nově přepracovaný 3D model již zkoumá a schvaluje pouze ten uživatel, který k němu měl připomínky. Toto bylo navrženo čistě pro zjednodušení a zkrácení celého schvalovacího procesu. V této části procesu aplikace komunikuje s API přes endpoint */approvals*, kde se prvně vytvoří instance schvalovacího procesu pro rodičovskou strukturu 3D objektu a následně se vytvoří modelová iterace zahalující vyjádření obou stran a všechny komentáře. Celá první část procesu je zachycena UML (Unified Modeling Language) digramem na obrázcích 2.3, 2.4 a 2.5, které vytvořil kolega Vančura [3].

Druhá fáze procesu sestává z poloautomatického generování ruzných variant schváleného 3D modelu. Pod slovem varianta se skrývá pouze povrchová a vzhledová úprava modelu, nikoliv manipulace s vlastní geometrií. Pro lepší představu uvažte například historický model Mýtské brány v Hradci Králové z počátku 15. století, který mimochodem také vznikl v rámci projektu VMČK. Na obrázku 2.6 lze vidět nejprve model bez úprav, poté s přidáním napadaného sněhu a následně povodňovým blátem. Autory těchto automatizovaných úprav textur jsou Denisa Sůvová [12] a Michal Zajíc [13], jejichž výsledky bakalářských prací mají v budoucnu sloužit právě v této fázi procesu pro generaci různých variant. Historik už v této části nefiguruje, místo něj se do procesu zapojuje server, na kterém se spouští vyvinuté skripty, generující žádané obměny. Nejpodstatnější roli zde má Grafik, který může libovolně parametrizovat tvorbu dalších podob modelu. V případě kdy je stále nespokojen s výsledky, má možnost požádat modeláře o ruční vytvoření. Tuto část procesu zachytil kolega Martinek opět UML diagramem na obrázku 2.7.

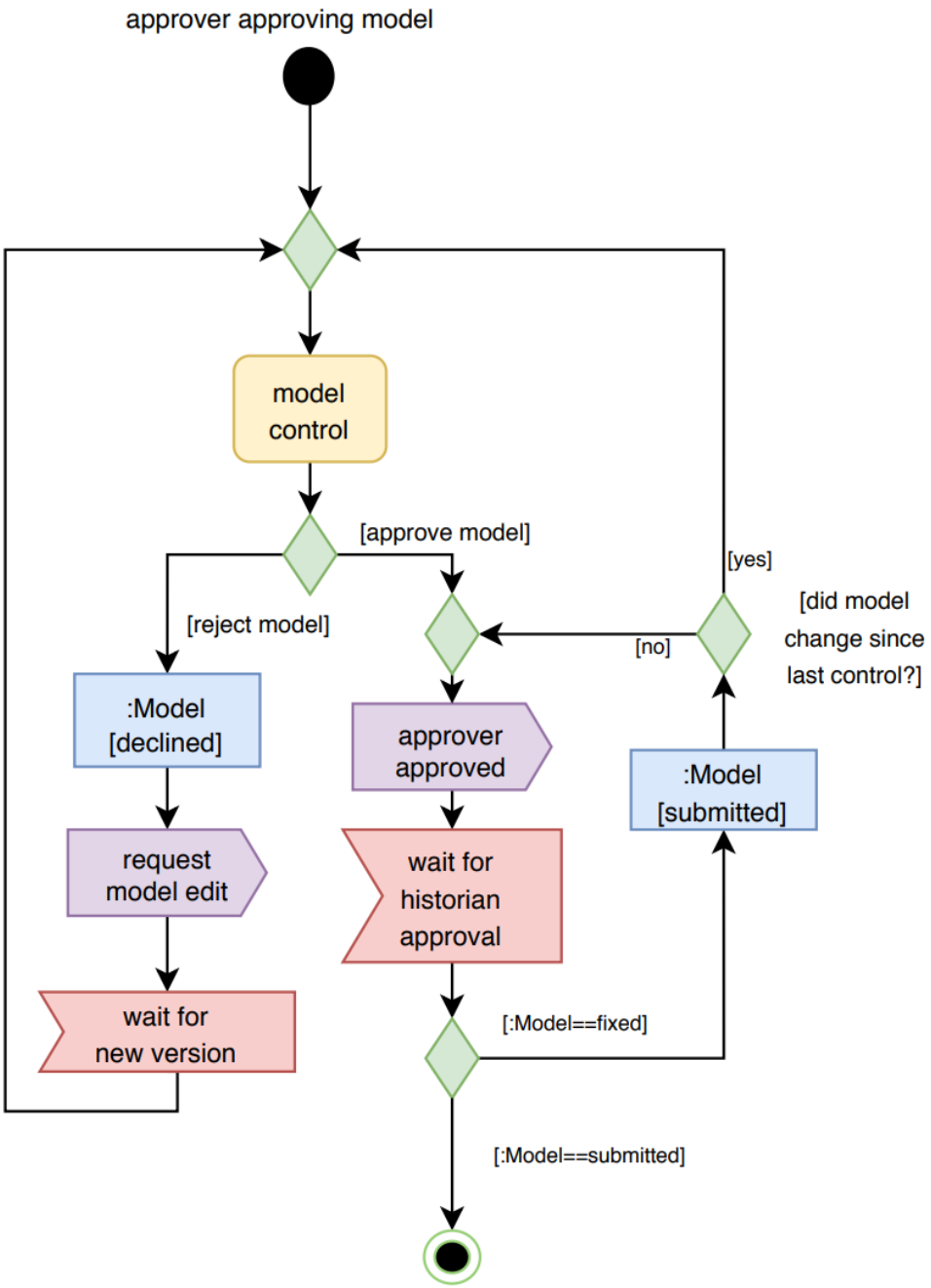
Pro shrnutí a lepší pochopení, čím se bude tato bakalářská práce zabývat, chci poznamenat, že nemám v plánu zahrnout druhou část schvalovacího procesu do návrhu ani do implementace. Výsledkem této práce by měla být aplikace ve virtuální realitě, umožňující grafikům či historikům zkoumat kvalitu odevzdaného modelu ve virtuálním světe, s možností vrátit objekt na přepracovaní s uvedenými poznámkami.



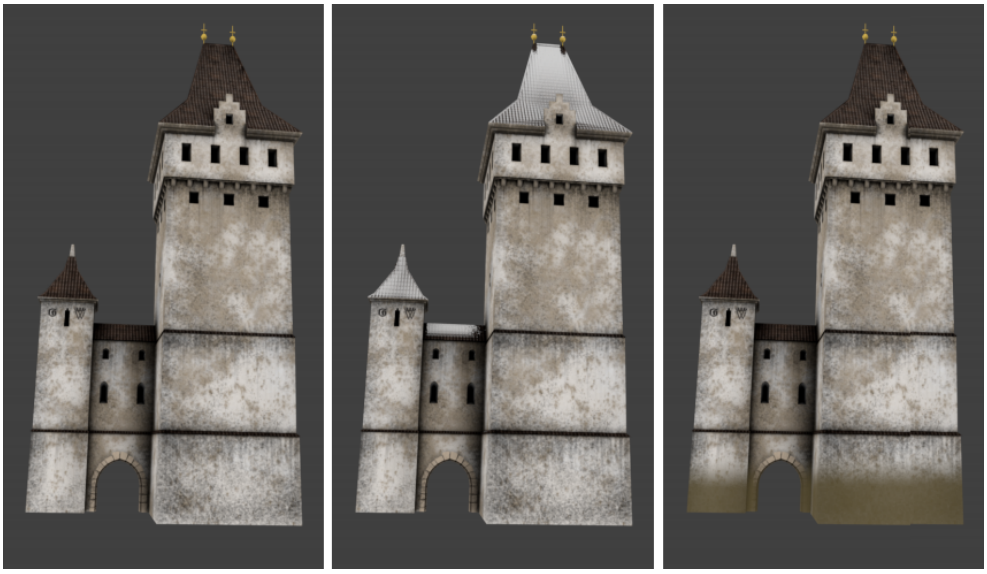
Obrázek 2.3: UML diagram zachycující tvorbu a schválení modelu. Navrženo v práci Daniela Vančury.



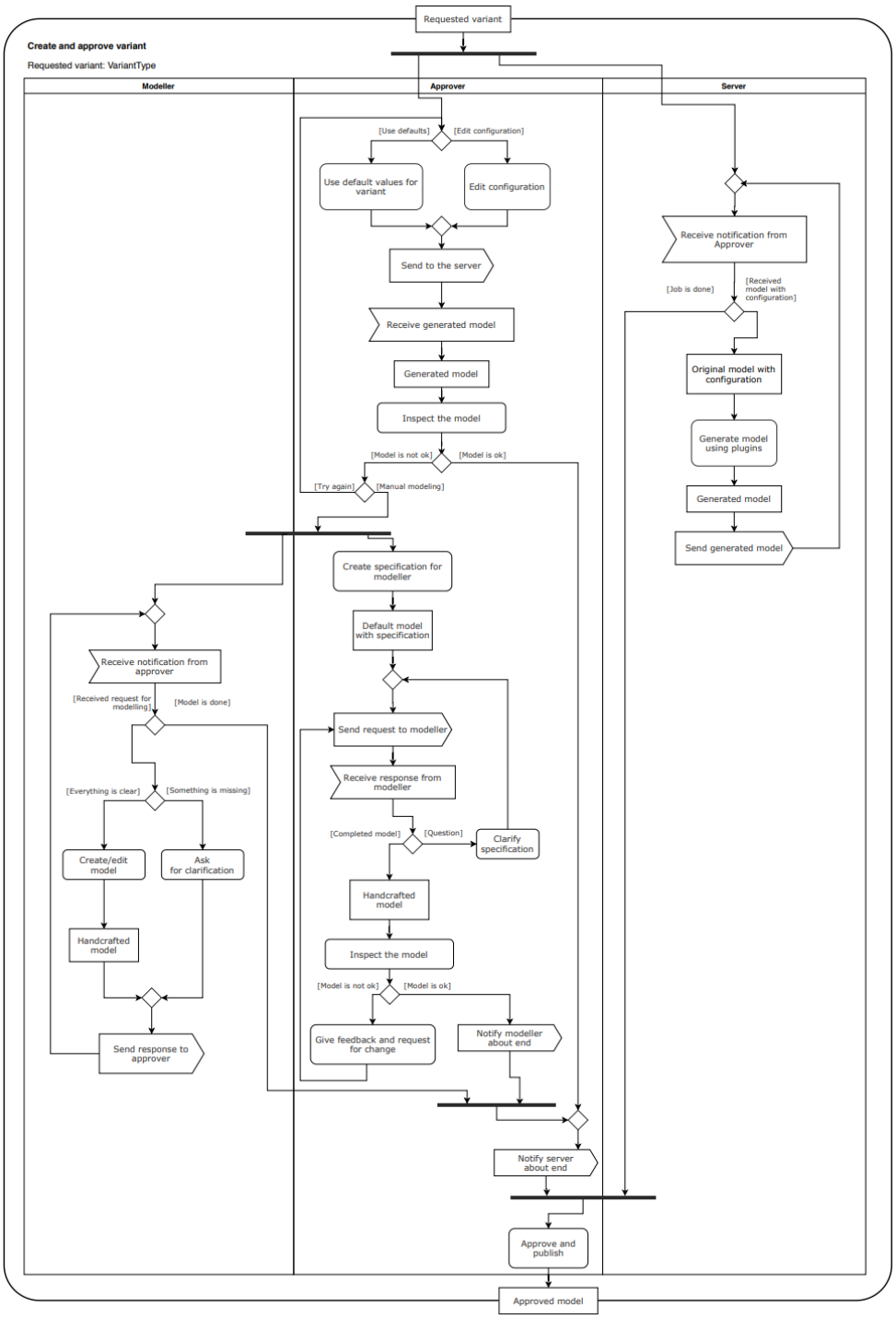
Obrázek 2.4: UML diagram zachycující schválení historikem. Navrženo v práci Daniela Vančury.



Obrázek 2.5: UML diagram zachycující schválení grafikem. Navrženo v práci Daniela Vančury.



Obrázek 2.6: Model Mýtské brány v Hradci Králové po aplikaci automatických úprav textur z práce Denisy Sůvové.



Obrázek 2.7: UML diagram zachycující druhou část schvalovacího procesu. Generace různých variant modelu z práce Michala Martinka.

2.3 Interní stav 3D objektů

Každý 3D objekt má svůj stav, který jasně udává, v jakém bodě schvalovacího procesu se nachází. Tato vlastnost propůjčuje vyvojáři klienstkých aplikací možnost dynamicky měnit akce, které uživatel může s objektem vykonávat. S obohacením privátního API kolega Sivák upravil některé z předchozích možných stavů objektů, proto bych zde rád shrnul aktuální řešení [4]. Životní cyklus 3D objektu v průběhu schvalovacího procesu zachytil kolega Sivák stavovým diagramem, který je uveden na obrázku 2.8.

UNFINISHED Defaultní stav, který objekt dostane po nahrání do systému. S tímto stavem modelu pracuje zatím pouze jen modelář a nahrává průběžně jeho nové verze.

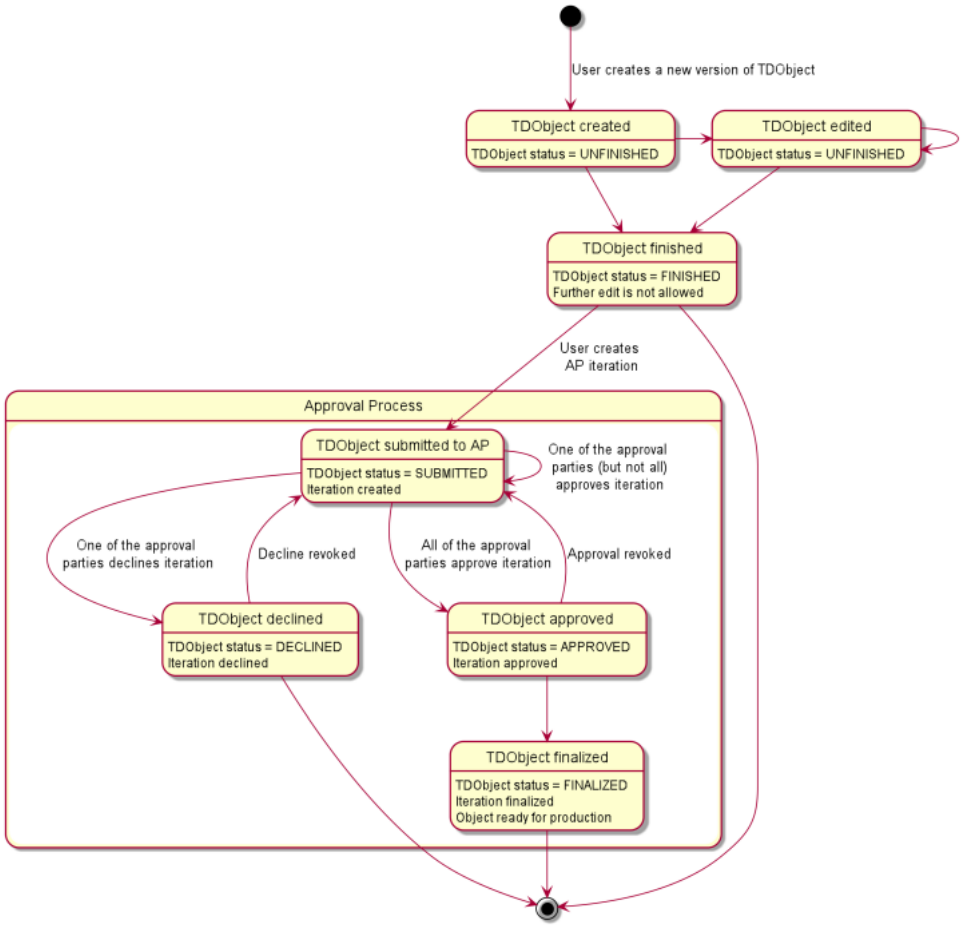
FINISHED Stav signalizuje, že modelář dokončil svou práci a je spokojen s odevzdaným výsledkem. Od této doby není možné objekt jakkoliv upravovat. Pokud modelář nechce, aby tento 3D objekt šel na schválení a nebyl součástí schvalovacího procesu, toto je pro něj finální stav. V klienstké aplikaci se dá na tento stav reagovat nabídnutím odevzdání ke schválení.

SUBMITTED Označuje objekt, který byl odeslán ke schválení a je součástí modelové iterace v aktivním schvalovacím procesu příslušné struktury. V tomto stavu zůstane do té doby, než obě role, jak historik tak grafik, neschválí iteraci. Aplikace tak na základě toho může odevzdaný model stáhnout pomocí API a vizualizovat ho pro snadné ohodnocení jeho kvality.

DECLINED Označuje stav, kdy se k objektu vyjádřili obě schvalovací autority a jedna z nich nebo obě ho zamítly. Pomocí API dokáže aplikace natáhnout korespondující komentáře a ukázat tak modeláři, proč jeho práce neprošla a co je nutné opravit.

ACCEPTED Odevzdaný objekt byl schválen všemi hodnotiteli a je připraven postoupit z modelové iterace do iterace variant (druhá část schvalovacího procesu, aplikující automaticky generované textury).

FINALIZED Finální stav ve kterém se objekt nachází na konci celého schvalovacího procesu. Jedná se o objekt připravený pro produkční použití v klientských aplikacích.



Obrázek 2.8: Stavový diagram z práce Dominika Siváka [4], zachycující možné stavy 3D objektu.

2.4 Cílová hardwarová platforma

Finálním výstupem této práce bude aplikace běžící ve virtuální realitě, je tedy zřejmé, že cílový uživatel bude muset vlastnit sadu vstupních a výstupních zařízení podporující VR pro její bezobtížné užívání. Na trhu najdeme mnoho různých možností vzhledem k tomu, že se virtuální realita stala poslední dobou velmi populární. Pro začínající programátory může být velmi obtížné se rozhodnout, pro kterou hardwarovou platformu mají vlastně vyvíjet.

Oculus VR Firma, která vytvořila jeden z nejznámějších a nejrozšířenějších HMD (Head mounted display) pro virtuální realitu. To co začalo jako projekt na *Kickstarter*, je dnes multimiliónový byznys zastřešený společností Facebook, která ho v roce 2014 koupila. *Oculus Rift* byl úplně první HMD, se

kterým firma prorazila na trhu a konkurovala ostatním. Pomocí technologie *inside-out tracking* je zařízení schopené uživateli poskytnout virtuální zážitek v celé místnosti, bez nutnosti instalace snímacích kamer. Díky svému ostrému duálnímu LCD (Liquid Crystal Display) displeji, poskytující 1080 x 1200 pixelů pro každé oko, je na trhu stále relevantní. Po akvizici Facebookem se firma rozhodla rozšířit svou cílovou skupinu snížením prodejní ceny a rozšířením dostupnosti novým modelem *Oculus Quest*. Toto zařízení je významné svým vestavěným operečním systémem (adaptace operačního systému Android), který umožňuje užívání HMD bez připojení k počítači. Později se začal prodávat *Oculus Link*, který jakožto doplněk umožňuje propojení s počítačem a hraní náročnějších her [14].

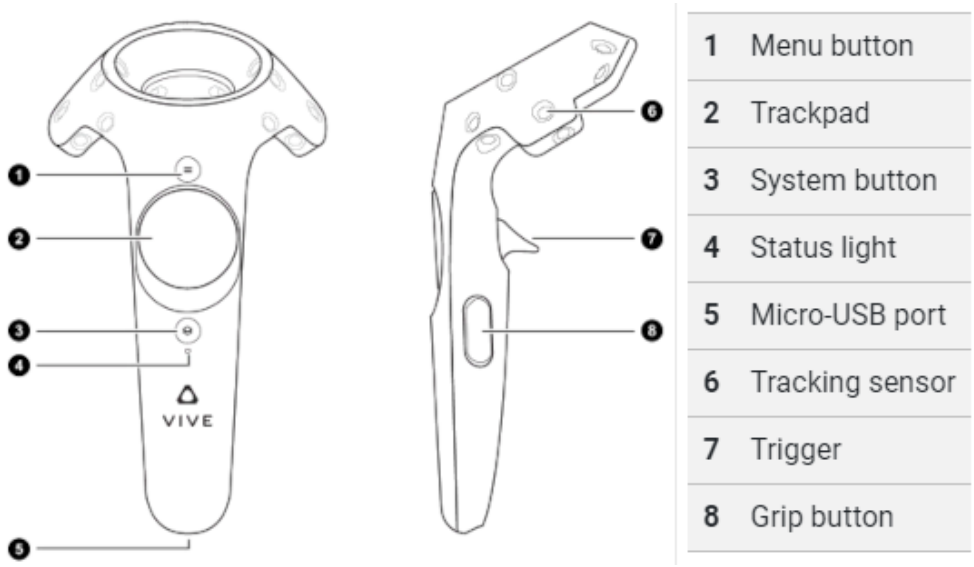
HTC & Valve Jeden z dalších kompetitorů na trhu s brýlemi pro virtuální realitu. Tyto dvě firmy se dohromady podíleli na vývoji *HTC Vive*, což je headset kterým se nejvíce proslavili. Podobně jako *Oculus Rift* nabizí displej s rozlišením 1080 x 1200 pixelů pro jedno oko, avšak na místo technologie LCD využívá OLED (Organic light-emitting diode). Součástí balení jsou dva ovladače, které slouží jako hlavní vstupní zařízení pro interakci s aplikací a zároveň snímají aktualní polohu rukou v prostoru. Na spodu ovladače se nachází tlačítko pro primární akci, tedy *Trigger button*, které se běžně používá například pro interakci s uživatelským rozhraním (potvrzení volby) nebo při střelbě ze zbraně v akčních hrách. Po stranách ovladače se nachází takzvaný *Grip button*, jehož nejčastější využití je sbíraní předmětů ve scéně a jejich následné odhození. Posledním typem vstupu který stojí za zmínku je *Trackpad*. Pomocí kruhového snímače dokáže přesně určit pozici palce a zároveň slouží jako pět ruzných tlačítek podle toho, jakou část tlačítka uživatel zmáčkne. Pro lepší vizualizaci rozmístění tlačítek na ovladači si zde dovolím přiložit schéma ovladačů *HTC Vive* 2.9.

Vedle ovladačů jsou součástí balení dvě kamery s úchyty, které je nutné umístit diagonálně naproti sobě tak, aby pokrývaly celou herní plochu. Minimální rozměry pro tuto plochu jsou dle oficiálních webových stránek výrobce 2 x 1.5 metrů, s tím že maximální vzdálenost mezi kamerami je 5 metrů. Naopak tedy od technologií Oculus VR, HTC Vive využívá takzvaný *outside-in tracking*, kde díky dedikovanému hardware pro snímání pohybu brýlí a ovladačů poskytuje lepší a plynulejší zážitek ve virtuálním prostoru. Zjednodušené zapojení kamer je možné vidět na obrázku 2.10. Nejvetší nevýhodou tohoto přístupu je fixní, předem nakonfigurovaná herní zóna, kde je headset aktivně sledován. Jakmile uživatel z této zóny vystoupí a kamery přestanou mít přímý pohled na nasazená zařízení, běžící program v headsetu začne vypadávat a v některých případech může způsobit nepříjemné závratě a pocity nevolnosti. Vzhledem k tomu, ze *HTC Vive* je plně imersivní headset, je velmi jednoduché vyjít z vymezené zóny a fyzicky zavadit o kolem stojící nábytek. Kvůli tomu byl vynalezen takzvaný *Chaperone* systém, který slouží jako světle

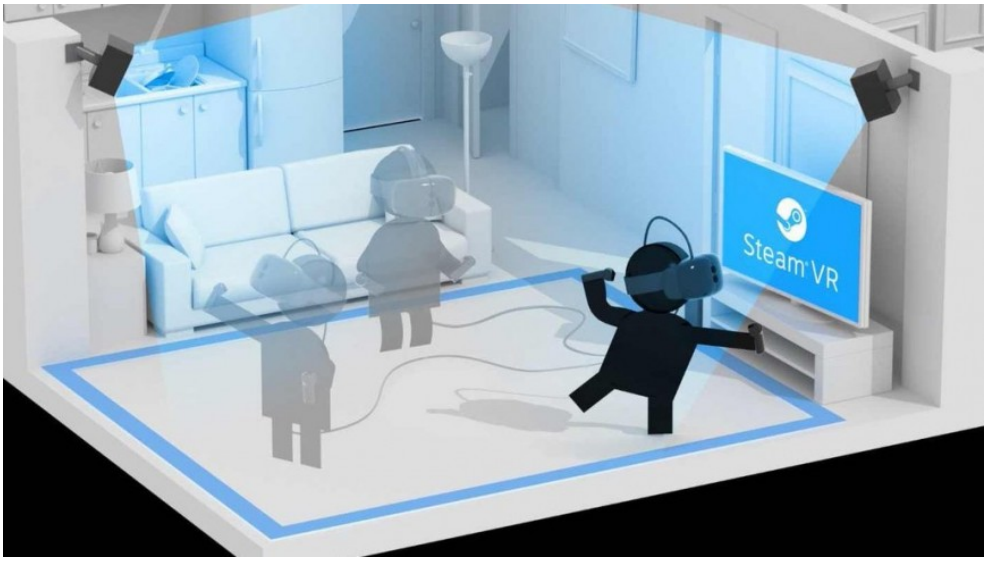
modrá síť promítající se do virtuálního světa, která naznačuje nastavené hranice herní plochy. Uživatel má tak stále přehled o tom, kde zhruba se v realném světě nachází (obrázek 2.11)

Ostatní zařízení Na trhu s headsety pro virtuální realitu existuje mnoho dalších výrobců, kteří se snaží zacílit na jinou skupinu zákazníků. Vzhledem k tomu, že tato práce se nemá zabývat rozborem aktuálního stavu trhu a analýzou všech dostupných zařízení, nebudu zde detailně vše popisovat, ale rád bych krátce zmínil zajimávé projekty. Mezi takové se řadí česká firma *VRgineers*, která se především zabývá vývojem špičkových headsetů pro profesionální použití. Své uplatnění nachází například v automobilovém průmyslu nebo v průmyslovém návrhu strojů. Jejich poslední headset *XTAL* vyhrál v roce 2020 cenu *Red Dot Award*, především díky svému novému designu, úchvatnému 8K rozlišení propůjčující ostrý pohled, rozšířenému zornému poli na 180 stupňů a automatickému zarovnání čoček v headsetu tak, aby kopírovaly vzdálenost očních čoček uživatele. Mimo jiné má headset vestavěný rozpoznávací systém pohybu rukou, není tedy nutné používat ovladače a je možné interagovat s virtuálním okolím velmi intuitivně. Tento headset spadá díky své kvalitě do úplně jiné cenové kategorie než například *HTC Vive* nebo *Oculus Rift*. Už jen díky svému návrhu je určen spíše pro korporátní nebo průmyslové učely, než pro jednotlivce [15].

V této kapitole jsem shrnul základní vlastnosti headsetu těch nejznámějších firem na trhu. Pro vlastní návrh a implementaci řešení problematiky, kterou se tato práce zabývá, je nutné si vybrat cílovou hardwarovou platformu, na které poběží finální aplikace. Rozhodl jsem se vyvíjet pro headset *HTC Vive* hned z několika důvodů, ale ten nejpodstatnější je fakt, že právě tyto brýle vlastním a je pro mě velmi pohodlné programovat a rychle zkoušet navrhnuté interakce s virtuálním prostředím. Ulehčuji si tak a zrychluji celý proces vývoje tím, že nemusím využívat žádné emulátory pro VR a zároveň jsem sám schopen posoudit jisté aspekty uživatelské přívětivosti jednotlivých navržených prvků (i když samotné uživatelské testovaní aplikace je také součástí této práce). Jak se dozvíme v následující podkapitole, samotný výběr hardware je sice důležitý, ale díky správnému výběru SDK (Software development toolkit) je možné zacílit více zařízení pomocí unifikovaného rozhraní.



Obrázek 2.9: *HTC Vive controller* s popisem jednotlivých tlačítek [16].



Obrázek 2.10: Nákres rozmístění dvou kamer od *HTC Vive* headsetu podporující *outside-in tracking* [17].

Obrázek 2.11: Nákres systému *Chaperone* od *HTC Vive*, poskytující uživateli přehled o konci herní plochy [18].

2.5 Vývojové prostředí Unity

Unity Engine 3D se řadí mezi nejrozšířenější vývojové prostředí pro tvorbu her a aplikací. Jeho hlavními přednosti je bezpochyby multiplatformní zaměřenost, komponentový systém, intuitivní uživatelské rozhraní a celkem nízká bariéra vstupu pro nové vývojáře. Toto vývojové prostředí jsem si předem vybral pro realizaci aplikace z důvodu předchozích pozitivních zkušeností a obrovské aktivní komunity nezávislých vývojářů, která se kolem Unity utvořila.

Vlastní vývoj aplikací probíhá v Unity editoru, který poskytuje přehled projektových souborů, pohled hlavní kamery, pohled do scény, hierarchii objektů ve scéně a v neposlední řadě inspektor, zobrazující detail označené entity. Engine je založen na komponentové architektuře, což zajišťuje modularitu a rozdělení funkcionalit do přepoužitelných celků. Základním kamenem jsou takzvané *GameObjects*, což jsou entity reprezentující objekty ve scéně. Na *GameObject* se dají přípínat předem definované komponenty a skripty, rozšiřující jeho funkcionalitu a upravující jeho chování. Jednoduchým příkladem komponenty, kterou má ve výchozím stavu každý objekt, je *Transform*. Obsahuje pozici objektu ve *World space*, jeho rotaci a škálu. Další typickou komponentou jsou právě skripty psané v jazyce C#. Vytvořené třídy dědí z rodičovské třídy *MonoBehaviour*, díky níž je skript připnutelný na *GameObject* a jeho veřejné proměnné jsou vidět přímo v editoru a dá se s nimi manipulovat.

Právě díky tomuto návrhu je velmi snadné ladění programu za běhu pomocí inspektoru v módu *Debug*, který zobrazuje mimo věřejné proměnné i privátní a dá se tak sledovat stav objektů ve scéně. Třída *MonoBehaviour*

obsahuje několik funkcí, které skriptovací backend Unity vždy sám od sebe spouští a na pozadí je orchestruje, což je zachyceno diagramem na obrázku 2.12. Mezi ně se řadí například níže vybrané metody:

Awake() Metoda slouží k inicializaci a nastavení výchozích hodnot skriptu před startem aplikace. Skriptovacím backendem je volána při načítání scény nebo tehdy, pokud rodičovský *GameObject* byl neaktivní, a jeho stav se změnil na aktivní. Je voláná i při tvorbě šablonovitých objektů, takzvaných *Prefabs*, za pomoci metody *Instantiate*. Pokud v aplikaci dochází k přepínaní různých scén nebo k načítání té stejné znovu, metoda **Awake()** se pokaždé zavolá [19]. Ukázkovým využití této funkce je inicializace návrhového vzoru **Singleton**, který reprezentuje v rámci běhu aplikace právě jednu instanci třídy. Metoda na začátku kontroluje, jestli už instance není vytvořena a označí ji jako statickou pomocí jiné Unity funkce *DontDestroyOnLoad*. Objekt se tím pádem nesmaže při změně scény a můžeme se na něj odkazovat pomocí veřejné proměnné *Instance*.

```
public class SingletonExample
{
    public static SingletonExample Instance {get {return instance;}}
    private static SingletonExample instance;

    private void Awake()
    {
        if(instance != null && instance != this)
        {
            Destroy(this.gameObject);
        }
        else
        {
            instance = this;
        }
        DontDestroyOnLoad(this.gameObject);
    }
}
```

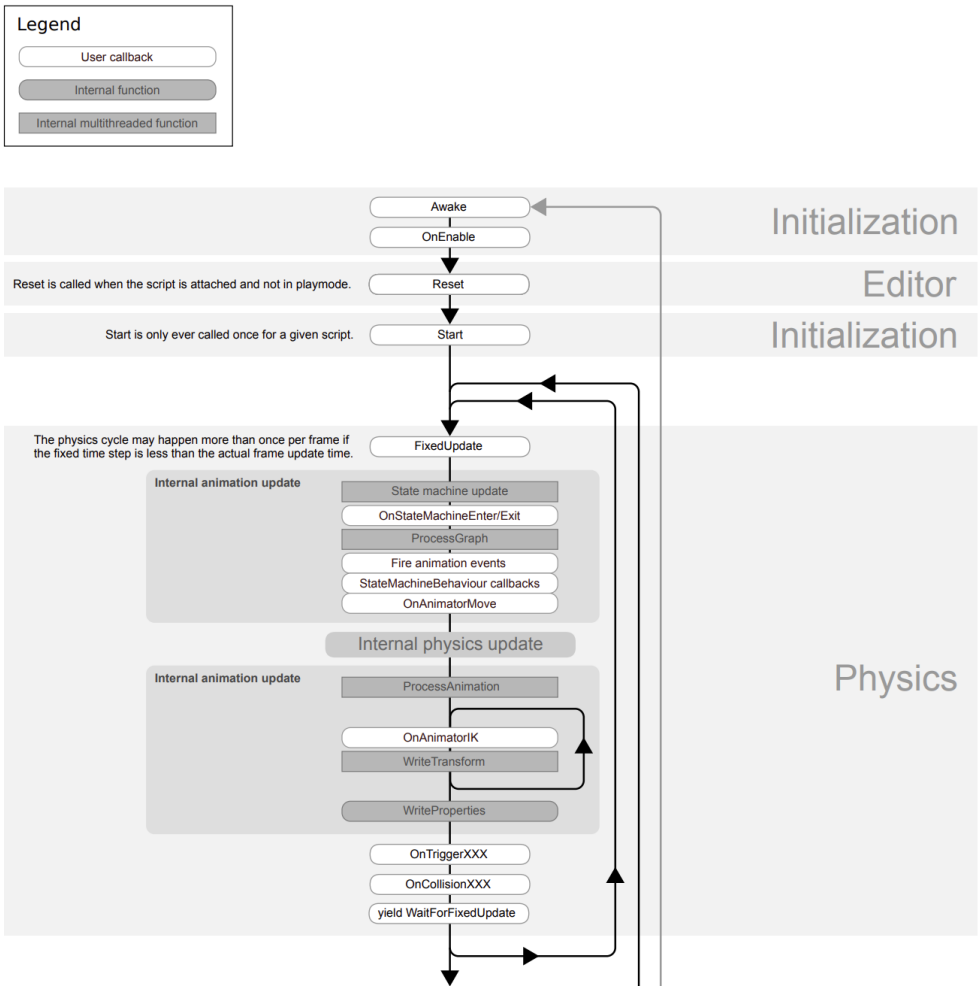
Start() Tato funkce je volána po skončení všech **Awake()** volání. Stejně jako ona se v životním cyklu skriptu volá pouze jednou, může se tedy zdát, že je zbytečná. Opak je však pravdou a toto rozlišení využije například třída A, potřebující pro svou inicializaci prvně inicializaci jiné třídy B. Unity zaručuje korektní volající sekvenci, aby se předešlo nedefinovanému chování [20].

Update() Spouštěn každý vyrenderovaný snímek. Hodí se pro definici periodického chování objektu ve scéně nebo k pravidelné kontrole nějakého stavu. Pokud bychom chtěli pracovat s pohybem předmětu v prostoru v kontextu fyzikálního modelu, je lepší použít **FixedUpdate**. Ten je nezávislý na snímkové frekvenci počítače, na kterém aplikace běží a pohyb tak působí plynuleji i na starších a méně výkonných hardware zařízeních [21].

StartCoroutine() Optimalizovaná cesta jak vykonat funkci bez čekání na její doběhnutí. Typicky korutina vykoná blok práce a poté odevzdá řízení zpět skriptovacímu backendu s časovým parametrem za jak dlouho se má znovu spustit v podobě sekund. Velmi často se korutiny využívají při asynchronním volání webového API nebo úkonů, které je třeba vykonávat opakovaně. Pro repetetivní úlohy jako je kontrola vzdáleností postavy hráče od nepřítele nebo pohyb postav pomocí umělé inteligence, jsou korutiny lepším řešením než ověřování během každého snímku v metodě **Update** [22].

Unity Engine 3D je mocný nástroj, s nímž lze vyvíjet 2D a 3D aplikace. Rozsáhlá komunita tak v průběhu let vytvořila řadu podpůrných balíčků, které si lze stáhnout a nainstalovat přímo skrze Unity. Rozšiřují základní funkcionality a obohacují už tak komplexní a rozvinutý systém. Jedním z těchto balíčku je **XR Interaction Toolkit**, který jsem si vybral pro implementaci VR aplikace. Zásadním cílem tohoto modulu je abstrahovat komunikaci s různými VR hardware zařízeními natolik, aby programátor mohl využívat sjednocenou sadu funkcí k definování specializovaných interakcí. Vyvinuté aplikaci je poté jedno, jaký headset a ovladače jsou připojeny a funguje tak napříč platformami. Zároveň balíček nechává určitou volnost vyvojáři v podobě definování vlastních eventů a interakcí s předměty ve scéně. Jeho hlavními komponenty jsou *XRBaseInteractor*, *XRBaseInteractable* a *XRInteractionManager*, který je spojuje. *XRBaseInteractor* se připíná na oba dva ovladače a umožňuje tak interakci s předměty ve scéně, pokud na sobě mají *XRBaseInteractable*. Skripty mají řadu eventů, vyvolané aktivní interakcí, na které se jde navěsit a vykonat tak nějakou akci, například po vyvolání *OnSelectedEnter* eventu můžeme změnit barvu objektu, kterého jsme se dotkli.

Existuje samozřejmě mnoho dalších Unity balíčku a SDKs (Software development toolkit), rozšiřující Unity o podporu VR. Klára Matoušková je ve své bakalářské práci [23] detailně rozebrala a porovnala jejich využití. Já jsem si **XR Interaction toolkit** zvolil pro návrh a vlastní implementaci VR aplikace hned z několika důvodů. Jedním z nich je už zmiňovaná podpora širokého spektra VR headsetů na trhu a transparentní přístup ke zdrojovým kódům komponentových skriptů XR tříd. Mnou později navrhnuté a realizované interakce s historickými předměty rozhodně nejsou triviální a a tento framework mi poskytne dostatek volnosti pro jejich realizaci.



Obrázek 2.12: Část životního cyklu skriptu v rámci Unity. Celý diagram je možné nalézt zde [24].

Kapitola 3

Návrh

3.1 Autentizace a autorizace

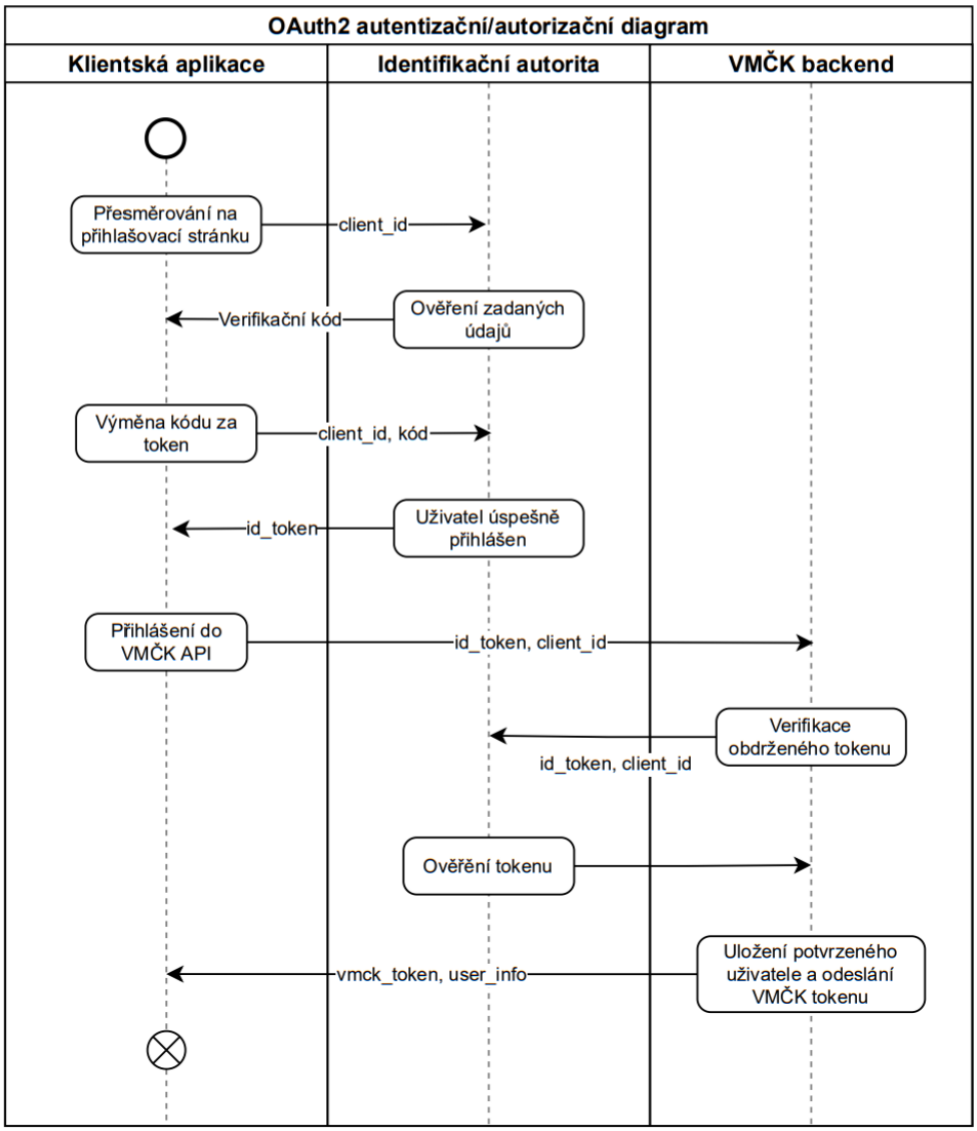
Aplikace bude komunikovat s privátním API a zároveň umožňovat uživateli zasahovat do schvalovacího procesu jednotlivých 3D modelů, je tedy nutné prvně uživatele nějak identifikovat a následně mu přiřadit patřičná oprávnění. Z hlediska zmiňovaného procesu je důležité zachytit, kdo schválil daný model, kdo ho vrátil na přepracování nebo kdo je autorem připojených komentářů. V předchozích verzích API se k backendu přihlašovalo pomocí vygenerovaného tokenu, který se distribuoval mezi koncové uživatele. Toto bylo velmi nepraktické a ještě ke všemu nepoužitelné z bezpečnostního hlediska.

S Dominikem Sivákem jsme proto navrli a později implementovali přihlašení pomocí Google účtů přes protokol OAuth2. Dominik se věnoval serverové části, tedy přijmutí a verifikaci Google tokenu od klientské aplikace a následné odeslání VMČK tokenu zpět, pomocí kterého se mohou provolávat další API endpoints. K tomuto účelu navrhl endpoint */auth/googlesigin* s parametry *idToken* a *Client ID*. Pro úspěšné přihlášení musí tedy moje klientská aplikace sestavit a odeslat korektní *POST* požadavek s těmito daty.

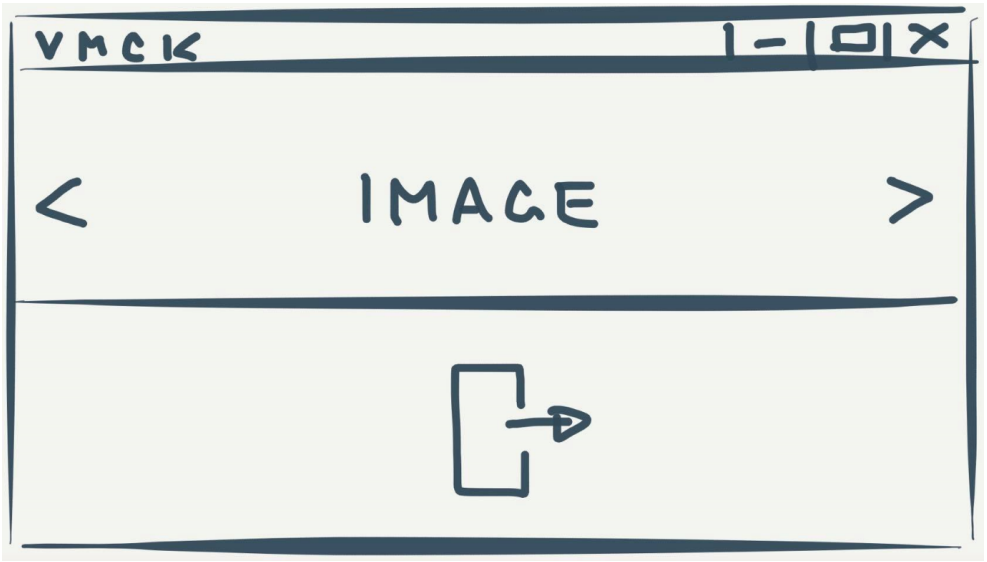
OAuth2 přihlašovací proces Pro navazující práce v projektu VMČK zde stručně popíši navhrnutý přihlašovací proces. Jedná se o dodržení specifikovaného autentizačního a autorizačního standardu, který umožňuje bezpežné a jednotné přihlášení napříč více webovými službami. V našem případě nám pomáhá s identifikací uživatele a zároveň není třeba v backendové části řešit uchovávání uživatelských hesel a správu účtů. Jedním z dalších přínosů je, že si uživatel nemusí zakládat žádné další účty a může využít svůj existující Google nebo školní účet. Vlastní proces začíná na straně klienta, kde je nutné uživatele přesměrovat na webovou stránku dané autority, v tomto případě Google, kde může zadat své údaje a provést přihlášení. V URL adrese této stránky je nutné uvést *Client ID* aplikace, která vyžaduje přihlášení. Pokud jsou zadané údaje korektní, vrátí se od identifikační autority odpověď ob-

sahující ověřovací kód. Tento kód je nutné následujícím volaním vyměnit za *idToken*, který už je privatní a obsahuje choulostivá data. Potvrzuje úspešné přihlášení skrze danou autoritu. V tuto chvíli už máme vše potřebné k provedení výše zmiňovaného */auth/googlesigin* s přiloženými *idToken* a *Client ID*. VMČK backend ověří zaslaný token a pokusí se vytvořit uživatele v tabulce *Users*. Pokud už záznam existuje pouze se o něm vrací informace včetně takzvaného *VMČK tokenu*, pomocí kterého se aplikace identifikuje API serveru při zasílání dalších požadavků. Celkové shrnutí tohoto procesu zachycuji na diagramu 3.1.

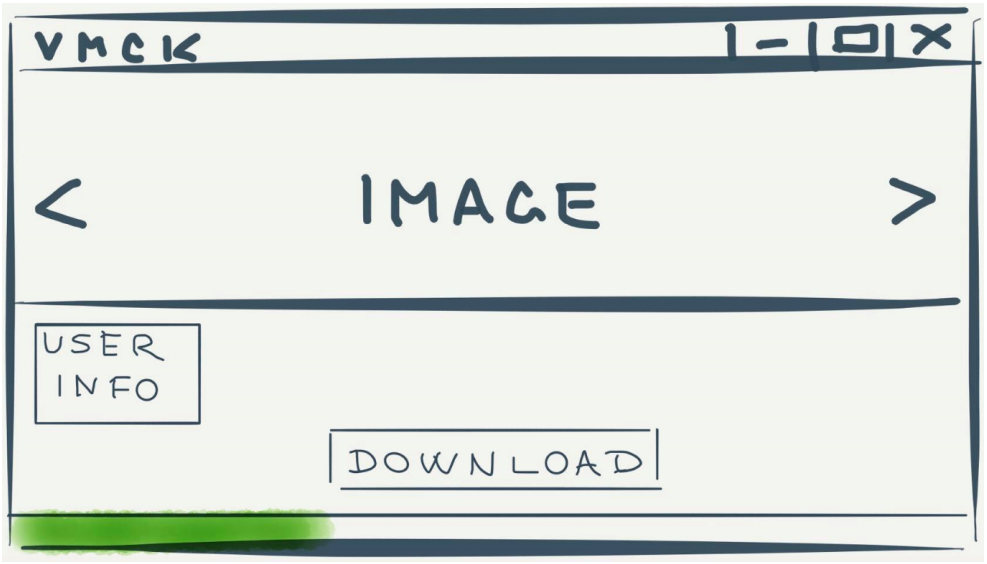
Launcher Pro vlastní přihlášení jsem se rozhodl pro finální aplikaci ve virtuální realitě navrhnout jednoduchý *Launcher*, skrze který se uživatel přihlásí. Mimo jiné, tento *Launcher* bude schopen stáhnout nejnovější verzi aplikace ze vzdáleného internetového repozitáře. K návrhu a pozdější implementaci tohoto *Launcheru* mě vedla skutečnost, že pro využití *OAuth2 přihlašovacího procesu* je nutné otevřít okno prohlížeče. Po důkladném zkoumání jsem zjistil, že ve virtuální realitě je velmi obtížné webový prohlížeč jakýmsi způsobem simulovat a otevřít. Nenašel jsem žádné řešení v podobě balíčku do Unity nebo knihoven třetích stran, které by jednoduše řešilo tento problém. Proto jsem se rozhodl navrhnout samostatnou desktopovou aplikaci pro operační systém Windows, která slouží jako vstupní bod, kde se uživatel přihlásí a následně se mu automaticky stáhne a nainstaluje VR aplikace. Tento *Laucnher* tedy bude řešit všechnu počáteční komunikaci jednak s identifikační autoritou (Google) tak i s privátním API VMČK a realizovat tak výše navržený způsob přihlašování. Zároveň se při každém spuštění vždy zkontroluje stav lokální verze a případně se stáhnou aktualizace. Tento *Launcher* budu vyvíjet v programovacím jazyce *C#* jako jednoduchou WPF (Windows Presentation Foundation) aplikaci. Na obrázcích 3.2 a 3.3 je možné vidět návrh uživatelského rozhraní, kterého se budu držet při implementaci.



Obrázek 3.1: Flowchart diagram zachycující OAuth2 autentizační/autorizační proces.



Obrázek 3.2: Navrh uživatelského rozhraní v *Launcheru*, předtím než se uživatel přihlásí.



Obrázek 3.3: Navrh uživatelského rozhraní v *Launcheru* poté, co se uživatel přihlásil.

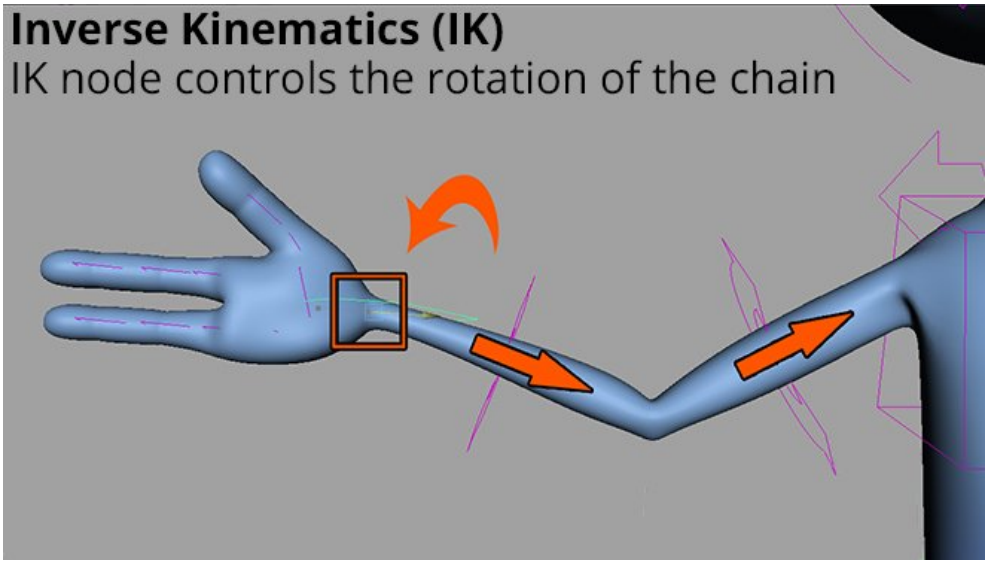
3.2 Pohyb hráče a avatar

Pro uživatele je nejdůležitější, aby se ve virtuální realitě cítil komfortně a mohl se intuitivně pohybovat. Pro člověka je samozřejmě nejintuitivnější fyzický pohyb, zde jsme však omezeni hardwarem (délka kabelu headsetu, plocha snímaná kamerami) a místností, kde se nachází. S touto problematikou už v minulosti bojovalo mnoho programátorů a bylo navrženo hned několik řešení, mezi které patří například teleportace pomocí laseru nebo posun ve virtuálním prostoru po stisknutí tlačítka. Z vlastní zkušenosti dokáži říci, že každému uživateli vyhovuje jiný typ pohybu. Kvůli tomu, že uživatel drží v rukách dva ovladače, jsem implementoval dvě výše uvedené varianty pohybu. Uživatel se tedy může dotýkat palcem levé ruky kruhového *Touchpad* tlačítka na ovladači a adekvátně se pohybovat v prostoru. Ti kteří preferují teleport jako metodu pohybu, mohou stlačit palcem stejné *Touchpad* tlačítko, tentokrát však na ovladači v pravé ruce. Po stisknutí se zobrazí mírně prohnutý laser, který vychází z uživatelovy pravé ruky a na svém konci má jednoduchý kruh. Tento kruh signalizuje cílovou pozici v prostoru, kam se uživatel přesune po uvolnění tlačítka. Nedává samozřejmě smysl, aby bylo možné se teleportovat například na zeď nebo na strop. Je nutné vytvořit takzvanou *Teleportation Area* (česky Teleportační plocha), která jasně vymezí validní cíle pro přesun. Více implementačních detailů je možné najít v kapitole 4.3.

S vlatním pohybem v prostoru je úzce spjata motorika těla a končetin. Pro věrohodný uživatelský zážitek z pohybu proto navrhuji jednoduchý Avatar, který má za úkol reprezentovat tělo uživatele ve VR. Model lidské postavy může být jakýkoliv, důležité však je, aby byl 3D model takzvaně *Rigged*. Ve zkratce jde o tvorbu kostry pro model, která umožňuje pohyb jednotlivých částí. Hierarchie modelu se určuje od kořenového uzlu přes ramena, ruce, nohy až po dílčí prsty. Dalším krokem je namapovat vybrané kosti modelu k softwarové reprezentaci headsetu a ovladačů, čili VR hardwaru. Během běhu aplikace tak specifické části modelu lidské postavy mění svou polohu v závisloti na poloze headsetu, respektive ovladačů.

Na první pohled se může zdát, že reprezentace těla pomocí VR Avatara je tímto vyřešena. Je důležité si však uvědomit, že se hýbe pouze s některými body v modelu jako je například zápěstí a krk. Ostatní části jsou touto translací neovlivněny a zůstávají na místě. Tato problematika se dá však elegantně vyřešit za použití *Inverzní Kinematiky*, která se mimo jiné především využívá při animaci 3D modelů. Tento nástroj umožňuje automatické dopočítání translace rodičů v modelové hierarchii na základě translace jejich potomka. Uvedeno na příkladu, posun zápěstí ovlivní polohu loketního a ramenního kloubu (obrázek 3.4). Samotná implementace je zjednodušena tím, že použité vývojové prostředí *Unity Engine 3D* inverzní kinematiku nativně podporuje a nabízí řadu předdefinovaných skriptů, kterým stačí vhodně nastavit parametry.

I přesto, že v aplikaci můžeme sledovat pouze vrchní část těla pomocí he-



Obrázek 3.4: Vizualizace *Inverzní Kinematiky* na příkladu pohybu zápěstí [25].

adsetu a dvou ovladačů, pohyb nohou avatara je klíčový pro více imerzivní zážitek. Dolní část těla je rozpohybovaná procedurálně generovanými animacemi, které jsou parametrizované natočením hlavy a rychlostí pohybu v daném směru. Při chůzi ve virtuálním prostoru se tak uživatel cítí pohodlně a vidí pod sebou své imaginární tělo s pohybujícíma se nohama.

3.3 Uživatelské rozhraní

3.3.1 Typy UI v Unity Engine 3D

Při návrhu uživatelského rozhraní ve virtuální realitě je nutné vzít v potaz hned několik možností jak UI prvky zobrazit. Z klasických počítačových her určitě všichni dobře známe takzvaný mód *Screen Space - Overlay*, který tyto prvky zobrazuje přes samotnou herní scénu a nabízí tak jednoduchou interakci s myší bez zásahu do scény. Tato varianta je bohužel pro VR nepoužitelná, právě kvůli zmíněnému překryvu a skutečnosti, že VR ovladače jsou přímo ve scéně, nemožné ovládat rozhraní.

Jednou z dalších méně ideálních možností je *Screen Space - Camera* zobrazení. Menu s UI elementy se tak zobrazuje přímo před danou kamerou s definovatelným odsazením a vždy následují tuto kameru v pohybu po scéně. V případě virtuální reality je kamerou vlastní headset a menu tak kopíruje sebemenší pohyb uživatelovi hlavy, včetně rotace. Díky tomu může tento režim způsobit nepříjemné pocity a vyvolat nevolnost.

Pro návrh a následnou realizaci uživatelského rozhraní ve VR je tedy nejlepší použít poslední variantu, což je *World Space* promítání. Umisťuje totiž

dané okno s ovládacími prvky přímo do prostoru herní scény, jako jeden z dalších objektů. Uživatel se kolem něj může pohybovat a interagovat s ním pomocí laserových paprsků, které vycházejí z ovladačů. Toto přirozené zakomponování UI do virtuálního světa se nazýva *Diegetické uživatelské rozhraní*.

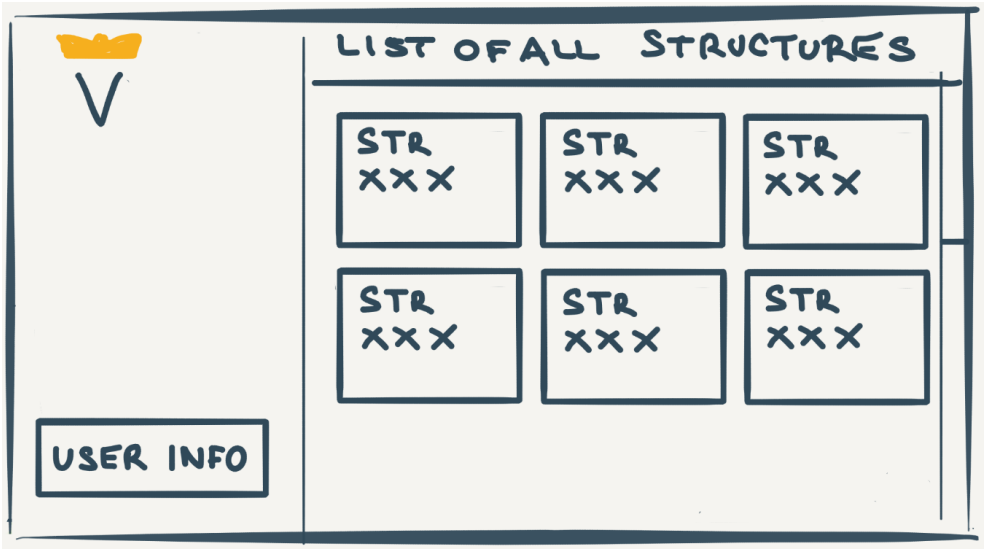
V první řadě je nutné vzít v potaz, jakou funkcionalitu očekávat od navrhovaného UI a čeho by měl skrze něj uživatel docílit. V mém případě je uživatelem historik nebo grafik, posuzující kvalitu 3D modelů. Pro něj je nejdůležitější výběr daného modelu a jeho následné zobrazení do scény. Po bližším zkoumání k němu pomocí rozhraní přidává komentář a schvaluje ho, či zamítá. To mě vede k návrhu pěti různých panelů, které dohromady utvoří diegetické uživatelské rozhraní.

3.3.2 Seznam všech struktur

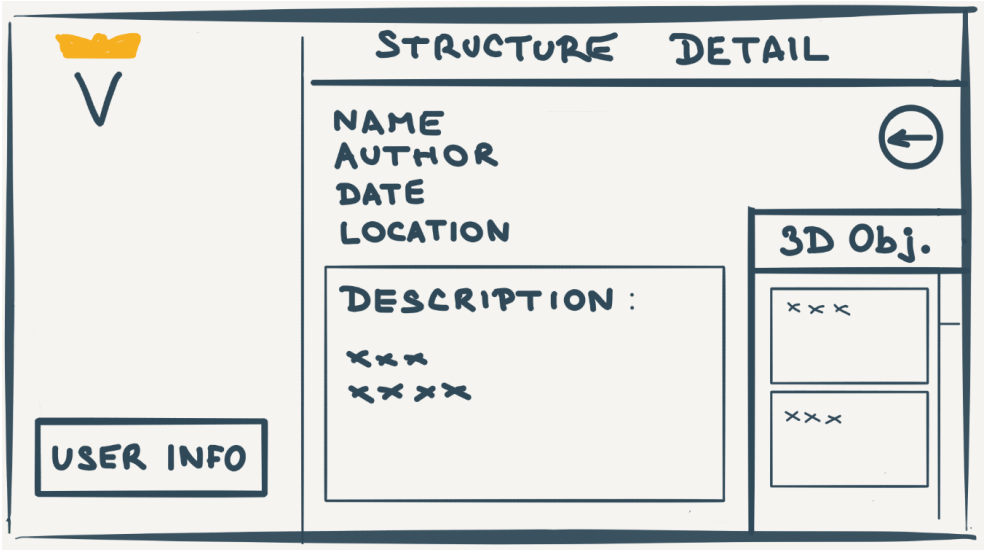
Tento panel slouží jako výchozí stránka. Obsahuje všechny aktuální struktury, nacházející se v backend databázi. Jak už bylo zmíněno v kapitole 2.2, struktura je hlavní entita obalující vše ostatní. Zakládá ji historik a zadává tak práci modeláři, který na základě poskytnutých dat vytváří model. Vzhledem k tomu, že před spuštěním aplikace počet struktur není známý a může být variabilní, panel se musí přizpůsobit pomocí vertikálního posuvníku. Náhledy struktur jsou totiž přidávány dynamicky v závislosti na jejich počtu. Na jedné straně se vždy zobrazí maximálně šest struktur a pro zobrazení ostatních musí uživatel posunout panel dolů. Posuvník je zároveň kompatibilní s laserem vycházejím z VR ovladače. Jednoduché náhledy jednotlivých struktur obsahují jejich název a autora, který je zadal. Náhledy slouží jako tlačítka, zobrazující detail dané struktury. Navrhnuté rozložení je možné vidět na obrázku 3.5.

3.3.3 Detail struktury

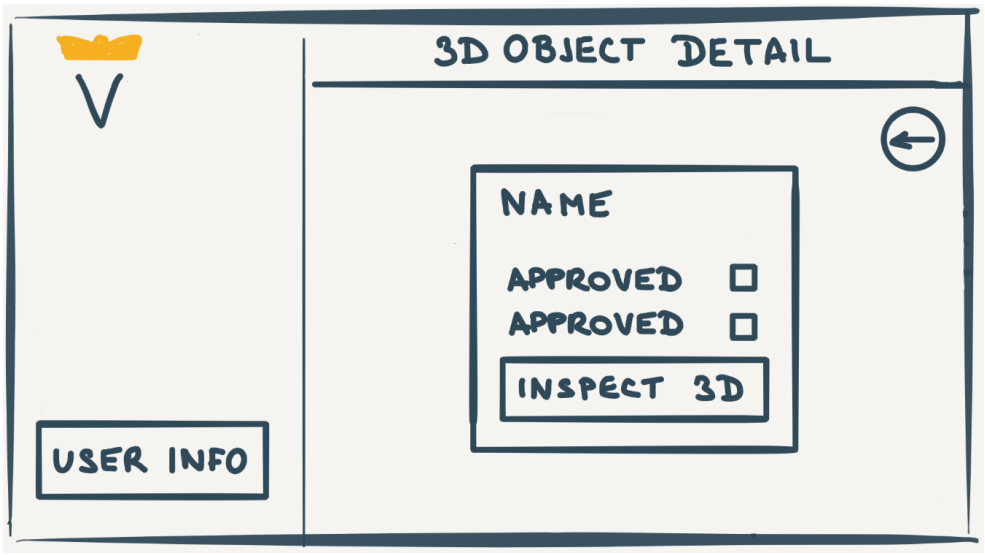
Na tomto panelu jsou vizualizovány informace o dané struktuře uložené v databázi. V levém horním rohu se nachází informace o tom, kdo strukturu založil, včetně datumu a lokace, kde se nachází. Pod tím nalezne uživatel box s popisem přibližující její původ nebo uvádějící jiné historické fakty. V pravé části se poté nachází menší panel, obsahující všechny varianty 3D objektů, které byly k této struktuře nahrány. Jedná se o odevzdanou práci modelářů, podléhající schvalovacímu procesu. Tyto náhledové karty 3D objektů fungují úplně stejně, jako náhledy struktur z předchozího panelu. Předem opět není známe kolik jich je a UI je opět řešené vertikálním posuvníkem. Jejich náhledy zobrazují název, status a stav schválení od historika a grafika. V neposlední řadě se v pravém horním rohu nachází kulaté tlačítko, které po zmáčknutí vrácí zpět na předchozí panel. Toto tlačítko se opakuje i u dvou následujících panelů a vždy vrací v hierarchii o jeden panel zpět. Návrh na obrázku 3.6.



Obrázek 3.5: Návrh panelu diegetického uživatelského rozhraní zobrazující všechny struktury.



Obrázek 3.6: Návrh panelu diegetického uživatelského rozhraní zobrazující detail struktury.



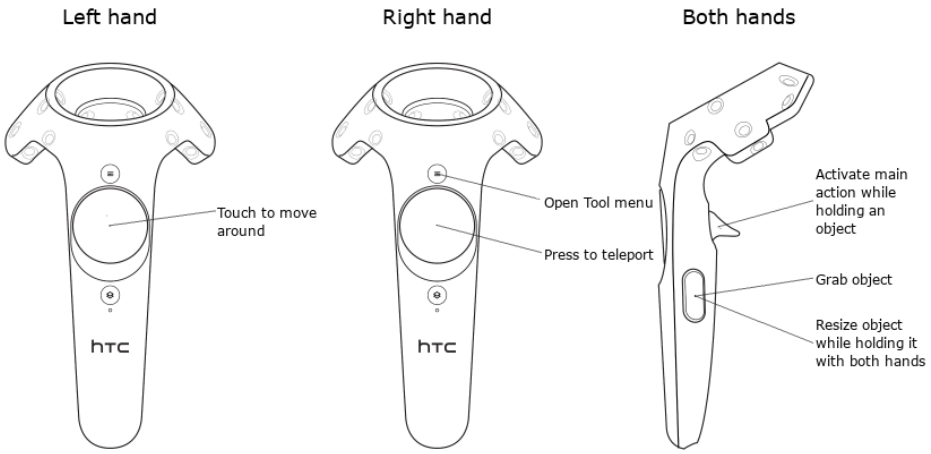
Obrázek 3.7: Návrh panelu diegetického uživatelského rozhraní zobrazující detail 3D objektu.

3.3.4 Detail 3D objektu

3D Objekt je další entita vystupující v projektové databázi. Je to abstraktní vrstva skrývající vlastní *Mesh* modelu, textury a další. Při získávání informací o konkrétním objektu pomocí API se v odpovědi mimo jiné nachází i stav schválení obou rolí a status, který indikuje v jaké fázi schvalovacího procesu objekt je. Všechny udáje jsou uživateli zobrazené na tomto panelu. V dolní části panelu je tlačítko *Inspect 3D Object*, které na pozadí stáhne a naimportuje do scény adekvátní model, tak aby s ním mohl uživatel interagovat a zkoumat ho. Náčrt je možné najít na obrázku 3.7.

3.3.5 Detail schvalovacího procesu

Tento panel zachycuje aktuální stav iterace, ve které se daný 3D objekt nachází v rámci schvalovacího procesu. K vidění jsou zde i všechny přidružené komentáře od předchozích schvalovatelů a uživatel pomocí tlačítka může přidat k iteraci vlastní komentář. V levém dolním rohu se nacházejí tlačítka pro schválení nebo zamítnutí objektu. Po bližším prozkoumaní a okomentovaní tak tady končí pro jednu roli schvalovatele celý proces. Je dobré zmínit, že má práce již pracuje s nejnovější verzí privátního API a bere v potaz nově implementované stavy 3D objektu v celém procesu. Jedná se o stavy *UNFINISHED*, *FINISHED*, *SUBMITTED*, *APPROVED* nebo *DECLINED* a *FINALIZED*. Pro lepší pochopení významu jednotlivých stavů odkazuji na kapitolu 2.3. Mimo vlastní zobrazení stavu s ním lze i pracovat, například uživateli nabídnout možnost odeslat daný objekt na schválení pokud je ve stavu *FI-*



Obrázek 3.8: Návrh popisu ovládání levého a pravého ovladače.

NISHED nebo ho stáhnout a importovat do scény, pokud je *SUBMITTED*. Je důležité poznamenat, že při každé změně v rámci schvalovacího procesu se mimo jiné upravují data struktury a 3D objektu. Na tyto změny je nutné reagovat komunikací s API a překreslením panelů uživatelského rozhraní.

3.3.6 Informace o uživateli a záhlaví

Mimo měnící se panely je součástí rozhraní také stálý panel, zobrazující informace o aktuálně přihlášeném uživateli, jako je jeho email, uživatelské jméno a přiřazená role v rámci schvalovacího procesu. Tyto údaje sloučí k lepší orientaci a ověření funkčnosti napojení aplikace na privátní API. Vzhledem k tomu, že samotné přihlášení probíhá přes *Launcher*, tyto parametry jsou programu předány skrze argumenty spuštění. V horní části rozhraní je situované záhlaví měnící svůj text v kontextu aktuálně otevřeného panelu.

3.3.7 Popis ovládání

Vzhledem ke složitosti aplikace a počtu různých interakcí s 3D modely jsem se rozhodl navrhnout grafické zobrazení nastavení ovládání, aby se uživatel dokázal rychle zorientovat, k čemu jsou tlačítka na ovladačích. Obrázek 3.8 je umístěn přímo ve scéně ve virtuálním prostoru, není však součástí hlavního menu. Stručně popisuje, co jednotlivá tlačítka na levém a pravém ovladači dělají a usnadní tak uživateli práci.

3.4 Sada nástrojů pro komentování

Jednou z nejdůležitějších částí schvalovacího procesu je komentování vytvořených 3D modelů. Je nutné poskytnout grafikům a historikům nezbytné nástroje ve virtuálním prostoru, pomocí kterých dokáží model řádně ohodnotit. Níže popisuji nástroje, které jsem sám navrhl na základě předešlé analýzy interakcí s objekty ve VR.

3.4.1 Mikrofon

Mezi největší limitace dnešních aplikací ve VR se určitě řadí schopnost získání textového vstupu od uživatele. Manuální vyťukávání písmen na simulované klávesnici pomocí laserových ukazovátek je velmi nepraktické a nepřirozené. Studie z roku 2019 [26] porovnává čtyři ruzné metody pro zadávání textu, ovšem i pomocí té nejlepší byl uživatel schopen napsat v pruměru maximálně 21 slov za minutu, což s porovnáním pruměrného počtu slov, které dokáže člověk napsat na klávesnici, není optimální. Proto jsem se rozhodl navrhnout mikrofon, jako jeden z nástrojů pro komentování 3D modelů. K vlastnímu rozpoznání textu z řeči využiji nějakou externí sluzbu, nejspíše *Google Speech to text API*. Tato funkcionalita zajistí pohodlné a intuitivní nahrání vlastního komentáře a jeho následné uložení do databáze v textové podobě.

3.4.2 Štětec

Nejen textový vstup je duležitý od schvalovatele. Možnost kreslení v 3D prostoru poskytuje další hloubku v předané informaci. Grafik nebo historik tak může pomocí nástroje *Štětec* například zakroužkovat oblast modelu, která potřebuje předělat nebo bodově označit určité nedostatky. Nakreslené obrazce se ve výsledku neukládají společně se zkoumaným modelem. Slouží pouze k lokální vizualizaci a je možné je například vyfotit a uložit pomocí nástroje *Fotoaparát*. V případě zahlcení scény změtí čar bude nástroj schopen tyto změny resetovat do původního stavu. Zároveň umožní uživateli výběr barvy tak, aby nedocházelo se smícháním barev v pozadí a obrazec tak mohl vyniknout. Šířka stopy štětce je pevně dána. Pro zachycení detailů u menších objektů je bude nutné prvně korektně naškálovat.

3.4.3 Metr

Nástroj, pomocí kterého může uživatel ve virtuálním prostoru změřit délku v reálných jednotkách. Tuto funkcionalitu využijí spíše historici, protože přesné měření je pro ně velmi důležité. Je nutno vzít na vědomí, že po zvetšení nebo zmenšení zkoumaného modelu ztrácí měření smysl, alespoň tedy pro implementaci, která se škálou měřeného předmětu nepočítá. Spolehnout se dá na vestavěný jednotkový systém přímo v *Unity Engine 3D*, který zaručí přesné mapování vzdálenosti jedna ku jedné.

3.4.4 Fotoaparát

Návrh fotoaparátu vznikl z důvodu nutnosti zachycení vizuálních dat, sloužících jako jiný typ komentáře k modelu než textový. Uživatel tak pomocí nástroje *Štětec* zaznačí nedostatky daného modelu přímo ve scéně a pomocí fotoaparátu vytvoří snímek obrazovky (*screenshot*), který později přidruží k iteraci schvalovacího procesu jako komentář typu obrázek.

3.4.5 Menu pro výběr nástroje

Po stisknutí primarního tlačítka na ovladači se uživateli zobrazí jednoduchý panel s tlačítky, reprezentující jednotlivé nástroje. Uživatel si tedy může vybrat, který nástroj chce používat s tím, že může být aktivní pouze jeden nástroj. Po výběru jiného se předchozí smaže. Zabrání se tak zahlcení scény příliš mnoho objekty.

3.5 Komunikace s privátním API

V této podkapitole se krátce věnuji navrhnuté komunikaci s privátním API, které jsem detailněji analyzoval v kapitole 2.1.1. Toto shrnutí by mělo usnadnit vývoj budoucích klienstkých aplikací, které s ním budou také komunikovat. V době psaní této práce jsem využíval verzi **3.0.5**, jehož dokumentace se nachází na této webové stránce v aplikaci Swagger [8].

Níže popisuji jednotlivé koncové body (dále jen anglicky *endpoint*) a k čemu je má aplikace využívá. Součástí popisuje je vždy i *mock-up* třída, sloužící pro deserializaci dat z odpovědi serveru. Na první pohled udává jasnou strukturu očekávaných dat, se kterými se dá jednoduše nakládat bez složitého parsování řetězce.

/auth/googlesigin Pomocí tohoto endpointu se uživatel ověřuje a přihlašuje do privátního API. V odpovědi od serveru dostává takzvaný *Bearer token*, který je vyžadován v hlavičce všech dalších odchozích požadavků. Vyčerpávajícím způsobem je proces přihlášení rozepsán v kapitole 3.1.

```
public class UserJSONDeserialized
{
    public string id;
    public string email;
    public string username;
    public string createdDate;
    public string name;
    public string role;
    public string status;
}
```

[8] https://app.swaggerhub.com/apis/sivakdom/private-api/3.0.5/

/structures Jak už jsem několikrát zmínil, struktura zastřešuje většinu ostatních entit. V případě finální aplikace stačí jednoduchý **GET** požadavek vracející seznam všech aktuálních záznamů. Ty se poté zobrazí na panelu všech struktur, který je součástí uživatelského rozhraní. Z *mock-up* třídy níže je vidět jasná závislost struktury a entity 3D objekt. Odráží se zde interní databázová reprezentace vazby 1:N. Součástí schvalovacího procesu je sice schvalování jednotlivých variant dané struktury (reprezentující entitu 3D objekt), není však nutné komunikovat přímo s endpointem */3Dobjects*. V každém případě se dá vystačit pouze s rodičovskou jednotkou struktury. Právě proto zde uvedu i *mock-up* třídu pro deserializaci 3D objektu, aby bylo jasné, jaká data obdržíme po vylistování všech struktur. Kvůli jmenným konvencím proměnných ve skriptovacím jazyce C# využívám na místo číslovaní název *TDObject*. Tento název se běžně používá napříč projektem VMČK a výrazy jsou za sebe kolikrát zaměňovány.

```
public class StructureJSONDeserialized
{
    public string id;
    public string city;
    public string name;
    public string description;
    public string createdDate;
    public string href;
    public LocationJSONDeserialized location;
    public string userId;
    public TDObjectJSONDeserialized[] allVariants;
}
public class TDObjectJSONDeserialized
{
    public string id;
    public string name;
    public bool isApprovedByApprover;
    public bool isApprovedByHistorian;
    public string createdDate;
    public string href;
    public int[] version;
    public string status;
    public string structureId;
    public string modelId;
    public string userId;
    public TextureJSONDeserialized[] textures;
}
```

/models Se využívá v aplikaci při stahování konkrétního modelu 3D objektu. Odpověď z API serveru obsahuje pouze binární řetězec, reprezentující přímo soubor modelu, není tedy nutné vytvářet třídu pro deserializaci. Tím se však částečně ztrácí informace o modelu, například jeho typ. Není možné předem zjistit, s jakou koncovkou se má binární řetezec uložit jako soubor. Při návrhu API se s tím počítalo a poté, co jsem nahlédl do interní databáze jsem zjistil, že tabulka **Model** obsahuje mimo jiné i *filename*, kde je jasně uvedena koncovka souboru, tedy typ modelu. Tato data však v aktuální verzi API nejsou zpřístupněna volajícím. Převážná většina uložených modelů v databázi mají koncovku *.obj*, což je jeden z nejrozšířenějších formátů, jak ukládat 3D geometrii. Z těchto důvodů má aplikace podporuje pouze stahování a import modelů typu **OBJ**.

/users Endpoint poskytující potřebnou funkcionalitu pro správu uživatelů. Vedle vypsání všech uživatelů je možné uživatele přidávat, upravovat či mazat. Pozor na to, že po založení novéhu učtu je označen jako *new* a není aktivován. Všechny požadavky, které mají v hlavičce jeho *Bearer token* jsou automaticky zamítnuty s důvodem nedostatečného oprávnění. V době psaní této práce neexistuje žádný centralizovaný systém spravující všechny uživatele. Je nutné ručně zaslat **PUT** požadavek, upravující práva účtu nebo ručně přepsat záznam v databázi. Z výše uvedených vzorových tříd lze vidět, že autor daných entit je zmíněn svým identifikačním číslem. Má aplikace využívá tento endpoint pro získání více informací o daném uživateli dle ID. API poskytyje rozhraní pro vylistování všech uživatelů, bohužel však bez možnosti filtrace pomocí identifikátoru. Nadbytečná iterace mezi výsledky je tedy nepraktická a v budoucnosti neudržitelná, vzhledem k nárůstu počtu uživatelů. Server vrací data ve stejné struktuře, jako v případě endpointu */auth/googlesignin*. Nebudu zde tedy opakovat definici *mock-up* třídy *UserJSONDeserialized*.

/approvals Tato část API endpointů vznikala během nejnovější iterace rukou Dominika Siváka, kterou jsem analyzoval v kapitole 2.1.1. Pokusím se v následujících větách stručně popsat důležité zdroje pro implementaci schvalovacího procesu ze strany klienstké aplikace.

Nejdůležitější entitou je **ApprovalProcess**. Slučuje pod sebe strukturu, ke které se váže, různé modelové iterace a varianty, včetně datumu vytvoření. Existuje jakási paralela mezi vazbou **Structure** a **3DObject**, a **ApprovalProcess** a **ModelIteration**. Každý proces tedy může mít více modelových iterací. Vztah mezi strukturou a schvalovacím procesem je pak definován jako vazba 1:1. Každá struktura tedy můžeme mít aktivní pouze jeden schvalovací proces.

Součástí tohoto procesu jsou modelové iterace a varianty. Varianty reprezentují stav objektu po aplikaci automaticky generovaných textur v rámci druhé části schvalovacího procesu, který již není v rozsahu této práce, jak

popisuji v kapitole 2.2. Na druhou stranu modelové iterace, které jsou pro finální aplikaci nezbytné, udržují informaci o stavu schválení oběma rolemi historik a grafik, identifikátor zkoumaného 3D objektu, datum a v neposlední řadě všechny komentáře schvalovatelů. Vzhledem k tomu, že modelová iterace musí být součástí schvalovacího procesu, je nutné zkontrolovat, jestli daná struktura dle identifikátoru má už nějaký aktivní proces a pokud ne, tak ho vytvořit. Až poté lze *POST* požadavkem přes endpoint */approvals/{approvalId}/iterations/models* vytvořit danému 3D objektu schvalovací iteraci. Důležitý je však stav objektu. Musí být ve stavu **FINISHED** a nesmí být součástí žádné jiné modelové iterace, jinak vytvoření selže. Aplikace tedy zobrazuje všechny aktuální modelové iterace na panelu detail schvalovacího procesu, společně s jejich statusem schválení od obou rolí.

Přijmutí či případné zamítnutí iterace probíhá přes */approvals/iterations/-model/{modelIterationId}/approval nebo rejection*. V těle požadavku se uvádí, za jakou roli se akce vykonává. Na backendu se následně ověří, že majitel *Bearer tokenu* má skutečně danou roli a oprávnění příkaz provést. To vše je reflektováno ve stavu iterace, který se adekvátně mění s každou další aktualizací procesu.

```
public class ApprovalProcessJSONDeserialized
{
    public string id;
    public string name;
    public bool active;
    public string phase;
    public string createdDate;
    public string structureId;
    public ModelIterationJSONDeserialized[] modelIterations;
}
public class ModelIterationJSONDeserialized
{
    public string id;
    public string name;
    public string approvalProcessId;
    public string tdObjectId;
    public string historianApprovingId;
    public string historianDecliningId;
    public string graphicianApprovingId;
    public string graphicianDecliningId;
    public string finalizedDate;
    public string createdDate;
    public CommentJSONDeserialized[] comments;
}
```

/comments Uživatel má možnost přidat libovolný počet komentářů ještě předtím, než schválí nebo zamítne zkoumaný model. K vytvoření komentáře použije výše navržený nástroj mikrofon, který převádí mluvené slovo na text. Komentář je úzce spojen s modelovou iterací, nedává totiž smysl, aby se vázál k 3D objektu nebo modelu. Po jejich úpravě nebo aktualizaci už nemusí mít význam. Všechny komentáře k dané iteraci jsou v aplikaci vizualizované na panelu detail schvalovacího procesu pomocí **GET** požadavku, na který server vrací odpověď strukturovanou dle níže uvedené *mock-up* třídy.

```
public class CommentJSONDeserialized
{
    public string id;
    public string type;
    public ContentJSONDeserialized content;
    public string byId;
    public string createdDate;
    public UserJSONDeserialized by;
}
```

3.6 Interakce s historickými předměty

V této kapitole navrhuji tři základní interakce s historickými 3D objekty. Slouží k usnadnění práce během zkoumání jejich kvality a měly by být natolik intuitivní, aby nijak neomezovali uživatele.

Uchopení V první řadě se jedná o samotné uchopení objektu a následný pohyb a rotaci s ním. Zde existují dvě varianty jak takový předmět vzít do ruky. Využít se dá takzvaného pivota, který napevno určí, jak se daný objekt uchopí, bez ohledu na polohu ovladače. Tato verze se většinou používá u zbraní nebo nástrojů, protože vyžadují určitý směr a pozici držení (například správné držení sekery). Ve finální aplikaci se tato varianta úchopu využívá u výše zmíněných nástrojů pro komentování (mikrofon, štětec, atd.). Pro manipulaci s historickým model je to však nepraktické. Proto jsem navrhl *Odsazovací metodu*, která si uloží původní polohu ovladače před interakcí s modelem a následné translace jsou počítané relativně k ní. To umožní uchopit předmět z jakékoliv strany na libovolném místě a nebude se automaticky přemísťovat k nějakému pivotu. Uživatel uchopí předmět do ruky poté, co stiskne a drží *Grip* tlačítko na ovladači. Jakmile stisk uvolní, objekt zůstane viset ve vzduchu, protože na něj záměrně nepůsobí gravitace.

Wireframe Jednou z dalších mnou navržených interakcí, důležitou především pro grafiky, je možnost zobrazení drátového modelu daného objektu (známe také jako *Wireframe*). Uživatel tak může do detailu prozkoumat interní polygonovou strukturu 3D modelu, bez potřeby použít jiné externí modelovací

programy, jako je například Blender nebo 3Ds Max. Tato funkcionalita je namapována na *Trigger* tlačítko ovladače a během toho, co uživatel drží v ruce historický exponát je možné přepínat mezi standardní texturou a síťovým modelem.

Škálování V neposlední řadě je možné zkoumaný objekt libovolně škálovat v prostoru. K vlastnímu zmenšování či zvetšování dochází poté, co ho uživatel uchopí oběma rukama a hýbe pažemi k sobě nebo od sebe. Dokáže tak ručně dle libosti nastavit velikost zkoumaného modelu. Aby se předešlo krajním situacím, jako je třeba to, že objekt už je tak veliký, že nejde rozumně zmenšit, je nutné omezit škálovaní maximálním a minimálním faktorem velikosti.

Kapitola 4

Realizace

V této kapitole se věnuji popisu implementace finálního prototypu. Hlavním učelem realizované aplikace je demonstrovat navržené interakce s historickými modely vznikajícími v rámci projektu VMČK a poskytnout platformu pro schvalovací proces. Prototyp vzniká na základě analyzovaných softwarových nástrojů v kapitole 2.5 a navržených funkcionalit v kapitole 3.

4.1 Launcher a přihlášení

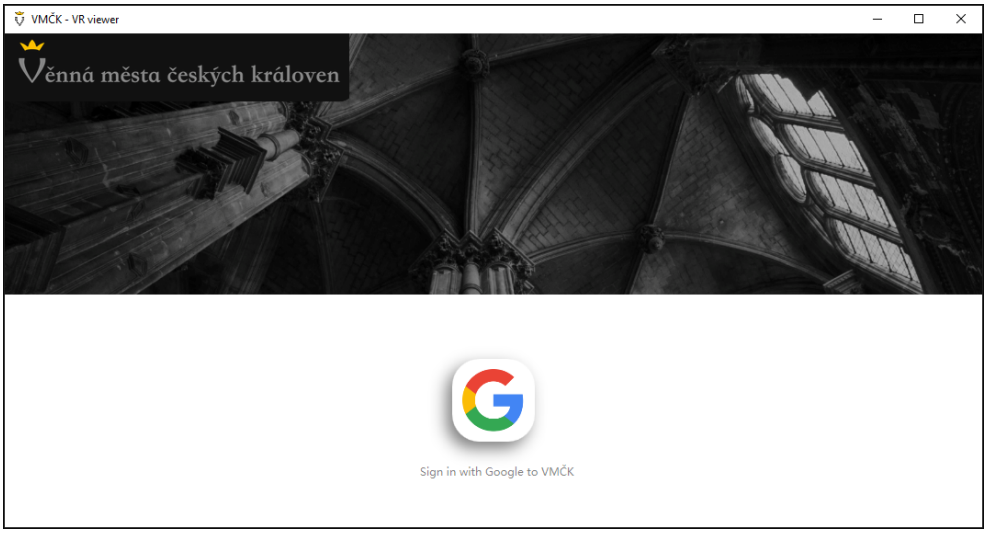
Jednou z největších motivací vytvoření samostatného *Launcheru* pro VR aplikaci je pro mě usnadnění přihlášení pomocí Google nebo školního účtu. Pro úspěšné dokončení přihlašovacího procesu pomocí protokolu **OAuth2** (popisuji v kapitole 3.1) je nutné uživatele přesměrovat pomocí webového prohlížeče na přihlašovací stránku dané autority. Nenašel jsem žádný balíček pro Unity, který by byl zdarma a zobrazoval ve virtuálním prostoru okno prohlížeče. Řešením je vytvoření jednoduché nativní aplikace pro operační systém Windows, která mimo přihlašení stahuje z internetového uložiště soubory VR aplikace, kterou následně instaluje.

Launcher je WPF (Windows Presentation Foundation) aplikace stavějící na frameworku **.NET** psaná v jazyce C#. Rozvržení grafických částí realizuji pomocí deklarativního jazyka XAML (Extensible Application Markup Language), který velmi zjednodušuje tvorbu uživatelského rozhraní. Po úspěšném přihlášení uživatele dochází ke kontrole lokální a remote verze VR aplikace. Pokud se liší, automaticky se zahájí stahování, jehož status je vizualizován ukazatelem průběhu. Součástí postavené verze *Launcheru* je konfigurační soubor **App.config** obsahující adresu privátního API projektu VMČK, adresu vzdáleného uložiště odkud se stahují soubory VR aplikace a klienstká ID potřebná během přihlašování. Tyto parametry je možné dle libosti upravit pokud to je třeba.

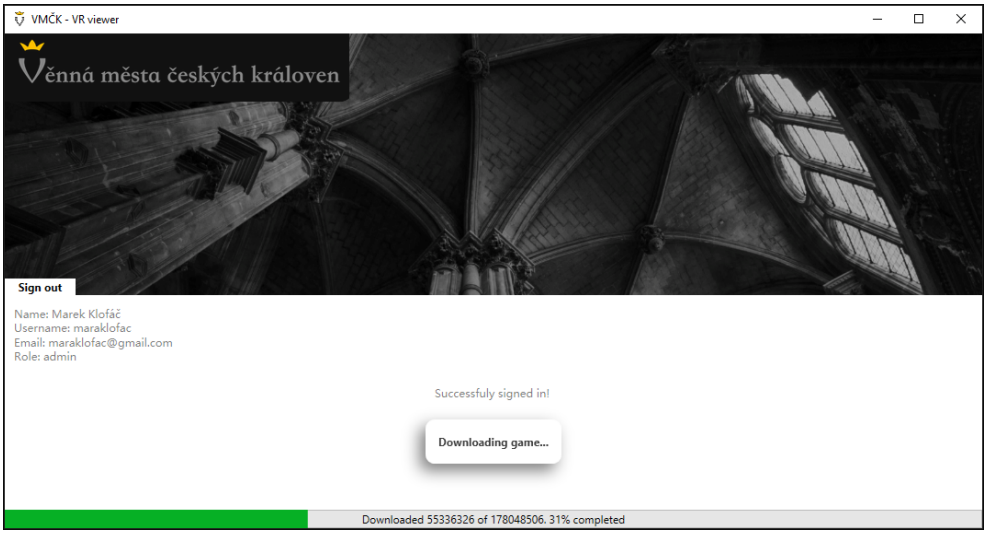
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="AUTH0_DOMAIN" value="dev-zl88hgcd.eu.auth0.com" />
        <add key="AUTH0_CLIENT_ID" value="Net1zt..............." />
        <add key="GCP_CLIENT_ID" value="102209.................." />
        <add key="VERSION_REMOTE_LOCATION"
            value="https://www.dropbox.com/s/62uh1beig1cpo8n/version.txt?dl=1"
            />
        <add key="BUILD_REMOTE_LOCATION"
            value="https://www.dropbox.com/s/y21e57f5b7rkih6/build.zip?dl=1"
            />
        <add key="VMCK_API_SERVER"
            value="http://109.123.202.213:3000"/>
    </appSettings>
</configuration>
```

Z výše uvedených parametrů si je možné všimnout, že v přihlašovacím procesu používám **Auth0**. Tato služba poskytuje klientským aplikacím jednoduchou cestu autentifikace a autorizace u autorit třetích stran jako je Google, Facebook, Twitter a další. Mimo jiné také poskytuje centrální správu uživatelů a řeší přístupová práva. Důvodem použití této služby je snaha obejít problém, na který jsem narazil během implementace. Otevření okna prohlížeče a přesměrovanání uživatele na přihlašovací stránku Google z *Launcheru* se vyhodnotí jako nedůvěryhodné a proces selže. Díky přímé integraci **Auth0** služby s **WPF** aplikací probíhá přihlašování externě a uživatel je přesměrován do dedikovaného okna v rámci *Launcheru*, kde se přihlásí. Jednoduchý tutoriál s funkčním demem je možné najít na webových stránkách této služby [27].

```
// Callback function, triggered upon the click on a login button
private async void LoginButton_Click(object sender, RoutedEventArgs e)
{
    // This client will connect for us to Google's OAuth app
    client = new Auth0Client(new Auth0ClientOptions
    {
        Domain = ConfigurationManager.AppSettings["AUTH0_DOMAIN"],
        ClientId = ConfigurationManager.AppSettings["AUTH0_CLIENT_ID"]
    });

    // Specify, that login should be only via Google OAuth2
    var extraParameters = new Dictionary<string, string>();
    extraParameters.Add("connection", "google-oauth2");

    // Asynchronously await for user to finish logging in
    LoginResult loginResult = await
        client.LoginAsync(extraParameters: extraParameters);
}
```



Obrázek 4.1: Úvodní stránka *Launcheru* kde se uživatel může přihlásit.



Obrázek 4.2: *Launcher* stahující VR aplikaci po úspěšném přihlášení uživatele.

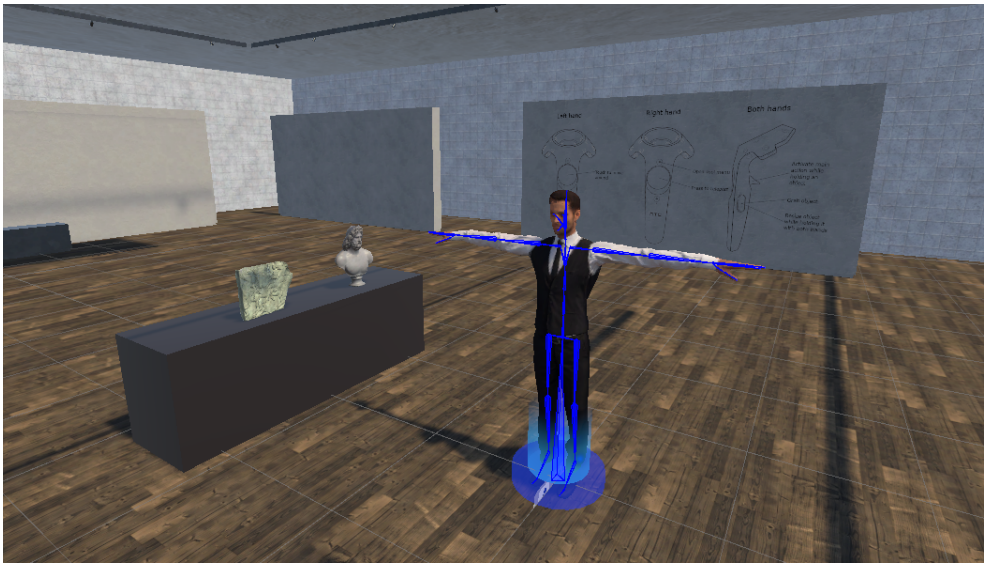
Po ověření přihlášení na straně Auth0/Google aplikace posílá obdržený identifikační token směrem do privátního API VMČK, čímž dokončuje celý proces přihlášení a uživatel může zapnout VR aplikaci. Na pozadí jí *Launcher* předává informace o přihlášeném uživateli pomocí parametrů spuštění procesu. Na základě těchto dat je poté schopna odesílat další požadavky na API.

Při tvorbě uživatelského rozhraní *Launcheru* jsem se inspiroval grafickým návrhem webové stránky projektu VMČK [9]. Finální design úvodní stránky a stahování VR aplikace po příhlášení je možné vidět na obrázcích 4.1 a 4.2.

4.2 Virtuální prostředí

Po spuštění VR aplikace se uživatel ocitne ve fiktivním světě, respektive ve virtuální galerii (obrázek 4.3). Toto prostředí má navodit patřičnou atmosféru pro zkoumání historických 3D modelů. Uprostřed galerie se nachází pracovní stůl, na kterém jsou odloženy dva testovací modely na nichž si uživatel může vyzkoušet interakce před vlastním importem reálných objektů ke schválení z API. Tyto ukázkové modely jsou volně dostupné a stáhnul jsem je z webového archivu britského národního muzea [28]. Součástí prostředí jsou vysoké panely, na které jsem umístil diegetické uživatelské rozhraní. Model galerie jsem zakoupil online [29]. Vytvořil ho profesionální 3D modelář Marcin Lubecki, který se věnuje především interiéru a realističnosti. Kvalitní vzhled galerie obohacuje uživatelský zážitek.

[9] Webová stránka projektu VMČK: https://www.kralovskavennamesta.cz/



Obrázek 4.3: Virtuální prostředí galerie s avatarem reprezentující tělo uživatele.

4.3 Pohyb ve virtuálním prostoru

Do finálního prototypu aplikace jsem implementoval dvě různé metody pohybu. První z nich je teleportace a ovládá se *Touchpad* tlačítkem na pravém ovladači. Realizaci mi usnadnil balíček **XR Interaction Toolkit**, který nabízí tvorbu šablonovitého objektu **Ray Interactor**. Tento objekt má na sobě připnuté potřebné skripty pro interakci s okolím (*XR Controller* a *XR Ray Interactor*). Upravil jsem alespoň vizuál vycházejícího paprsku, který je ve výchozím stavu bílý a rovný. Nyní je mírně prohnutý a na svém konci vizualizuje kuželovitým objektem cílové místo teleportace. Nedílnou součástí teleportačního systému je i skript *Teleportation Area*, který jsem připnul na herní objekt podlahy místnosti. Označil jsem tak validní oblast, kam se lze přesouvat.

Vedle teleportace jsem implementoval ještě jednu metodu pohybu a to **Continuous movement**. Po doteku palce *Touchpad* tlačítka na levém ovladači dochází k pohybu těla uživatele v prostoru. Ve výpočtu se bere v úvahu natočení uživatelovy hlavy pro korektní posun a vliv gravitace na pohyb v ypsilonové ose. Tato varianta může být pro nezkušené uživatele virtuální reality nepříjemná a způsobovat jim nevolnost. Pro mě osobně je toto preferovanější metoda pohybu.

```
public class ContinuosMovement : MonoBehaviour
{
    public XRNode inputSource;
    public float speed = 1.0f;
    public float gravity = -9.81f;

    private float fallingSpeed;
    private XRRig rig;
    private Vector2 inputAxis;
    private CharacterController characterController;

    void Update()
    {
        // Read input from controller's touchpad
        InputDevice device =
            InputDevices.GetDeviceAtXRNode(inputSource);
        device.TryGetFeatureValue(CommonUsages.primary2DAxis, out
            inputAxis);
    }

    private void FixedUpdate()
    {
        // Move with body in X and Z axis, influence by head direction
        Quaternion headYaw = Quaternion.Euler(0,
            rig.cameraGameObject.transform.eulerAngles.y, 0);
        Vector3 direction = headYaw * new Vector3(inputAxis.x, 0,
            inputAxis.y);
        characterController.Move(direction * Time.deltaTime * speed);

        // Apply gravity to movement on Y axis
        fallingSpeed += IsGrounded() ? 0 : gravity *
            Time.fixedDeltaTime;
        characterController.Move(Vector3.up * fallingSpeed *
            Time.fixedDeltaTime);
    }
}
```

K zajištění co nejpohodlnějšího a nejrealističtějšího zážitku pro uživatele jsem dle návrhu implementoval VR avatara, který má za úkol reprezentovat lidské tělo. Díky nativní podpoře inverzní kinematiky přímo ve vývojovém prostředí Unity byla realizace velmi snadná. Využil jsem komponenty **Rig Builder**, **Two Bone IK Constraint** pro ruce modelu a **Multi-parent Constraint** pro hlavu. Po definici parametrů a bodů omezení, které ovlivňují chování a pohyb rodičovských objektů v hierarchii už stačilo pouze namapovat VR hardware (ovladače a headset) na tyto body omezení. Ty poté kopírují v reálném čase jejich pohyb a transformují připnutý model lidské postavy, kterou je možná vidět na obrázku 4.3. Již zmiňovaný model lidské postavy je volně dostupný na internetu ke stažení [30].

4.4 Uživatelské rozhraní

Od samotného návrhu v kapitole 3.3 jsem se při implementaci diegetického uživatelského rozhraní trochu odchýlil. Je součástí scény a nachází se na jednom z panelů v galerii. Rozhraní je rozděleno do třech hlavních bloků a to do informačního panelu zobrazující data přihlášeného uživatele, vrchního panelu ukazující název aktuálně otevřené stránky a hlavního panelu, kde se postupně zobrazuje seznam všech struktur (obrázek 4.4) a detail struktury.

S uživatelským rozhraním se dá interagovat pomocí laserového ukazetele, který vychází z ovladače pokud na něj míří. Plátno, na které se UI prvky vykreslují, má nastavený *World space* renderovací mód a připnutou komponentu **Tracked Device Graphic Raycaster** pro zaznamenání výše zmíněného ovládacího laseru. Přechod mezi jednotlivými panely jsem realizoval pomocí aktivního stavu *GameObjectů*, který dle potřeby vypínám a zapínám. Návrat z detailu struktury na panel všech struktur je tak možný za použití tlačítka zpět v levém horním rohu.

Panel detailu struktury obsahuje tři podsekce, info, 3D objekty a schválení. V info části (obrázek 4.5) jsou zobrazena stažená data týkající se dané struktury včetně názvu, autora, data založení, místa nálezu a odborné deskripce. V záložce *3D objekty* (obrázek 4.6) jsou vylistovány všechny adekvátní varianty 3D objektů. Každá položka nabízí pro rychlý náhled jméno, autora, verzi, status 3D objektu a akční tlačítko. Toto tlačítko nabízí dvě možnosti uživateli, co s objektem může dělat v závislosti na jeho stavu. Pokud je ve stavu **FINISHED**, uživatel ho může odeslat ke schválení. Pokud má 3D objekt jakýkoliv jiný stav, uživateli se nabízí možnost prozkoumání modelu ve VR. Pro vizualizaci jsem opět volil vertikální skupinové rozložení s posuvníkem stejně jako u seznamu všech struktur na domovské stránce (v Unity to je UI komponenta **Scrollbar**). Není možné totiž předem odhadnout, kolik takových variant bude struktura mít. V poslední záložce *schválení* (obrázek 4.7) se nachází detail schvalovacího procesu dané struktury a všechny aktivní modelové iterace. Náhled opět nabízí jméno 3D objektu ke které je iterace vztažena, status schválení rolemi historik a grafik a tlačítko pro zobrazení detailu.

Posledním implementovaným panelem je detail modelové iterace (obrázek 4.8) kam se uživatel dostane kliknutím na tlačítko detailu v záložce *schválení*. Na této stránce je uživatel upozorněn na to, ať zkontroluje přiřazenou roli během přihlášení. Pokud se totiž pokusí schválit danou iteraci za roli kterou nemá, dojde k selhání volání API a je nutné restartovat celou aplikaci. Implementoval jsem sadu tlačítek, pomocí kterých může uživatel schválit iteraci za roli historik nebo grafik za účelem otestování jednotlivých *endpoints*. V produkční verzi této aplikace by chtělo dynamicky zobrazit pouze jedno tlačítko **Approve** a na pozadí by se v požadavku na API odesílala přiřazená role. V aktuálním stavu backend řešení projektu neexistuje žádný nástroj pro správu uživatelu, je tedy nutné ručně nastavit práva nově přihlášeným. Pokud

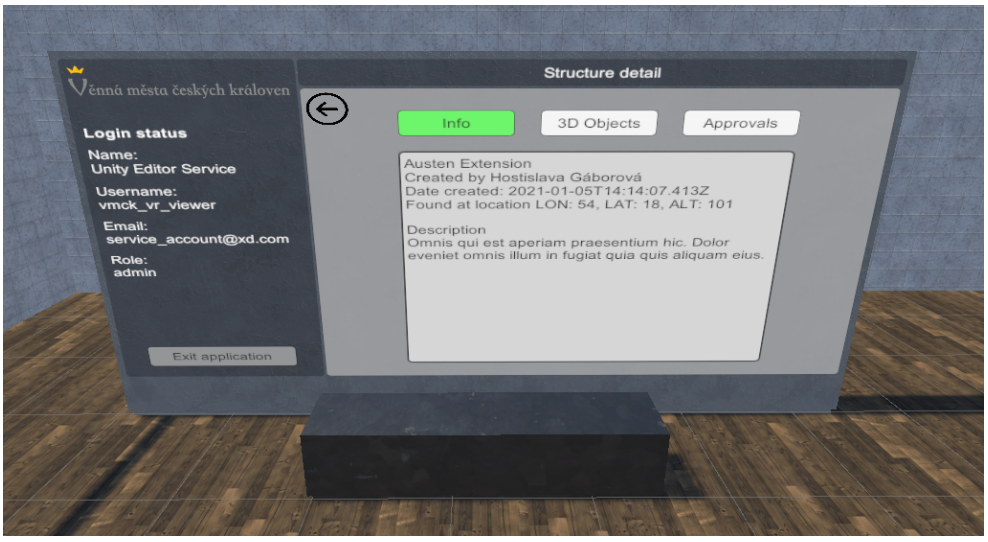
jste do aplikace přihlášení jako administrátor, schválení jako historik a grafik projde bez problému. Na panelu detail modelové iterace je možné najít po pravé straně všechny předchozí komentáře, které jsou k ní vzpjaty. Možností je samozřejmě přidání vlastního komentáře pomocí nástroje *Mikrofon*.

Většina komunikace s privátním API probíhá na základě interakce uživatele a rozhraní. Pro zastřešení komunikace jsem vytvořil komponentu **VMCKConnector**, což je skript obsahující všechna potřebná data a funkce pro navázání spojení se vzdáleným API serverem. Veškerá volání se vykonávájí asynchronně pro nepřerušovaný běh aplikace. Jakmile server odpoví, **VMCKConnector** vyvolá specifickou událost s odpovědí, na kterou se zbytek komponent v aplikaci navěsí a reaguje na ní.

Jednoduchým příkladem volání je vylistování všech dostupných struktur. Tato korutina se spouští v komponentě **MainMenuManager** při inicializaci hlavního panelu všech struktur, kde dochází i k navěšení na událost *GetStructuresCallDone*. Po doběhnutí níže uvedeného kódu v API konektoru je tato událost vyvolána a komponenta hlavního menu dynamicky zobrazí obdržená data všech struktur uživateli. Ostatní volání jsou založena na podobném asynchronním přístupu a spouštění událostí.

```
public IEnumerator GetStructures()
{
    string url = VMCK_API_SERVER + "/structures";
    using(UnityWebRequest www = UnityWebRequest.Get(url))
    {
        www.SetRequestHeader("Authorization", "Bearer " + bearerToken);
        yield return www.SendWebRequest();
        if(www.isNetworkError || www.isHttpError)
        {
            CallFailed(url, www.error);
        }
        else
        {
            GetStructuresCallDone.Invoke(www.downloadHandler.text);
        }
    }
}
```

Obrázek 4.4: Seznam všech struktur, hlavní stránka uživatelského rozhraní.



Obrázek 4.5: Detail struktury, záložka zobrazuje informace o struktuře.

Obrázek 4.6: Detail struktury, záložka zobrazuje varianty 3D objektů.

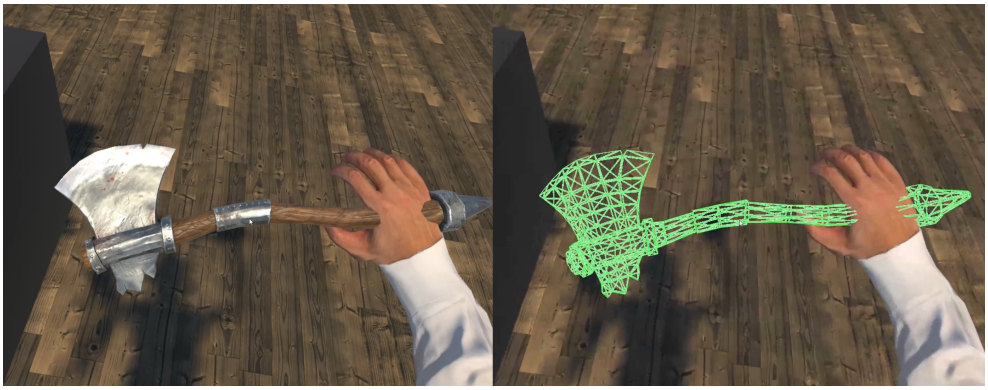
Obrázek 4.7: Detail struktury, záložka zobrazuje modelové iterace v rámci schvalovacího procesu struktury.

Obrázek 4.8: Detail modelové iterace, panel zobrazuje připojené komentáře.

4.5 Interakce s objekty

Pro realizaci interakcí s historickými modely jsem implementoval vlastní třídu **XRExhibitGrabInteractable**, která rozšiřuje základní třídu **XRGrabInteractable** z balíčku **XR Interaction Toolkit**. Má třída plní dvě funkcionality, a to uchopení objektu v místě doteku (*Odsazovací metoda*) a škálování objektu. Úchop jsem realizoval pomocí funkcí *OnSelectEntered* a *OnSelectExited*, které jsou vyvolané interními Unity událostmi. Během doteku ovladače a předmětu se prvně uloží výchozí pozice a rotace ovladače a následně se přesune na místo objektu. Je to prakticky obrácený postup klasické interakce, kdy předmět přiskakuje k ruce. Poté co uživatel předmět upustí, ovladači se nastaví předtím uložená pozice a rotace. Díky tomu může uživatel model libovolně natáčet a manipulovat s ním. Pro registraci doteku je potřebná komponenta *Rigidbody*, která určuje vliv fyzikálního systému na objekt. Zaklikl jsem v ní možnost **IsKinematic** a odškrtl **Use Gravity**, aby model uživateli neustále nepadal na zem a mohl ho v klidu ze všech stran zkoumat.

V metodě **Update** ve třídě **XRExhibitGrabInteractable** se každý snímek kontroluje a ukládá aktuální vzdálenost ovladačů, pokud aktivně drží předmět. Jestliže je momentální vzdálenost ovladačů větší nebo menší než předchozí snímek, historický model se zvětší, respektive zmenší. Ve výpočtu počítám s jistou odchylkou, která toleruje například třes rukou. Tyto mikro pohyby jsou ignorovány a je nutné explicitně roztáhnout ruce. Celý proces škálování je omezen minimálním a maximálním násobkem výchozí škály modelu, aby nedošlo k moc velkému zvětšení nebo moc malému zmenšení.



Obrázek 4.9: Rozdíl mezi normálním a síťovým zobrazením modelu ve finální aplikaci.

Pokud uživatel drží exponát v ruce, stisknutím *Trigger* tlačítka přepne zobrazení do síťového (*Wireframe*). Tuto činnost jsem definoval jako reakci na **OnActivate** událost z **XRExhibitGrabInteractable** skriptu. Volá se funkce **ChangeShader** v druhé komponentě **WireframeRenderer**, kterou jsem také implementoval. Tento jednoduchý skript přepíná mezi výchozím shaderem objektu a síťovým shaderem *SuperSystems/Wireframe-Transparent*, který je součástí sady shaderů, dostupných jako balíček do Unity [31]. U složitějších objektů je nutné měnit shader jejich potomků. Skript **WireframeRenderer** prochází celou hierarchii objektu a postupně mění všechny shadery. Výsledné síťové zobrazení je možné vidět na obrázku 4.9

4.6 Nástroje

Z navržené sady nástrojů pro komentování z kapitoly 3.4 jsem realizoval většinu. Jediné co jsem z časových důvodů nestihl je *Fotoaparát*. Pro výběr nástroje jsem vytvořil jednoduchý panel s tlačítky (obrázek 4.10), který se vždy zobrazí směrem k uživateli, nezávisle na jeho otočení a s určitým odsazením od ovladače. Z toho důvodu dochází k výpočtu rotace panelu v lokálním prostoru ovladače. Počítání souřadnic a rotace panelu ve světovém prostoru není dostačující a korektní. Jednotlivá tlačítka naznačují o jaký nástroj se jedná pomocí ikon a po jejich stisknutí se daný nástroj objeví v prostoru přímo před uživatelem.

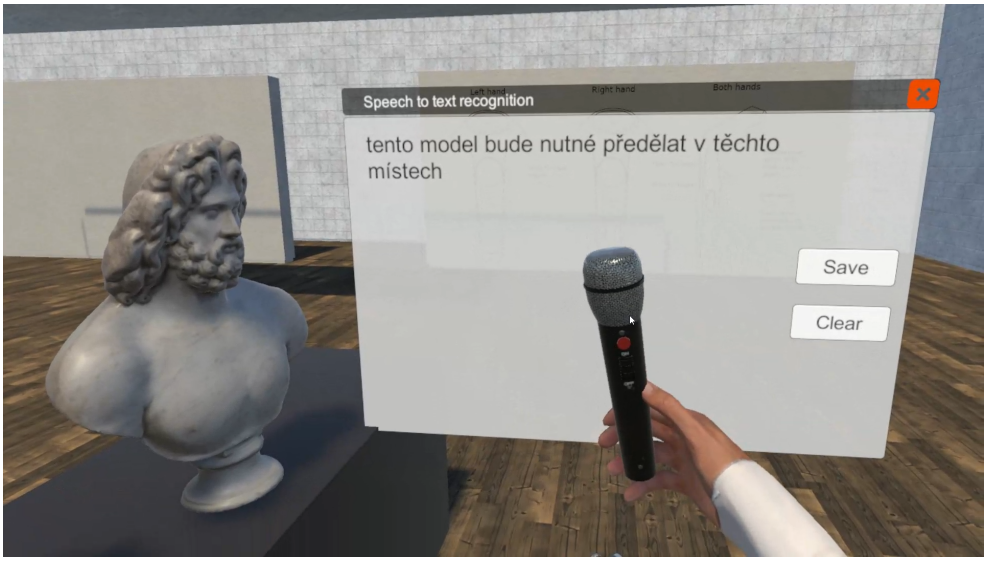
Mikrofon Vlastní model mikrofonu [32] jsem upravil a přidal barevný cylindr signalizující aktivitu mikrofonu. Je uchopitelný stejně jako ostatní předměty, pomocí *Grip* tlačítka a stistknutím *Trigger* tlačítka začne nahrávání audio stopy. V reálném čase, po dobu stisknutí tlačítka, se tato stopa odesílá do služby **Google Speech API**, kde se překládá díky *Speech-to-text* rozpoznávání na text, který se vrací do aplikace a vypisuje se do okna. Uživatel tak nemusí ručně zadávat komentáře a používá svůj hlas. Pro komunikaci s Google API využívám balíček do Unity [33], který ve výchozím nastavení podporuje pouze angličtinu. Po průzkumu kódu jsem byl schopen doimplementovat podporu českého jazyka přidáním slovníku jazyků a upravení požadavku pro API volání. Výběr jazyka je tak nastavitelný přímo z Unity Editoru. Výše odkazovaný repozitář obsahuje stručný návod jak nastavit službu na Google Cloudu a vyexportovat potřebné přístupové údaje pro komunikaci skriptů. Stažený soubor *gcp_credentials.json* je nutné umístit v projektové hierarchii do složky *Assets/StreamingAssets* a následně stačí na nějaký *GameObject* připnout komponentu **StreamingRecognizer**. Tento skript poskytuje hned několik událostí, na které se lze navěsit a reagovat tak například na obdržení částečně přeloženého textu vypsáním do příslušného okna. V odevzdané aplikaci a projektu této práce se používá má osobní instance **Google Speech API** služby a přístupové údaje k ní. Pokud budete chtít navazovat na moji práci, prosím nahraďte soubor *gcp_credentials.json* za vlastní. Finální realizaci mikrofonu je možné vidět na obrázku 4.11.

Štětec Pro základní model štetce jsem využil [34], který jsem nadále upravil a přidal kruhové těleso pro výběr barvy. Stejně jako ostatní nástroje má na sobě štětec připnutou komponentu **XR Grab Interactable**, díky které ho lze uchopit do ruky. Funkcionalitu malování v prostoru jsem realizoval pomocí komponenty **LineRenderer**, která vykresluje křivku mezi body. Nové body se tvoří u špičky štětce při pohybu nástrojem v prostoru. V **Update** funkci kreslícího skriptu se kontroluje vstup z *Touchpad* tlačítka, který ovlivňuje aktuální výběr barvy. Zvolená barva nastavuje jak hrot štětce tak i odstín kreslené křivky. Z tlačítka lze přečíst dvojici číselných souřadnic X a Y, které reprezentují polohu palce na *Touchpadu*. Tyto souřadnice následně konvertuji pomocí převodu do radiánů a matematických funkcí sinus a kosinus do barevného prostoru HSV (hue, saturation, value) a získávám tím zvolenou barvu. Úryvek kódu převáděcí funkce naleznete níže (inspirace z [35]).

```
private void ChangeBrushColor()
{
    // Calculate and normalize angle from X & Y coordinates from the
        controller
    float angle = Mathf.Atan2(touchpadInput.x, touchpadInput.y) *
        Mathf.Rad2Deg;
    float normalAngle = angle - 90;
    if(normalAngle < 0)
    {
        normalAngle = 360 + normalAngle;
    }
    // Convert to radians and calculate max X and Y coords
    float rads = normalAngle * Mathf.PI / 180;
    float maxX = Mathf.Cos(rads);
    float maxY = Mathf.Sin(rads);

    // Calculate percentage between current X, Y coords and max
    float percentageX = Mathf.Abs(touchpadInput.x / maxX);
    float percentageY = Mathf.Abs(touchpadInput.y / maxY);

    // Prepare color in HSV space
    float hue = normalAngle / 360.0f;
    float saturation = (percentageX + percentageY) / 2;
    float value = 1f;

    // Set paintbrush tip to this new color
    Color color = Color.HSVToRGB(hue, saturation, value);
    paintBrushHead.GetComponent<Renderer>().material.color = color;
    currentLineColor = color;
}
```

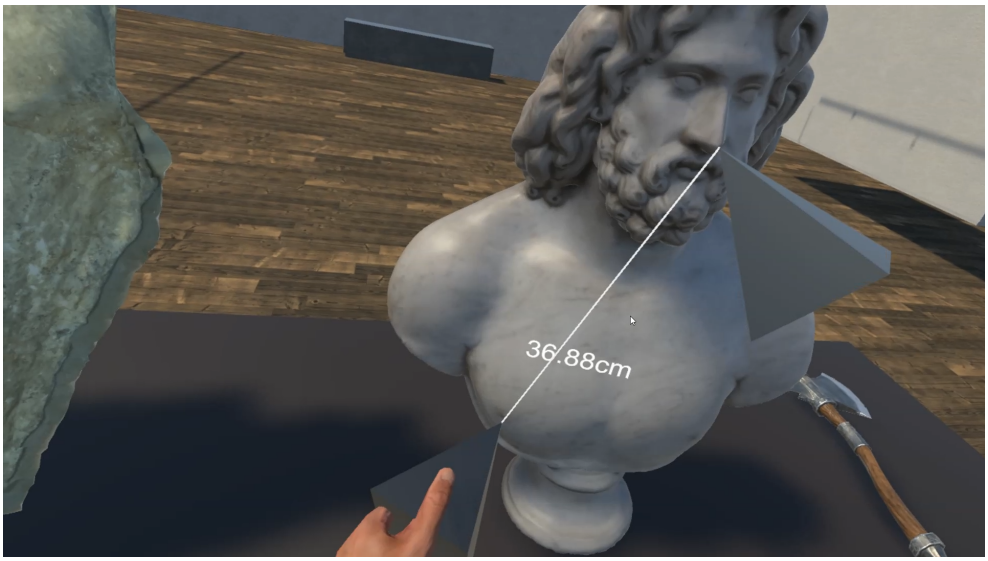
Metr Nástroj se skládá ze dvou jehlanů, mezi jejichž hroty se automaticky měří vzdálenost. Pro vizualizaci vzdálenosti jsem použil textové pole vypisující délku v centimetrech a komponentu **LineRenderer**, která vykresluje křivku mezi vrcholy jehlanů. S jehlany lze libovolně hýbat a umístit je v prostoru. Díky vestavěnému metrickému systému v Unity tento nástroj dokáže přesně měřit vzdálenost odpovídající vzdálenosti v reálném světě. Ze všech nástrojů byla implementace metru nejjednoduší a výsledek je vidět na obrázku 4.13.

Obrázek 4.10: Panel pro výběr nástrojů ke zkoumání a komentování.



Obrázek 4.11: Nástroj mikrofon společně s panelem, kde se zobrazuje mluvené slovo v textové podobě.

Obrázek 4.12: Nástroj štětec, pomocí kterého lze kreslit v prostoru.



Obrázek 4.13: Nástroj metr, který měří přesnou vzdálenost mezi hroty jehlanů.

Kapitola 5

Testování prototypu

Uživatelské testování jedné z posledních verzí prototypu VR aplikace jsem provedl v UsabilityLabu na ČVUT FIT. Jakožto moderátor testování jsem instruoval skupinu studentů celoživotního vzdělávání Univerzity Hradec Králové z předmětu *Počítačová grafika pro mírně pokročilé*. Vyučující tohoto předmětu Mgr. Klára Rybenská, Ph.D. se testování také zúčastnila. Příležitost prezentovat a testovat svoji aplikaci jsem dostal ještě předtím, než byl celý vývoj dokončen. Získal jsem však cenné názory uživatelů, pro které byla tato aplikace navržena a jejich nápady jsem do finálního vydání zčásti zahrnul. Připravil jsem specifické testovací scenáře, ověřující v té době již implementované funkcionality.

5.1 Testovací scénáře

5.1.1 Přihlášení

Odhadovaný čas
2 min.

Zaměření scénáře
Otestování funkčnosti přihlášení uživatele v *Launcheru* pomocí osobního nebo školního Google účtu a následné spuštění VR aplikace.

Výchozí stav
Spuštěný *Launcher* na úvodní stránce zobrazující přihlašovací tlačítko. Předchozí soubory VR aplikace neexistují, dojde k jejich stažení a nainstalování.

Finální stav
Uživatel je úspěšně přihlášen a vlevém horním rohu se zobrazují informace o jeho profilu včetně přiřazené role. *Launcher* úspěšně stáhnul a nainstaloval VR aplikaci a zobrazuje tlačítko ke spuštění aplikace.

Instrukce pro testera

1. Klikněte na tlačítko přihlášení.
2. Přihlašte se pomocí osobního Google účtu.
3. Zkontrolujte informace o přihlášení.
4. Počkejte, až se stáhne a nainstaluje VR aplikace.
5. Spusťte aplikaci.

Očekávané kroky

1. Kliknutí na tlačítko *Sign in with Google to VMČK*.
2. Provedení přihlášení v nově otevřeném okně Google autority.
3. Přečtení informací o přihlášení v levé části obrazovky.
4. Vyčkání na stáhnutí a instalaci VR aplikace.
5. Kliknutí na tlačítko *Play*, čímž se spustí aplikace.

5.1.2 Pohyb a ovládání

Odhadovaný čas
3 min.

Zaměření scénáře
Tento scénář má za úkol seznámit uživatele s pohybem ve virtuálním prostoru a zhodnotit dvě implementované metody pohybu, teleport a kontinuální pohyb.

Výchozí stav
Uživatel si po příhlášení v *Launcheru* nasadí VR headset a do rukou uchopí ovladače.

Finální stav
Uživatel se umí ve virtuálním prostoru pohybovat dvěma různými způsoby a seznámil se s ovládáním.

Instrukce pro testera

1. Nasaďte si headset a uchopte do rukou ovladače.
2. Zmáčkněte palcem pravé ruky velké kruhové tlačítko na ovladači a držte.

3. Namiřte vycházející paprsek z ovladače někam na zem dál od sebe.
4. Uvolněte tlačítko a přemístěte se.
5. Dotkněte se palcem levé ruky velkého kruhové tlačítka na ovladači.
6. Najděte v prostoru velkou tabuli s popisem ovládání obou ovladačů a dojděte k ní (je jedno jakou metodou).
7. Otevřete si nabídku nástrojů pro komentování stisknutím primárního tlačítka.
8. Opětovným stisknutím ho opět zavřete.

Očekávané kroky

1. Nasazení headsetu a ovladačů. Můžeme pomoci s nasazením.
2. Stisknutí *Touchpad* tlačítka na pravém ovladači a následná teleportace.
3. Dotknutí se *Touchpad* tlačítka na levém ovladači a pohyb v prostoru.
4. Chůze nebo teleportování k tabuli s popisem ovládání.
5. Kliknutí primárního tlačítka a otevření panelu nástrojů pro komentování.
6. Opakované kliknutí pro zavření panelu.

5.1.3 Zkoumání historického modelu

Odhadovaný čas
7 min.

Zaměření scénáře
Otestování funkcionality stažení a importu modelu z privátního API za běhu aplikace. Uživatel následně interaguje se staženým modelem, pomocí obou rukou ho zvětšuje a zmenšuje a zobrazuje si jeho síťový model.

Výchozí stav
Uživatel má prázdné ruce a stojí před hlavním uživatelským rozhraním.

Finální stav
Uživatel drží stažený model z API v ruce, poté co mu změnil velikost a přepnul vzhled na síťový model (*Wireframe*).

Instrukce pro testera

1. Najděte v seznamu všech struktur strukturu *Alisha Crest* a otevřete ji.
2. Otevřete nabídku *3D Objects*.
3. Prozkoumejte první uvedený objekt ve virtuální realitě.
4. Uchopte stažený model oběma rukama a zvětšte ho.
5. Uchopte model pouze jednou rukou a přepněte ho do síťového modelu.

Očekávané kroky

1. Nalezení a kliknutí na tlačítko *Alisha Crest* v seznamu všech struktur.
2. Kliknutí na tlačítko subsekce *3D Objects*.
3. Kliknutí na tlačítko *Inspect in VR* hned u prvního záznamu 3D objektů.
4. Stažení a import modelu z privátního API na pozadí.
5. Uchopení předmětu oběma rukama pomocí bočního *Grip* tlačítka na ovladačích.
6. Zvětšení modelu pohybem paží od sebe.
7. Uchopení předmětu pouze do jedné ruky.
8. Stisknutí *Trigger* tlačítka na ovladači pro změnu na síťový model.

5.2 Výsledky

Všemi testovacími scénaři prošli pouze čtyři lidé z devíti testovaných. U ostatních muselo být testování přerušeno z důvodu náhlé nevolnosti a dezorientace. Nemyslím si, že důvodem bylo špatné navržení virtuálního prostředí nebo funkcionality, spíše nezkušenost testerů s VR. Pro většinu z nich to byl první VR zážitek. Po dotázání testerů, kteří prošli všemi testovacími scénaři, jsem zjistil, že už měli předchozí zkušenost s VR. Níže shrnuji výsledky jednotlivých testovacích scénářů a předkládám návrhy, jak aplikaci na jejich základě upravit.

Přihlášení V průběhu tohoto testovacího scénáře v žádném případě nenastala chyba nebo delší zdržení. Uživatelé se dokázali intuitivně přihlásit pomocí tlačítka s logem Google. Pro většinu bylo přihlášení ve vyskakovacím okně známé, na stejný koncept už v minulosti narazili v jiných aplikacích. Zároveň ocenili v *Launcheru* zelený ukazatel průběhu, podle kterého mohli sledovat stav stahovaných souborů virtuální aplikace a její následné instalování.

Pohyb a ovládání Pro mnohé testery byla praktičtější metoda teleportace. Pohyb v prostoru pomocí palce levé ruky byl nepříjemný i pro zkušenější uživatele. **Continuous movement** metodě nepřidával ani fakt, že pohyb se zahájil po doteku palce *Touchpad* tlačítka. Docházelo tak k nevyžádanému pohybu ve scéně během nasazování ovladačů na ruce nebo když tester omylem zavadil palcem o tlačítko. Tento problém by se dal spravit přemapováním pohybové akce na stisk tlačítka, ne na dotek. Testeři měli chvíli problém najít v komplexní scéně tabuli s popisem ovládání. Poté, co ji spatřili, rychle se k ní dokázali dostat pomocí teleportu a bez problémů přečíst jednotlivé popisky. Tuto tabuli použili během ostatních testovacích scénářů v případě, že si nebyli jisti jak dále pokračovat. Primární tlačítko pro zobrazení panelu nástrojů pro komentování tak tester dokázal bez problémů díky tabuli s vizualizovaným ovladačem najít a zobrazit ho. Většina testerů automaticky klikla na zobrazená tlačítka nástrojů a začala kreslit se štětcem nebo měřit pomocí metru. Mírně se tak odchýlili od původního testovacího scénáře.

Zkoumání historického modelu Tento testovací scénář odhalil nepraktický návrh uživatelského rozhraní. Testeři měli problém s otevřením detailu dané struktury, protože po stiknutí tlačítka se zároveň posouvala stránka se seznamem všech struktur dolů. Museli tak mnohdy opakovaně klikat, než aplikace zaregistrovala klik na tlačítko zobrazení detailu struktury. Tento problém by se dal vyřešit fixací vertikálního posuvníku pokud uživatel míří na tlačítko zobrazení detailu. Jednomu účastníkovi se zdál zobrazovaný text moc malý a měl problém s jeho přečtením. Zde musíme brát v potaz to, že VR headset si nasazoval bez brýlí, je tedy možné, že tento problém nastal kvůli refrakční vadě oka. Po kliknutí na tlačítko *Inspect in VR* se v pozadí aplikace stahuje příslušný model z privatního API a následně se nahrává do scény. Proces importu však není volán asynchronně a aplikace čeká na to, až doběhne. Uživateli se mezitím v brýlích zobrazí externí obrazovka operačního systému s tím, že se čeká na VR aplikaci. To na většinu testerů působilo nekomfortně a preferovali by zůstat v aplikaci a sledovat procentuální stav importu. Po náhrání a zobrazení 3D modelu s ním začali interagovat. Někteří měli potíže s nalezením bočního tlačítka na ovladači, pomocí kterého se dá model uchopit. Velmi si pochvalovali možnost objekt zvetšit a přepnout na *Wireframe*.

Závěr

V této práci jsem se věnoval vývoji aplikace ve virtuální realitě pro zkoumání a schvalování historických 3D modelů. V první části jsem analyzoval práce předchozích iterací projektu VMČK, které položily základ backendovému řešení a privátnímu API. Pro korektní návrh uživatelského rozhraní a funkcionalit aplikace jsem rozebral již navržený schvalovací proces a nově upravený interní stav objektů uložených v databázi. Částečně jsem popsal nejznámější VR headsety na trhu a analyzoval vývojové prostředí Unity Engine 3D, ve kterém jsem později aplikaci naprogramoval.

V další fázi jsem navrhl samostatnou aplikaci *Launcher* pro operační systém Windows umožňující přihlášení uživatele do VMČK systému. Pro vlastní VR aplikaci jsem navrhl dvě různé metody pohybu, reprezentaci těla pomocí avatara, diegetické uživatelské rozhraní s vícero panely, sadu nástrojů pro komentování historických modelů zahrnující štětec, metr a mikrofon, potřebnou komunikaci s privátním API pomocí jednotlivých *endpoints* a interakce s historickými předměty včetně uchopení, škálování a zobrazení síťového modelu.

V kapitole *Realizace* popisuji, jakým způsobem jsem navržené funkcionality implementoval. Výstupem je funkční aplikace, která komunikuje s privátním API projektu VMČK a umožňuje schválení nebo odmítnutí daného historického modelu. Tomu předchází případné okomentování a průzkum kvality díky navrženým interakcím.

Prototyp aplikace jsem podrobil uživatelskému testování s předem připravenými testovacími scénáři. Výsledky a zpětná vazba testerů ukázaly drobné nedostatky především v uživatelském rozhraní a metodách pohybu ve virtuálním prostoru.

Všechny stanovené cíle této práce považuji za splněné. Největším přínosem je bezpochyby existující platforma, skrze kterou mohou historici a grafici zkoumat daný 3D model a vyjádřit se k němu. Nepotřebují spouštět jiný externí program pro vizualizaci 3D modelu a mají možnost si ho prohlédnout ve virtuální realitě.

Literatura

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

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

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

[4] Sivák, D.: *Věnná města českých královen - Backend administrační části*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[5] AltexSoft: What is API: Definition, Types, Specifications, Documentation. [online], 2019, [cit. 12. března 2021]. Dostupné z: `https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/`

[6] ThehungryBrain: REST API Architectural Constraints. [online], 2020, [cit. 12. března 2021]. Dostupné z: `https://www.geeksforgeeks.org/rest-api-architectural-constraints/`

[7] SmartBear: OpenAPI Specification. [online], 2020, [cit. 12. března 2021]. Dostupné z: `https://swagger.io/specification/`

[8] SmartBear: Swagger. [online], 2021, [cit. 12. března 2021]. Dostupné z: `https://swagger.io/`

[9] Antoš, P.: *Věnná města českých královen - Webová aplikace pro schvalovací proces 3D modelů*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[10] Autodesk: About Creating 3D Wireframe Models. [online], 2015, [cit. 17. března 2021]. Dostupné z: https://knowledge.autodesk.com/support/autocad/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/AutoCAD-Core/files/GUID-84E193D7-A18D-4EE2-B978-19E4AFBCAEEC-htm.html

[11] RapidAPI: POST – What is the POST Method? [online], 2021, [cit. 17. března 2021]. Dostupné z: https://rapidapi.com/blog/api-glossary/post/

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

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

[14] Technologies, F.: Get Started Developing with Oculus. [online], 2021, [cit. 9. dubna 2021]. Dostupné z: https://developer.oculus.com/get-started/

[15] VRgineers: Enter the Enterprise VR Age. [online], 2021, [cit. 9. dubna 2021]. Dostupné z: https://vrgineers.com/

[16] Corporation, H.: About the VIVE controllers. [online], 2021, [cit. 9. dubna 2021]. Dostupné z: https://www.vive.com/eu/support/vive/category_howto/about-the-controllers.html

[17] Langley, H.: Inside-out v Outside-in: How VR tracking works, and how it's going to change. [online], 2017, [cit. 9. dubna 2021]. Dostupné z: https://www.wareable.com/vr/inside-out-vs-outside-in-vr-tracking-343

[18] Lang, B.: Oculus is Working on a 'Chaperone'-like Boundary System for Touch. [online], 2016, [cit. 9. dubna 2021]. Dostupné z: https://www.roadtovr.com/oculus-touch-chaperone-vr-boundary-wall/

[19] Technologies, U.: MonoBehaviour.Awake(). [online], 2021, [cit. 6. června 2021]. Dostupné z: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html

[20] Technologies, U.: MonoBehaviour.Start(). [online], 2021, [cit. 6. června 2021]. Dostupné z: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html

[21] Technologies, U.: MonoBehaviour.Update(). [online], 2021, [cit. 6. června 2021]. Dostupné z: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html

[22] Technologies, U.: MonoBehaviour.StartCoroutine. [online], 2021, [cit. 6. června 2021]. Dostupné z: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html

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

[24] Technologies, U.: Order of execution for event functions. [online], 2021, [cit. 6. června 2021]. Dostupné z: https://docs.unity3d.com/Manual/ExecutionOrder.html

[25] Pluralsight: Key 3D Rigging Terms to Get You Moving. [online], 2014, [cit. 25. května 2021]. Dostupné z: https://www.pluralsight.com/blog/film-games/key-rigging-terms-get-moving

[26] Boletsis, C.: Controller-based Text-input Techniques for Virtual Reality: An Empirical Comparison. [online], 2019, [cit. 5. května 2021]. Dostupné z: https://doi.org/10.20870/IJVR.2019.19.3.2917

[27] Guard, D.: Auth0 and WPF / Winforms. [online], 2021, [cit. 14. června 2021]. Dostupné z: https://auth0.com/docs/quickstart/native/wpf-winforms

[28] Museum, T. B.: A museum of the world, for the world. [online], 2021, [cit. 19. června 2021]. Dostupné z: https://sketchfab.com/britishmuseum/models

[29] Lubecki, M.: VR Square Gallery 4k. [online], 2019, [cit. 19. června 2021]. Dostupné z: https://sketchfab.com/3d-models/vr-square-gallery-4k-45f0c745a303476d964422c5a3b3865a

[30] renderpeople: Eric Rigged 001 3D Model. [online], 2020, [cit. 19. června 2021]. Dostupné z: https://free3d.com/3d-model/eric-rigged-001-771956.html

[31] Chaser324: Unity Wireframe Shaders. [online], 2017, [cit. 21. června 2021]. Dostupné z: https://github.com/Chaser324/unity-wireframe

[32] Abdelhak, S. A. M.: Black condenser microphone Free 3D model. [online], 2020, [cit. 21. června 2021]. Dostupné z: https://www.cgtrader.com/free-3d-models/electronics/audio/black-condenser-microphone

[33] Shoham, O.: Unity Google Streaming Speech-to-Text. [online], 2020, [cit. 21. června 2021]. Dostupné z: https://github.com/oshoham/UnityGoogleStreamingSpeechToText

[34] xixihaha: Unity Google Streaming Speech-to-Text. [online], 2015, [cit. 21. června 2021]. Dostupné z: https://www.cgtrader.com/free-3d-models/furniture/furniture-set/painting-brush-3fced712229f0ec5b4c8768e8eb3ee14

[35] Brooke, A.: Unity VRTK Examples. [online], 2017, [cit. 21. června 2021]. Dostupné z: https://github.com/SyntonicApps/unity-vrtk-examples/blob/master/Assets/unity-vrtk-examples/Scripts/ColorWheel.cs

Příloha A

Seznam použitých zkratek

3D Three dimensional

API Application programming interface

GUI Graphical user interface

HMD Head mounted display

HSV Hue, saturation and value

HTTP Hypertext Transfer Protocol

LCD Liquid crystal display

OLED Organic light-emitting diode

REST Representational state transfer

RPC Remote procedure Call

SDK Software development kit

SOAP Service object access protocol

UI User interface

UML Unified modelling language

URL Uniform resource locator

VMČK Věnná města českých královen

VR Virtual reality

WPF Windows presentation foundation

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `build.zip` ..... archiv se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `link.txt` ..... odkaz na vzdálený repozitář se zdrojovými kódy
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF