Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Webová platforma pro sdílení digitálních prezentací kulturních objektů

Diplomová práce

Bc. Michal Szabo

Školitel: Mgr. Pavel Mlčoch
Konzultant: Ing. Marta Vohnoutová

České Budějovice 2021

Bibliografické údaje

Szabo M., 2021: Webová platforma pro sdílení digitálních prezentací kulturních objektů [Web platform for sharing digital presentations of cultural objects Mgr. Thesis, in Czech.] – 82 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Annotation

This master's thesis deals with the design and implementation of a web platform for sharing digital cultural objects while using the augmented reality. The first part of the thesis presents a list of existing applications dealing with a similar topic. Subsequently, the intended functions of the application and scenarios of its use are defined. Then the set of technologies is introduced and the system architecture is described. At the end of the work, implementation of the application is explained.

Key words

AR, React, Node.js, Typescript, MongoDB, REST API

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

České Budějovice, 14. dubna 2021 podpis:

Poděkování

Rád bych poděkoval vedoucímu práce panu Mgr. Pavlu Mlčochovi a paní Ing. Martě Vohnoutové, za podnětné rady a za čas věnovaný této práci. Dále děkuji rodině a přátelům za podporu při studiu.

Obsah

Seznam použitých zkratek

Zkratka	Význam
AR	Augmented Reality
2D	Two Dimensional
3D	Three Dimensional
MR	Mixed Reality
AV	Augmented Virtuality
VR	Virtual Reality
URL	Uniform Resource Locator
REST	Representational State Transfer
API	Application Programming Interface
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
CSS	Cascading Style Sheets
JWT	JSON Web Token
ASCII	American Standard Cod efor Information Interexchange
MERN	MongoDB, ExpressJS, ReactJS, NodeJS
NPM	Node Package Manager
CLI	Command Line Interface

Zkratka	Význam
JS	JavaScript
MVC	Model-View-Controller
SQL	Structured Query Language
NoSQL	Not only Structured Query Language
IaaS	Infrastructure as a Service
DDoS	Distributed Denial of Service
UI	User Interface
DOM	Document Object Model
VDOM	Virtual Document Object Model
XML	Etensible Markup Language
XHTML	Extensible HyperText Markup Language
JSX	JavaScript XML
ECMA	European Computer Manufacute's Associaition
XHR	XMLHttpRequres
WebAR	Web-based Augmented Reality
WebGL	Web Graphics Library
WebRTC	Web Real-Time Communication
CORS	Cross-Origin Resource Sharing
MIME	Multipurpose Internet Mail Extensions
gLTF	Graphics Language Transmission Format

Zkratka	Význam
SPA	Single Page Application
EJS	Embedded JavaScript

1 Úvod

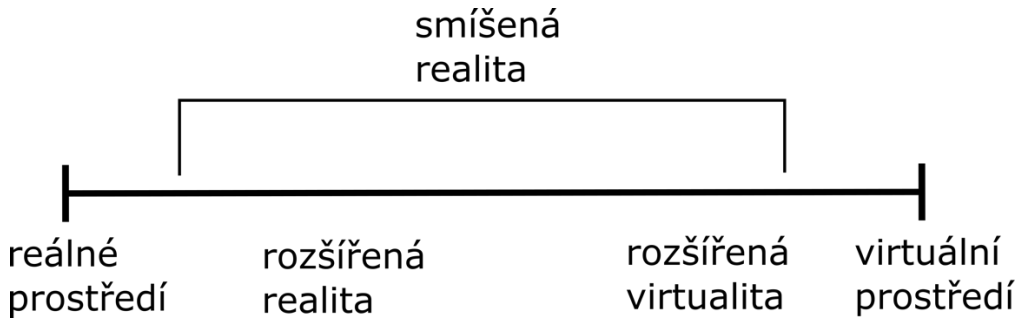
Aplikace s rozšířenou realitou (AR) jsou v dnešní době využívány mnoha způsoby a uplatňovány v různých odvětvích, jako například průmysl, armáda, výukové programy, nebo zábavní průmysl. Díky AR je možné uživateli zprostředkovat informace pro něj ne zcela obvyklou a zároveň zajímavou formou.

V naprosté většině případů se v dnešní době jedná o nativní AR aplikace, které je nutné fyzicky instalovat do zařízení, ve kterém mají být použity. Tato diplomová práce si kladla za cíl tvorbu interaktivní aplikace, jejíž výhodou by bylo, že ji není nutné instalovat, ale byla by volně dostupná pomocí webového prohlížeče. Takto vytvořená aplikace má umožnit neomezenému počtu institucí a uživatelů vytvářet projekty, které by sloužily pro sdílení digitálních prezentací kulturních objektů. Díky této aplikaci je možné prezentovat 3D modely objektů (v tomto případě kulturních artefaktů, historických předmětů či uměleckých děl), se kterými by bylo uživateli dovoleno interagovat. Tímto přístupem je docíleno vytvoření zábavné interaktivní aplikace, která díky své neotřelé formě poutá pozornost k historickým faktům a informacím o kulturních objektech a památkách.

Teoretická část práce si klade za cíl zmapovat aktuální světové trendy v oboru využití AR v oblasti kultury, historie a památek. Praktická část se poté věnuje návrhu aplikace, výběru technologií a popisuje samotnou implementaci.

2 Vymezení pojmu rozšířená realita

Na obrázku 1 je znázorněno schéma virtuálního kontinua vymezeného Milgramem [1]. Ten ve svém schématu zavádí pojem smíšená realita (mixed reality – MR). Smíšená realita označuje prostředí, které je vymezeno na jedné straně prostředím reálným (prostředí, které obsahuje pouze reálné prvky) a prostředím virtuálním (prostředí, ve kterém převládají virtuální prvky) na straně druhé. Do tohoto kontinua je možné začlenit veškeré druhy smíšených prostředí, tato prostředí se budou lišit pouze poměrem zastoupení reálných a virtuálních prvků. Na základě tohoto rozlišujeme prostředí zvané rozšířená realita (augmented reality – AR) a rozšířená virtualita (augmented virtuality – AV). Zatímco rozšířená realita se vyznačuje začleněním virtuálních prvků do zcela reálného prostředí, rozšířená virtualita využívá uměle vytvořeného prostředí, do kterého zavádí prvky reálné.



Obrázek 1 *– Schéma virutálního kontinua (upraveno z [2])*

2.1 Rozšířená realita

Rozšířená realita je technologie, ve které je reálný svět obohacen o objekty generované počítačovým programem. Umožňuje prolnutí digitálního obsahu a jeho začlenění do našeho vnímání reálného světa. Kromě 2D a 3D modelů, s kterými je AR neodmyslitelně spojována, lze do vnímání skutečného světa zakomponovat i jiné multimediální objekty, jako zvukové soubory a videa, nebo textové údaje [3]. Tyto poskytované informace může AR předávat dvěma způsoby:

- Konstruktivně – přidání objektu do reálného světa

- Destruktivně – maskování přírodního prostředí

Entity generované pomocí AR můžeme klasicky vidět díky fyzickému zobrazovacímu zařízení, typicky pomocí fotoaparátu mobilního zařízení. AR může být dále definována jako systém naplňující tři základní funkce:

- Propojuje skutečný a virtuální svět
- Umožňuje interakci v reálném čase
- Digitálními 3D objekty doplňuje reálné prostředí

AR prolíná virtuální prvky s fyzickým světem tak, aby byly vnímány jako část skutečného prostředí. Tímto způsobem mění uživateli vnímání okolního reálného světa.

Komerční využití AR bylo prvně uplatněno v zábavním průmyslu. Aplikace s AR následně překlenuly komerční odvětví a začaly se uplatňovat v armádě, ve vzdělávání, nebo v medicíně.

2.2 Rozšířená realita *vs* Virtuální realita

Rozšířená realita je často milně zaměňována za pojem virtuální realita (virtual reality – VR). Z předchozí definice rozšířené reality můžeme dojít k závěru, že AR přidává, nebo částečně překrývá jinou realitu v prostřední nebo realitě uživatele. Naproti tomu VR zcela nahrazuje skutečné prostředí uživatele realitou, která je generována počítačem.

Dvě uvedené reality od sebe lze odlišit způsobem, jakým jsou dodávány koncovému uživateli. Obě interaktivní prostředí se od sebe liší v použitých zařízeních. Virtuální realita vyžaduje náhlavní soupravy, takzvané headsety, které jsou typicky doprovázené dalšími zařízeními, které umožňují komunikovat a provádět akce přímo ve virtuálním světě. Oproti tomu rozšířené realitě stačí pouhé zobrazovací zařízení a kamera, k čemuž dostatečně poslouží například mobilní telefon.

Skutečnost, že rozšířená realita je v naprosté většině případů poháněna mobilními zařízeními je jednou z hlavních příčin růstu AR průmyslu. Ať už jde o mobily, počítače, či tablety, naprostá

většina domácností již některé z těchto zařízení vlastní a majoritní část těchto zařízení je již připravena k tomu poskytnout interaktivní zážitek.

3 Zmapování existujících aplikací

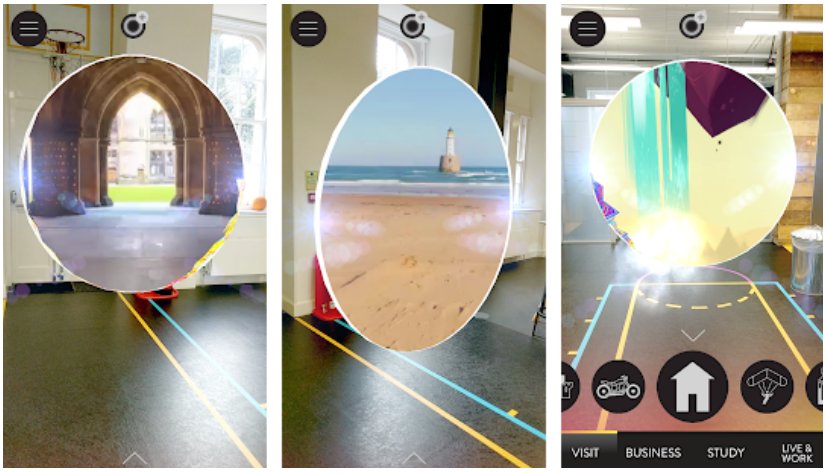
V následující sekci jsou uvedeny příklady některých populárních aplikací využívajících rozšířenou realitu pro prezentaci multimediálních objektů v oblasti kultury a historie. Sekce je dále rozdělena na popis aplikací vytvořených v České republice a aplikace vytvořené v zahraničí.

3.1 Světové AR aplikace

Podsekce obsahující ukázky některých světových aplikací využívajících rozšířenou realitu.

3.1.1 Portal AR – Step into Scotland

Tato mobilní aplikace se uživateli snaží zprostředkovat zajímavé Skotské památky z pohodlí domova. Činí tak prostřednictvím 360° obrázků a videí. Jedná se o aktuální mediální objekty pořízené v posledních letech. Aplikace funguje tak, že uživatel nejprve naskenuje volný prostor na zemi, aby se ověřilo, že má kolem sebe dostatek volného prostoru. Následně program vytvoří v prostoru portál, do kterého uživatel vstoupí a uvnitř je obklopen 360° obrázkem, nebo videem [4]. Aplikace byla vytvořena v úzké spolupráci se společností Google a týmy VisitScotland, SDI, Talent Scotland a Scotland.org. Aplikace je dostupná pro iOS a Android.

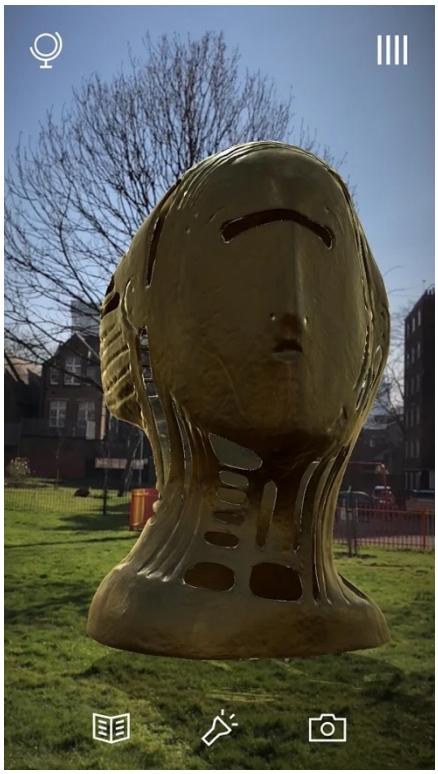


Obrázek 2 *– Ukázka aplikace Portal AR (dostupné z [5])*

3.1.2 Civilisations AR

Tato mobilní aplikace ze studia BBC přináší pestrou škálu propracovaných modelů z oblasti umění a kultury napříč celým světem. V této aplikaci lze nalézt zejména historické artefakty a rozšiřující informace o nich. V Egyptě je možné například prozkoumat starodávné sarkofágy a mumie uvnitř nich [6]. Může tedy dobře posloužit jako výuková pomůcka. Po zapnutí aplikace se zobrazí zeměkoule se zvýrazněnými miniaturami modelů v různých kontinentech a zemích. Po kliknutí na nějakou miniaturu se objekt pomocí AR přenese, kam chceme a následně s ním můžeme interagovat – například otevření sarkofágu a zkoumání mumie, nebo interakce v podobě vybarvování / přebarvování zobrazovaných modelů. Aplikace je dostupná pro iOS i Android.

Obrázek 3 *– Výběr objektu v aplikace Civilisations AR (dostupné z [7])*



Obrázek 4 *– Ukázka 3D objektu v aplikaci Civilisations AR (dostupné z [7])*

3.1.3 PIVOTtheWORLD

PIVOT je zkratka pro „the Point of Interest Visual Optimization Tool“, ve volném překladu tedy „Nástroj pro vizuální optimalizaci zájmu“. Jedná se o mobilní aplikaci, která odhaluje uživatelům, jak vypadala místa v minulosti a poskytuje k nim dodatečné informace. Zobrazuje historická média z různých lokací – od všedních, až po turistické atrakce. Tato média zde představují fotografie, videa, audio, nebo text. V aplikaci funguje navigace k historickým bodům, u nichž se zpřístupní patřičné médium – fotografie, video, audio apod. Multimediální objekty se poté mohou zobrazit dvěma způsoby. Buď klasicky, bez využití rozšířené reality, nebo jako překrytí pomocí AR, kde dosadíme daný objekt do místa, ve kterém se nacházíme [8]. K místu lze přiřadit více multimediálních objektů. Ověřené instituce nahrávají materiály vzdáleně a pomocí mapy vyberou místo, na kterém se mají zobrazovat. Koncový uživatelé poté mohou tyto materiály zobrazovat pouze v lokalitách pro ně určených.

Obrázek 5 *– Ukázka aplikace PIVOTtheWORLD (dostupné z [9])*

3.1.4 Chevré 3D

Chevré 3D je mobilní aplikace od společnosti Artefacto. Jejím hlavním cílem je zhodnotit historické dědictví francouzského středověkého kopce, který se nachází nedaleko malého města La Bouxiere v departementu (administrativní a samosprávní území) Ille-et-Vilaine. Díky aplikaci je možné obnovit historickou věž, stojící na onom kopci, do její původní formy, a to právě díky rozšířené realitě. Uživatelé mohou prozkoumat více již neexistujících památek v okolí. Například kapli, mlýn, nebo románský most. Na mapě je dále plno bodů, které se dají rozkliknout a nabízejí interaktivní videa, rozšiřující informace, nebo různé hry. V průběhu cesty lze dále zobrazovat informace o fauně a flóře. Pro zvýraznění starověké doby navíc vývojáři do aplikace přidali model draka vznášejícího se vedle kaple [10].

Obrázek 6 *– Historická věž v aplikaci Chevré 3D (dostupné z [10])*

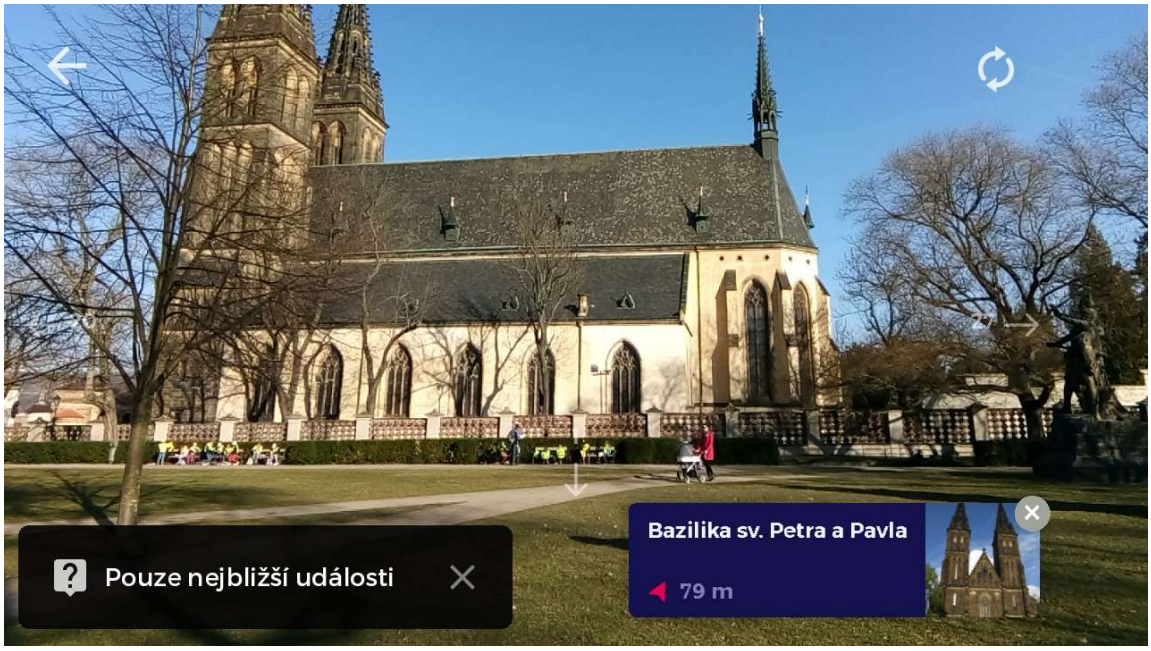
Obrázek 7 *– 3D model vznášejícího se draka v aplikaci Chevré 3D (dostupné z [10])*

3.2 České AR aplikace

Podsekce obsahující ukázky českých aplikací využívajících rozšířenou realitu.

3.2.1 Arthur

Aplikace Arthur funguje jako interaktivní asistent, který pomocí AR nabízí přehled o památkách a zajímavých kulturních akcích v okolí. Pro zjištění toho, co se v okolí děje Arthur v aplikaci vyznačí, kterým směrem se vydat a zobrazí náhled dané památky, nebo akce. Tyto náhledy po rozkliknutí poskytnou další dodatečné informace, které obsahují například historii daného místa, nebo adresu. K místům vzdáleným méně než 500 metrů lze dojít pomocí navigace přímo v aplikaci. Arthur navíc poskytuje funkci „Co to je?“, která umožní vyhledat informace o příslušném objektu poté, co je na něj namířeno fotoaparátem mobilu. Aplikace v současné době poskytuje informace o nejvýznamnějších památkách České republiky. Nejpropracovanější částí jsou ale města, se kterými společnost spolupracuje a pro které zajistí pasportizaci (zaevidování hmotného i nehmotného majetku do databáze) zajímavostí z katastru města. Město si poté může díky rozhraní samo spravovat seznamy akcí a určovat jejich priority [11].



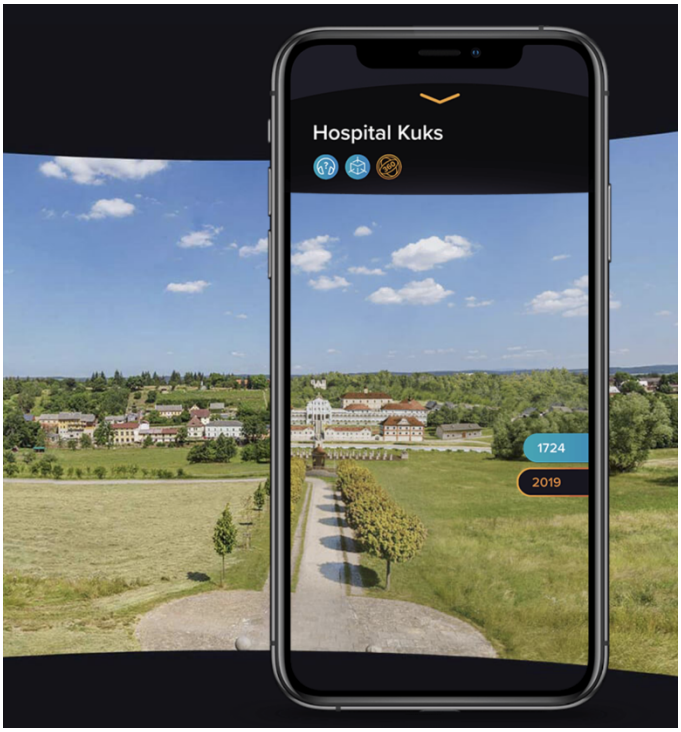
Obrázek 8 *– Ukázka aplikace Arthur (dostupné z [12])*

Obrázek 9 *– Přehled památek v okolí v aplikaci Arthur (dostupné z [12])*

3.2.2 Visit.More

Aplikace Visit.more byla vytvořena pro areál historického hospitálu Kuks v Královéhradeckém kraji. Uživatelům dovoluje prohlížení pamětihodností v podobě, jakou měly v dobách největší slávy. Stačí namířit fotoaparát do areálu a aplikace nám nabídne pohled na krajinu, jak asi vypadala v roce 1724. Model bylo možné rekonstruovat díky zachovaným historickým pramenům, zejména pak obrazy Kuksu. Stávající podoba areálu byla 3D letecky naskenovaná pomocí dronů s přesností až na jeden milimetr, což umožnilo detailní vymodelování 3D objektů, například soch, které si poté můžeme zobrazit v krajině, otáčet s nimi a prohlédnout si je ze všech

stran. Aplikace dále nabízí dodatečné informace o areálu, dobové obrazy, fotografie a audio průvodce [13].



Obrázek 10 *– Areál Kuks v aplikaci Visit.More (dostupné z [14])*

3.3 Shrnutí

Z příkladů uvedených výše vyplývá, že dosavadní aplikace se zaměřují většinou pouze jedním směrem:

- prezentují propracované 3D modely památek a objektů
- nabízejí navigaci v prostoru a dodatečné informace v podobě textu, fotografie, nebo videa

Tyto dvě funkcionality ale většinou nejsou spojeny do jednoho projektu. Pokud ovšem projekt spojuje obojí, jedná se pouze o úzce zaměřenou oblast, nebo konkrétní historickou památku, pro kterou byla aplikace vytvářena na míru. Navíc jsou programy z drtivé většiny dostupné pouze pro mobilní zařízení a je nutná jejich instalace a nenabízejí interakci s jakoukoliv částí programu pomocí jiných zařízení.

4 Scénáře aplikace pro zvýšení zájmu o kulturní obsah

Největším problémem současných aplikací s rozšířenou realitou zaměřených na kulturní obsah je fakt, že většinou neposkytují možnost využívat jakoukoliv část programu bez nutnosti využití AR a kvůli tomu jsou použitelné zejména na mobilních zařízeních a uživatel u stolního počítače se nedostane ani k textovým informacím. V této sekci budou popsány různé scénáře využití zamýšlené aplikace s vědomím, že je nutné zprostředkovat část informací i bez nutnosti využití AR a umožnit tak přístup k informacím uživatelům, kteří momentálně nedisponují zařízením s kamerou, nebo si chtějí zjistit informace o prvcích, ke kterým není využití AR nezbytně nutné. Následující scénáře budou implementovány dále v praktické části práce.

4.1 Scénáře uživatele

Následující části popisují dva rozdílné přístupy uživatelů z pohledu použitého zařízení.

4.1.1 Uživatel u stolního počítače

Takového uživatele si můžeme představit jako někoho kdo sedí za počítačem a má omezenou možnost využít rozšířenou realitu aplikace. Jelikož aplikace nebude čistě mobilní, ale bude se jednat o webovou aplikaci přístupnou z běžného webového prohlížeče, bude mít i tak uživatel možnost dostat se k velké části obsahu. Po výběru historického místa bude uživateli nabídnut výčet památek a objektů uložených v systému. Uživatel bude mít konkrétně možnost čtení textu, prohlížení fotografií, přehrávání audia a videa. Dále bude mít možnost plnit různé kvízy. A v neposlední řadě možnost zobrazit si 3D modely s využitím, nebo bez využití rozšířené reality. K využití rozšířené reality bude nutné, aby zařízení disponovalo kamerou a uživatel k této kameře umožnil webové aplikaci přístup.

4.1.2 Uživatel s mobilním zařízením

Zde je možné si představit uživatele, který do aplikace vstoupil z webového prohlížeče na mobilním zařízení s fotoaparátem. Bude mít stejné možnosti jako uživatel v předešlém scénáři. Navíc bude mít možnost vstoupit do sekce „Navigace“. Tato sekce budou sloužit k navigaci v reálném světě k prvkům, které disponují informací o jejich umístěný (souřadnicemi). Navigace

bude provedena pomocí mapy, a navíc pomocí AR ukazujícím směr k požadovanému místu. Nebude se ovšem jednat o navigaci ve fyzických objektech – například muzeum, kde by bylo nutné navigovat s přesností na centimetry. Zde by bylo s webovou aplikací velmi obtížné dosáhnout spolehlivé a přesné navigace s rozumnou přesností.

4.2 Funkcionalita aplikace

Definice funkcí aplikace vychází ze zadání práce a z konzultací s vedoucím práce Mgr. Pavlem Mlčochem.

Aplikace si klade za cíl poskytnout neomezenému počtu institucí a uživatelů nástroj umožňující sdílení virtuálních prezentací historických a kulturních památek. Tyto sbírky by mohly obsahovat několik druhů multimediálních objektů (obrázky, videa, zvukové soubory, 3D modely), nebo kvízů. Možná interakce s 3D modely v prostředí rozšířené reality a plnění kvízů má za účel předávat informace neotřelou a hravou formou. Uživatelé by k této aplikaci přistupovali výhradně pomocí webového prohlížeče, bez nutnosti cokoliv fyzicky instalovat do svých zařízení.

Na základě této skutečnosti byl vytvořen model popisující možnosti jednotlivých rolí uživatelů. Jedná se o tři role. Nepřihlášený uživatel, přihlášený uživatel, administrátor aplikace. Všechny uvedené role mají množinu společných možností, a to:

- Zobrazení přehledu, na kterém se nachází všechny schválené projekty
- Zobrazení detailu konkrétního projektu a s tím spojený přehled jednolitých mediálních objektů k projektu přiřazeným
- Zobrazení detailu mediálního objektu, obsahující jeho popis a prostředí pro zobrazení, nebo přehrání
- Vyplnění kvízů, které mohou být časově omezené
- V případě 3D modelů možnost zobrazit model pomocí rozšířené reality
- Možnost přechodu do sekce s navigací, která bude sloužit jako navigace ve venkovních prostorech a bude obohacena o možnost navigace pomocí rozšířené reality

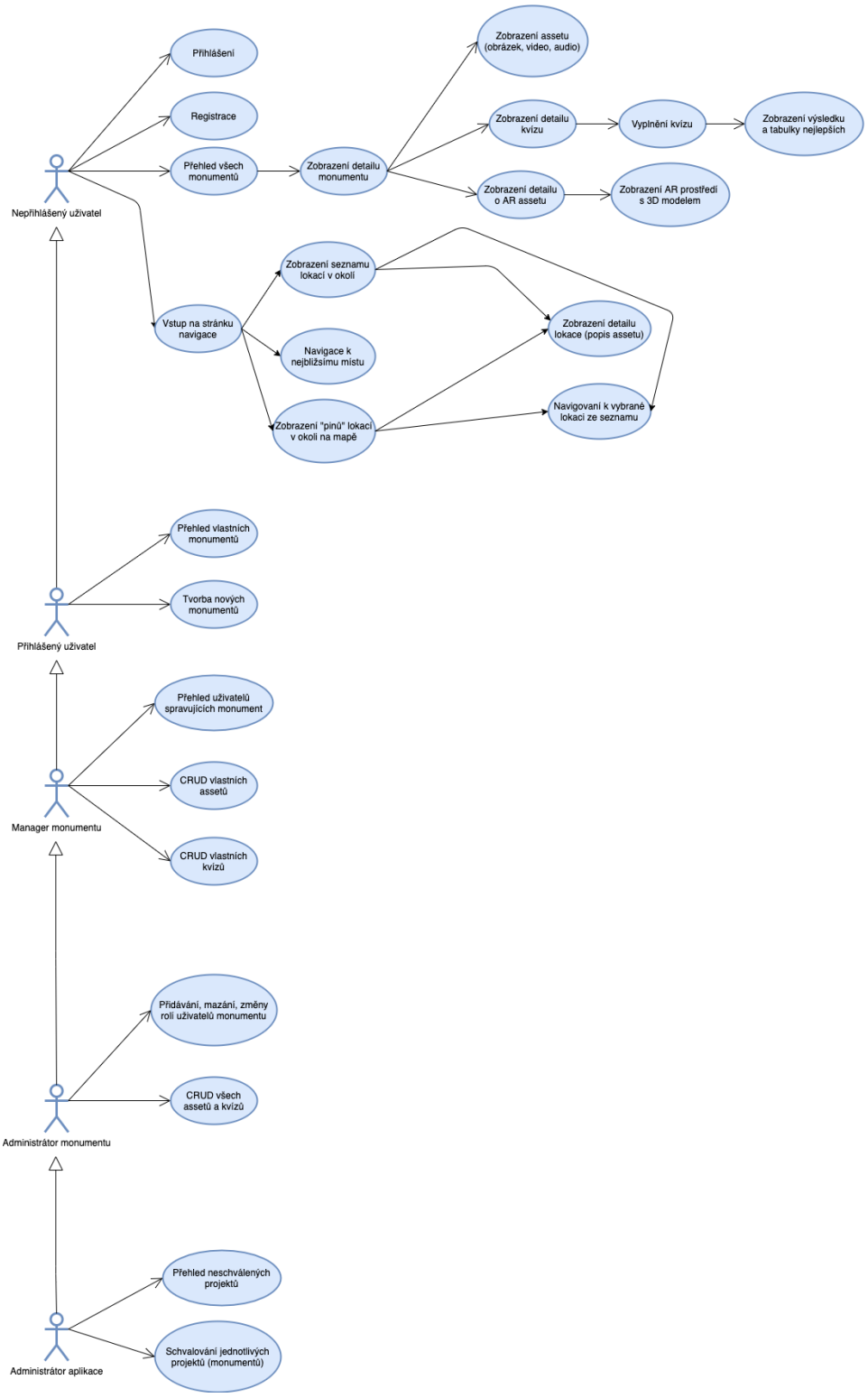
Role přihlášeného uživatele je rozšířena o tyto funkcionality:

- Možnost vytváření, editace a mazání projektů. Jednotlivé projekty lze zneviditelnit, takže nejsou zobrazovány na přehledu všech projektů a jsou dostupné pouze pomocí URL
- Přiřazování nových multimediálních objektů k vytvořeným projektům, s tím související editace a mazání. Tyto objekty lze zneviditelnit stejně jako projekty
- Možnost k jednotlivým objektům přidávat koordináty, které poté slouží k navigaci v části aplikace pro to určené
- Tvorba kvízů
- Přiřazování dalších členů, kteří by mohli spravovat obsah daného projektu. Těmto členům je dále možno přiřadit dvě role v rámci projektu. Administrátor, který má práva k tvorbě, editace a mazání všech objektů. A manager, který má stejná práva, ale týkající se pouze vlastních objektů

Poslední role administrátora celé aplikace má navíc jednu funkci:

- Schvalování a zamítání projektů. Po tvorbě nového projektu je potřeba jeho schválení, aby se k němu mohli uživatelé dostat. Pokud administrátor usoudí, že některý projekt není vhodný, může jeho schválení zamítnout a projekt nebude dostupný na přehledu projektů, ani pomocí URL adresy

Na následujícím obrázku (Obr. 11) je zobrazen use-case diagram vytvořený v rámci návrhu aplikace.



Obrázek 11 – *Use-case diagram*

5 Návrh projektu

V této části práce bude popsána architektura aplikace. Dále popis databázové struktury, návrh REST API (aplikační rozhraní) serveru, popis autentizace a ukázky předpřipravených grafických návrhů webové aplikace.

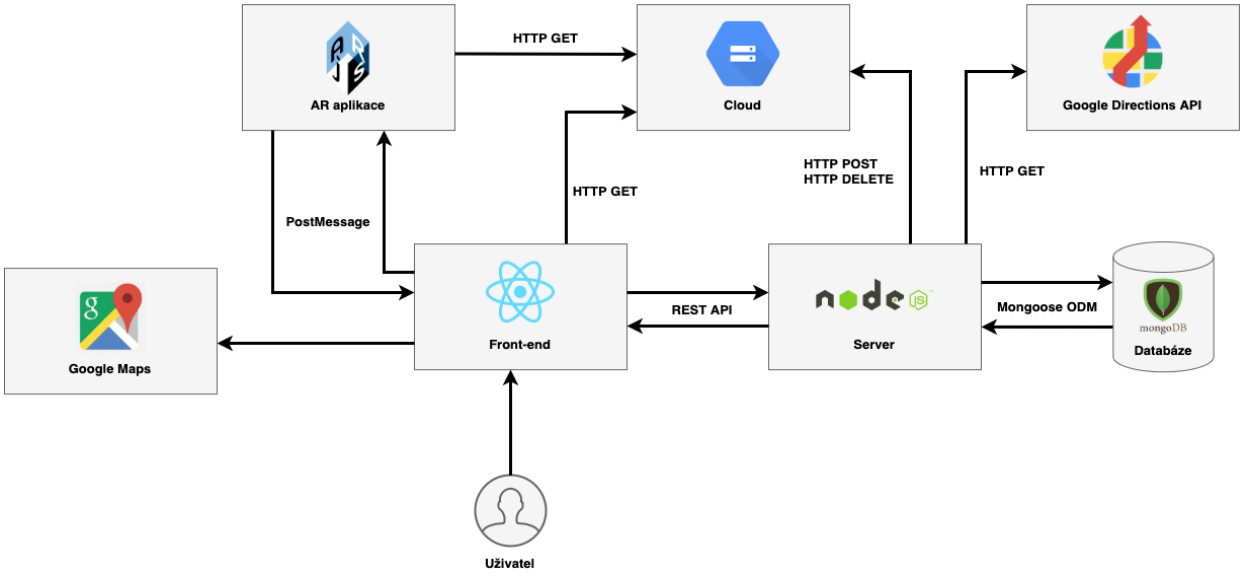
5.1 Architektura

Aplikace se skládá ze dvou hlavních stavebních kamenů – server (backend) a webová aplikace (frontend). Na tyto části poté navazuje několik dalších bloků. Komunikace mezi backend a frontend částí aplikace probíhá pomocí REST API (standard komunikace pomocí aplikačního rozhraní). Pro přenos informací byl použit formát JSON, jehož hlavní výhoda spočívá v přehlednosti dat a jejich struktuře. Tento formát je dnes již ve většině aplikací považován za standard.

Serverová část následně komunikuje se třemi dalšími bloky. S databází, do které zapisuje a získává uložené informace. Dále se vzdáleným úložištěm (cloud), do kterého ukládá multimediální objekty. A také s webovou službou poskytující informace o umístění a kalkulaci různých druhů dopravy. Tato poslední část je v aplikaci využita v části „Navigace“, která je popsána v sekci praktického využití.

Frontend je ten díl aplikace, ke které přistupuje uživatel pomocí webového prohlížeče. Kromě serveru komunikuje tato část s další frontend aplikací, která je do té hlavní vkládána pomocí HTML (HyperText Markup Language) rámce (iframe). Tato vedlejší webová aplikace má za úkol poskytování prostředí AR. Díky využití iframu uživatel nepozná, že komunikuje s dvěma aplikacemi najednou, ale jeví se mu jako jednolité. Komunikace mezi těmito částmi probíhá pomocí JavaScriptové metody *PostMessage*, která nabízí zabezpečené odesílání a přijímání zpráv napříč různými aplikacemi a doménami. Hlavní frontend část dále komunikuje s cloudem, ze kterého využívá URL adresy multimediálních objektů a ty poté vykresluje v aplikaci. Webová část také komunikuje s nástrojem Google Maps, který využívá v sekci „Navigace“.

Na obrázku 12 naleznete strukturu znázorňující výše popsanou architekturu aplikace. Použité technologie budou vysvětleny v sekci „Implementace“.

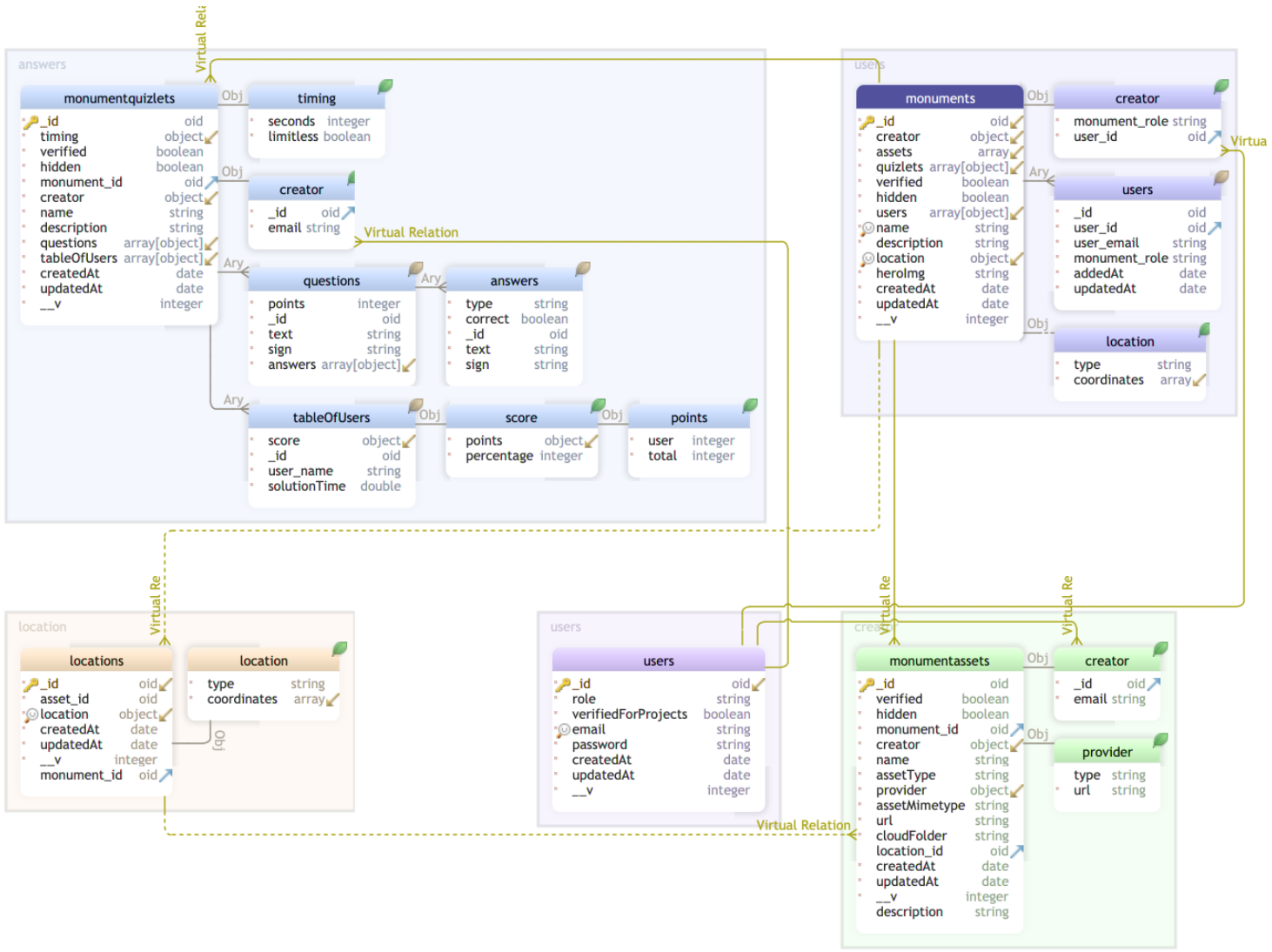


Obrázek 12 *– Architektura aplikace*

5.2 Databázová struktura

Databáze bude uchovávat data o vytvořených historických projektech (*Monument*), které obsahují různé multimediální objekty (*Asset*). K těmto objektům lze přiřadit informace o poloze (*Location*) využívaných v sekci „Navigace“. Projekty mohou také obsahovat kvízy (*Quizlet*). Všechny popsané prvky vytváří registrovaní uživatelé (*User*).

Z výše uvedeného vyplývá, že se databáze skládá z pěti základních dokumentů. Některé dokumenty obsahují navíc dokumenty vnořené. Tento způsob struktury umožňuje využitá NoSQL databáze MongoDB, která bude popsána v sekci „Implementace“. Schéma databáze je vyobrazeno na následujícím obrázku (Obr. 13).



Obrázek 13 – *Databázové schéma*

5.3 Návrh API

Aplikační rozhraní slouží ke komunikaci backend a frontend části aplikace. Jak již bylo zmíněno v předešlé části práce, serverová část bude fungovat jako REST API, což umožní webové aplikaci přístup a manipulaci se zdroji serveru. Rozhraní bude sloužit pro:

- práci s historickými projekty – *Monuments*
- práci s multimediálními objekty – *Assets*
- práci s kvízy – *Quizlets*
- získávání polohy objektů – *Locations*
- získávání informací o uživatelích – *Users*
- autentizaci – *Auth*

Následující tabulky shrnují nabízené operace RESP API. Ne všechny metody jsou dostupné všem uživatelům. Ty jsou rozděleny do tří skupin dle role uživatele. Volně dostupné pro všechny (Tabulka 1). Operace vyžadující přihlášení v aplikaci (Tabulka 2) a operace dostupné pouze administrátorovi aplikace (Tabulka 3).

Tabulka 1 *– Volně dostupné operace*

URL	HTTP metoda	Obsah požadavku
/api/monument	GET	
/api/monument/:monument_id	GET	monument_id
/api/monument-asset/monument/:monument_id	GET	monument_id
/api/monument-asset/:asset_id	GET	asset_id
/api/quizlet/monument/:monument_id	GET	monument_id
/api/quizlet/:id	GET	id
/api/quizlet/:id/solution	POST	id, user_name, questions, solutionTime
/api/quizlet/:id/table	GET	id
/api/location/places	POST	latitude, longitude
/api/location/polyline	POST	latStart, lngStart, latFinish, lngFinish
/api/location/place/:location_id	GET	location_id
/api/user	POST	email, password
/api/auth	POST	email, password

Tabulka 2 *– Operace vyžadující přihlášení uživatele*

URL	**HTTP metoda**	**Obsah požadavku**
/api/monument/own	GET	
/api/monument	POST	file, name, description, hidden
/api/monument/:monument_id	PATCH	monument_id, description, hidden
/api/monument/:monument_id	DELETE	monument_id
/api/monument/:monument_id/management	POST	monument_id, user_email, role
/api/monument/:monument_id/management	PATCH	monument_id, user_id, role
/api/monument/:monument_id/management	DELETE	monument_id, user_id

URL	HTTP metoda	Obsah požadavku
/api/monument-asset/monument/:monument_id	POST	monument_id, file, name, description, assetType, hidden
/api/monument-asset/:asset_id/monument/:monument_id	DELETE	asset_id, monument_id
/api/monument-asset/:asset_id/monument/:monument_id	PATCH	asset_id, monument_id, name, description, hidden
/api/quizlet/monument/:monument_id	POST	monument_id, name, description, hidden, timing, questions

URL	**HTTP metoda**	**Obsah požadavku**
/api/quizlet/:id/monument/:monument_id	PATCH	id, monument_id, name, description, hidden
/api/quizlet/:id/monument/:monument_id	DELETE	id, monument_id
/api/auth	GET	

Tabulka 3 *– Operace dostupné pouze administrátorovi aplikace*

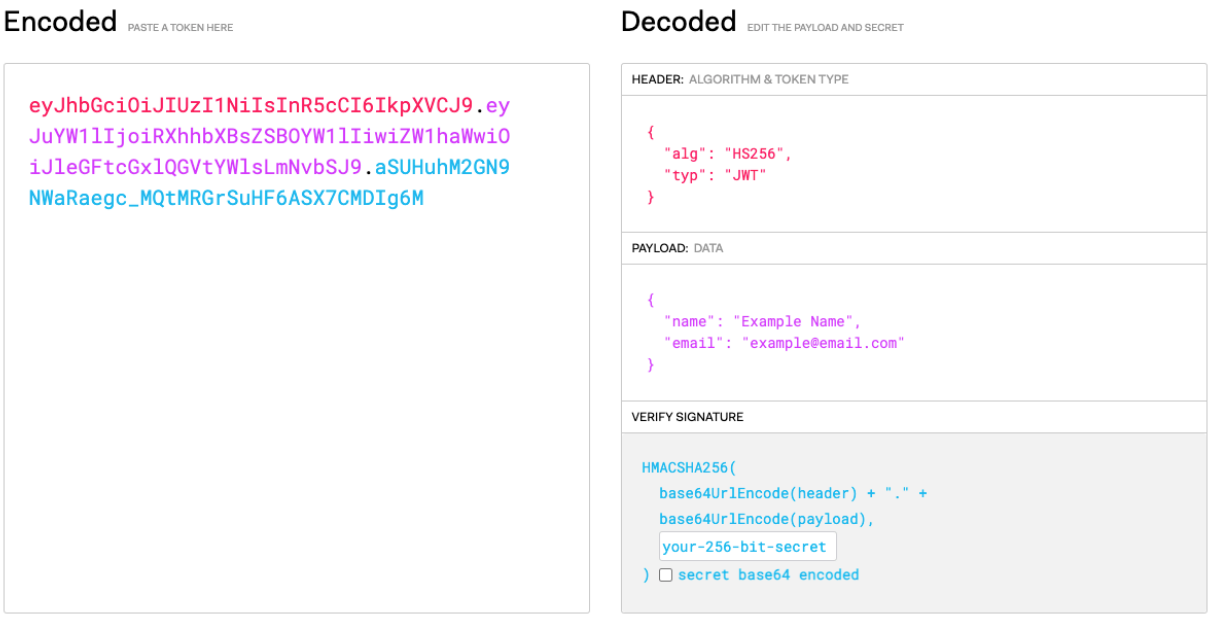
URL	**HTTP metoda**	**Obsah požadavku**
/api/monument/hidden	GET	
/api/monument/verification	GET	
/api/monument/:monument_id/verification	PATCH	monument_id, verified

5.4 Autentizace

Pro využití zabezpečených operací nabízených REST API je zapotřebí prokázat totožnost uživatele, který zaslal požadavek na server. V této práci bylo využito autentizace pomocí JWT (JSON Web Token). Jedná se o standard, který definuje způsob bezpečného přenosu informací pomocí tokenů v datovém formátu JSON. JWT se skládá ze tří částí:

- Hlavička – obsahuje informaci o typu tokenu (*typ*) a druhu použitého šifrovacího algoritmu (*alg*) jako například HMAC, SHA256, nebo RSA.
- Data – druhá část tokenu obsahuje zašifrované informace o uživateli jako například jeho jméno, nebo email
- Podpis – podpisová část slouží k validaci celého tokenu. Vytváří se ze zašifrované hlavičky a těla tokenu. Podpis slouží k ověření, že nebyl token změněn v průběhu požadavku

Výstupem JWT jsou tři textové řetězce (Obr. 14) oddělené tečkami kódované v Base64-URL (schéma kódování prezentující binární data ve formátu textového řetězce ASCII). Tyto tři textové řetězce znázorňují výše popsanou hlavičku, data a podpis tokenu [15]. Řetězí se do jednoho uceleného řetězce a lze jej tak snadno předávat pomocí HTTP dotazů.

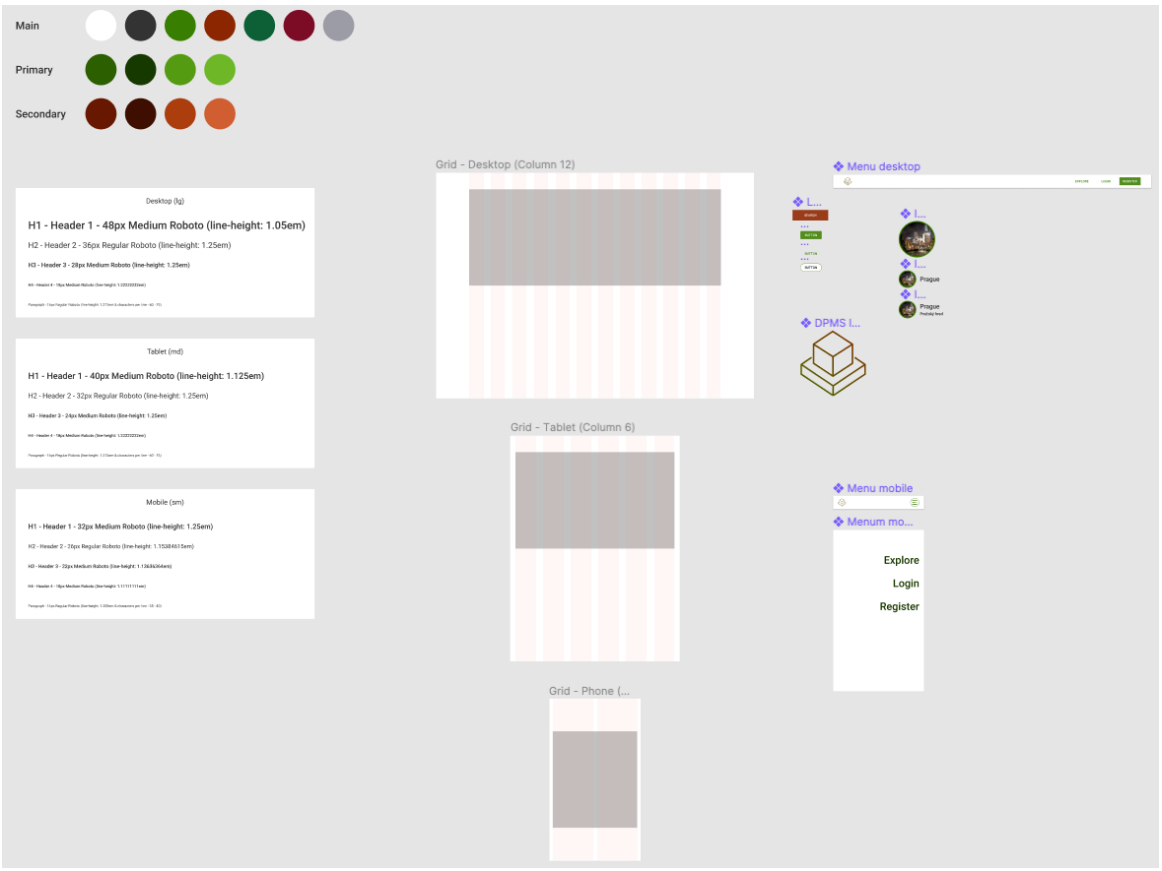


Obrázek 14 *– Ukázka šifrování JWT (dostupné z [16])*

5.5 Návrh grafického rozhraní

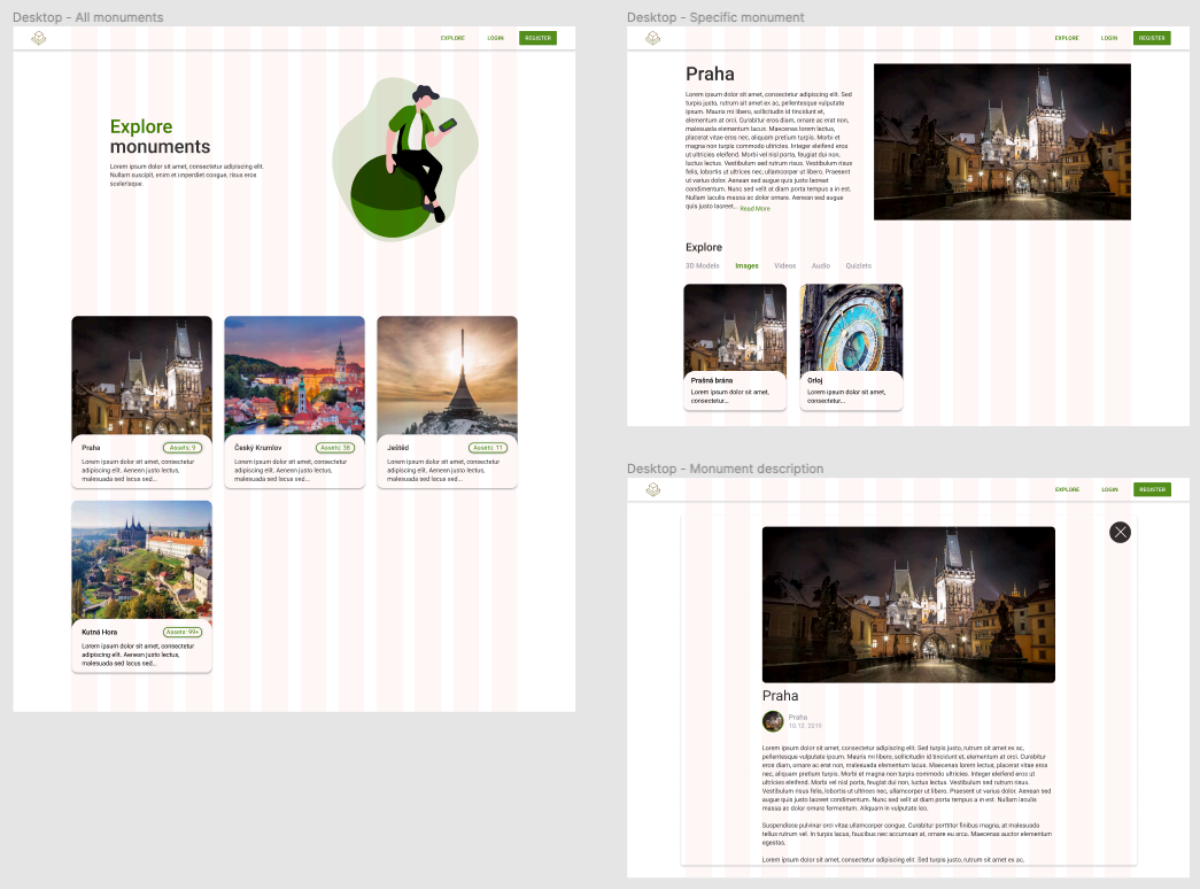
Pro vývoj uceleného vzhledu webové aplikace bylo dále přistoupeno ke kroku návrhu grafického rozhraní. K tomuto účelu byl použit software pro tvorbu vektorové grafiky Figma. Tento nástroj nabízí tvorbu grafických návrhů ve webovém prohlížeči, kde se jednotlivé návrhy zároveň automaticky zálohují.

Nejprve bylo nutné sestavit barevnou paletu a sjednotit typografii celého webu. Dále byly vytvořeny společné opakující se prvky použité napříč více návrhy, jako šablona rozložení obrazovky klasického počítače, nebo mobilního zařízení. Dále například komponenta menu a logo aplikace. Náhled těchto společných prvků je vyobrazen na následujícím obrázku (Obr. 15).

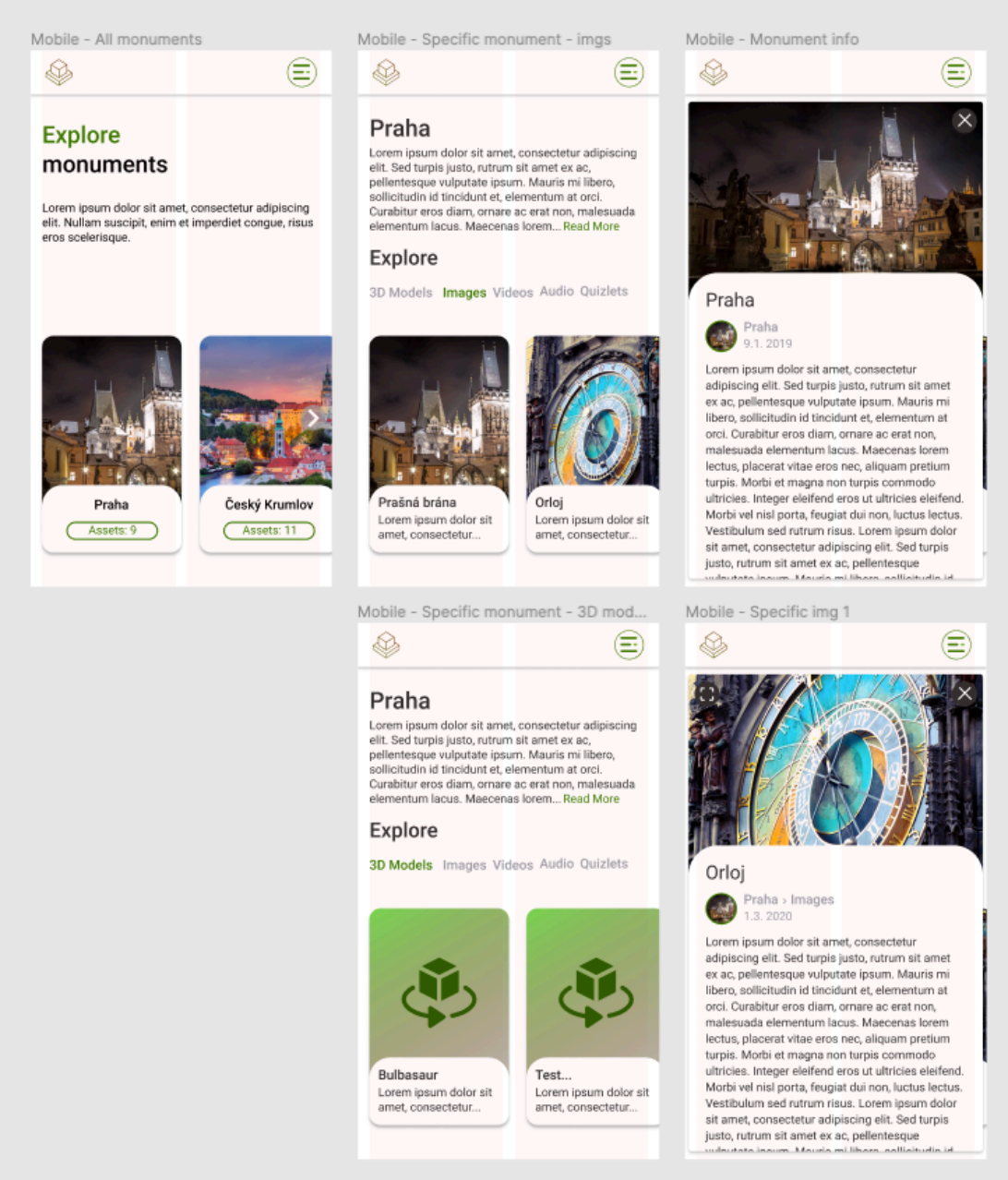


Obrázek 15 – *Komponenty v editoru Figma*

Po vytvoření společných komponent byla vytvořena grafika pro desktopovou a mobilní verzi webu. Ukázky navrhnutých obrazovek můžete vidět na následujících obrázcích (Obr. 16 a Obr. 17).



Obrázek 16 *– Ukázka desktopové grafiky v editoru Figma*



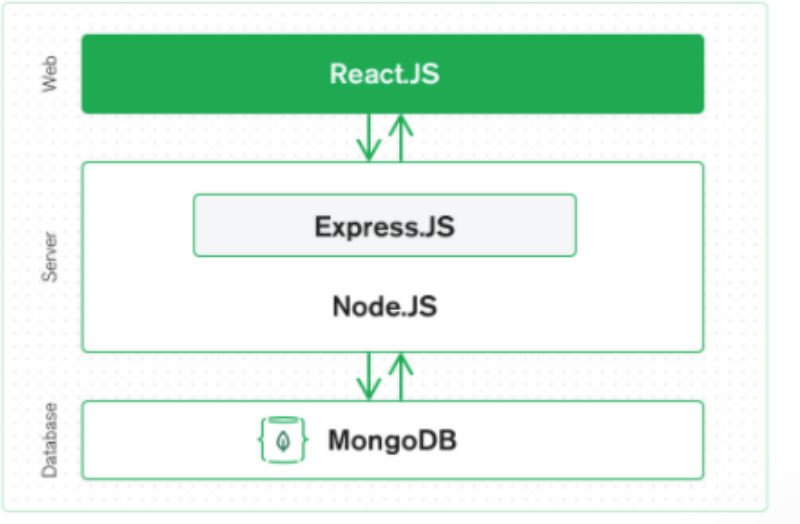
Obrázek 17 *– Ukázka mobilní grafiky v editoru Figma*

6 Implementace

Tato kapitola se zabývá tvorbou jednotlivých částí projektu. V první řadě bylo nutné vybrat sadu nástrojů a technologií, které budou při vývoji použity. Dále je již v práci popsána samotná implementace aplikace.

6.1 Použité technologie

Vzhledem ke zkušenostem autora s programovacím jazykem JavaScript byla zvolena sada technologií označovaná jako MERN. Tato zkratka skrývá názvy hlavních nástrojů použitých při vývoji. Jedná se o MongoDB, Express.js, React.js, Node.js. MERN architektura tak umožňuje kompletní konstrukci full-stack (web, server, databáze) aplikací s využitím pouze jazyka JavaScript a formátu JSON [17].



Obrázek 18 *– Komunikace mezi vrstvami v MERN architektuře (dostupné z [17])*

Jednotlivé knihovny byly vybrány na základě zkušeností autora a dále dle požadavků na specifické funkcionality aplikace. Kvůli velkému množství použitých knihoven budou níže popsány jen ty nejstěžejnější.

6.1.1 Společné technologie

Díky uplatnění pouze jednoho programovacího jazyka bylo možné využít několik stejných technologií jak na webu, tak i na serveru. Ty nejdůležitější jsou popsané v následujících podsekcích.

6.1.1.1 NPM

Node Package Manager (NPM) je registr sloužící pro sdílení JavaScriptového kódu a s více než milionem různých balíčků kódu se jedná o největší registr softwaru na světě. Z tohoto registru je možné stahovat a využívat volně dostupné balíčky určené pro řešení specifických druhů problémů o různých velikostí [18]. Může se jednat o malou část kódu řešící nějaký častý jednoduchý problém, nebo o složitou testovací knihovnu. Základní myšlenkou NPM registru je poskytnutí menších stavebních kamenů (balíčků), které se snaží dobře řešit jeden problém. Pomocí této metody je možné skládat nespočet balíčků do jednoho projektu a vytvářet tak velké aplikace.

```
{
    "name": "example",
    "version": "1.0.0",
    "author": "Michal Szabo",
    "description": "package.json example",
    "main": "index.js",
    "license": "ISC",
    "engines": {
        "node": "^15.9.0",
        "npm": "^7.5.3"
    },
    "scripts": {
        "start": "npm start",
        "build": "npm build"
    },
    "dependencies": {
        "typescript": "^4.2.3"
    }
}
```

Listing 1 *– Ukázka souboru package.json*

NPM také slouží jako správce závislostí (package manager) pro projekty. Jeho součástí je CLI (Command Line Client), jehož pomocí může uživatel stahovat balíčky z NPM registru pomocí příkazové řádky, nebo spouštět nadefinované skripty [18]. Knihovny se ukládají do složky *node_modules* a informace o použitých balíčcích, jejich verzích a dalších závislostech jsou obsaženy v souboru *package.json*. Při tvorbě projektu nebyla tato funkce využita, ale pro správu balíčků byl využit správce Yarn popsaný níže.

6.1.1.2 Yarn

Yarn je balíčkovací nástroj (package manager) umožňující sdílet a používat JavaScriptové knihovny z NPM registru. Nejedná se přímo o náhradu za balíčkovací nástroj NPM, protože ke svému fungování využívá několik NPM knihoven a v případě stahování nových balíčků využívá stejný registr knihoven. Rozdíl oproti NPM je ovšem ve způsobu a rychlosti instalace jednotlivých balíčků. NPM provádí instalaci balíčků sekvenčně. To znamená, že stahování a instalace dalšího požadovaného balíčku nemůže začít, dokud není dokončena instalace toho předešlého. Oproti tomu Yarn umožňuje instalaci paralelní a značně tak urychluje celý proces. K udržování informací o instalovaných knihovnách využívá soubor *package.json* jako NPM [19]. Většina instalovaných knihoven má další závislosti – další knihovny, které ke svému chodu potřebují. Informace o nich si Yarn udržuje v souboru *yarn.lock*. Nástroj dále nabízí několik užitečných příkazů, které NPM neposkytuje. Například k zjištění neznámé knihovny v *node_modules* lze použít příkaz *yarn why***,** který vypíše, které balíčky jsou na dané knihovně závislé.

6.1.1.3 TypeScript

TypeScript je open-source programovací jazyk vyvinutý a spravovaný společností Microsoft. Slouží jako nadstavba nad jazykem JavaScript a rozšiřuje ho o funkcionality, které jsou známé z objektově orientovaného programování. Především pak o možnost statického typování. Jelikož se jedná o nadmnožinu jazyka JavaScript, je každý JavaScriptový soubor validní TypeScriptový soubor. K fungování TypeScriptového kódu v produkčním prostředí (například ve webovém

prohlížeči) je nutné jeho přeložení do jazyka JavaScript [20]. K tomuto účelu byl využit nástroj Babel, jehož funkcionalita je vysvětlena ve vlastní podsekci.

Na ukázce kódu níže (Listing 2) je znázorněno, jakým způsobem umožňuje TypeScript přiřadit zamýšlený datový typ proměnné *num* a návratový typ funkci *getText*.

```
const num: number = 2;

const getText = (): string => 'some string text';
```

Listing 2 *– Anotace typů v jazyce TypeScript*

6.1.1.4 ESLint

ESLint je nástroj pro analýzu statického kódu a identifikaci problematických vzorů v JavaScriptovém kódu. Pravidla, která se mají kontrolovat jsou konfigurovatelná vývojářem aplikace a ESLint poté hlídá jejich dodržování. V souboru *.eslintrc.js* lze nastavit, jaké soubory a pravidla se musí dodržovat. Dodržování těchto striktních pravidel umožňuje vývojáři psát čistší a udržitelnější kód [21].

6.1.1.5 Prettier

Prettier je knihovna vynucující konzistentní styl kódu v celém projektu na základě konfigurace jednotlivých typů souborů. Zlepšuje udržitelnost a čitelnost kódu napříč celou aplikací. Nástroj také umožňuje nastavení automatického formátování různých druhů souborů [22]. Nastavení se provádí v souboru *.prettierrc*.

```
{
    "singleQuote": true,
    "printWidth": 80,
    "editor.formatOnSave": true,
    "tabWidth": 4,
    "useTabs": false,
    "trailingComma": "none",
    "semi": true
}
```

Listing 3 *– Konfigurační soubor .prettierrc*

6.1.1.6 Webpack

Webpack je open-source nástroj pro zpracovávání souborů. Nabízí možnost balíčkovače (module bundler) a spouštěče úloh (task runner). Jeho primárním účelem je skládání JavaScriptových modulů do ucelených balíčků pro použití v produkčním prostředí. Díky rozšiřujícím knihovnám je také možné transformovat téměř jakýkoliv druh souboru [23]. Například HTML, CSS, nebo různé druhy assetů jako jsou obrázky, fonty, ikony a podobně. Webpack se můžou spouštět z příkazové řádky, ale pro pohodlnější použití je vhodné vytvořit alias v souboru *package.json* a spouštět ho pomocí NPM nebo Yarnu.

Webpack dále nabízí *webpack-dev-server*, který značně usnadňuje vývoj aplikací a nabízí kontrolu změny kódu a automatického buildu (sestavení aktualizovaného kódu) za běhu aplikace. Tato funkce je také známá jako „*live reloading*“.

Požadovaná konfigurace webpacku se provádí buďto rovnou v příkazové řádce, nebo je možné konfiguraci vytvořit v souboru a na toto nastavení poté pouze odkazovat. Projekt může obsahovat neomezeně konfiguračních webpack souborů pro různé druhy potřeb vývojařů. Typicky se jedná o vývojové prostředí, tedy vývojový server s automatickým buildem a dále o sestavení produkčního buildu aplikace. Lze také vytvořit obecný konfigurační soubor, ze kterého mohou ostatní webpack nastavení vycházet.

```
const config = merge(commonConfig, {
    mode: 'development',
    output: {
        filename: 'js/main.js',
        chunkFilename: 'js/[name].[chunkhash].js',
        hotUpdateChunkFilename: 'hot/[hash].hot-update.js',
        hotUpdateMainFilename: 'hot/[hash].hot-update.json'
    },
    devServer: {
        hot: true,
        host: '127.0.0.1',
        port: '8100',
        contentBase: 'dist',
        historyApiFallback: true,
        writeToDisk: true,
        disableHostCheck: true
    },
    devtool: 'cheap-module-source-map'
});
```

Listing 4 *– Ukázka webpack-dev-server konfigurace*

```
"scripts": {
    "build": "webpack --config ./config/webpack.prod.ts --colors",
    "start": "webpack-dev-server --config ./config/webpack.dev.ts --colors"
}
```

Listing 5 *– Přiřazení různých Webpack konfigurací v souboru package.json*

6.1.1.7 Babel

Babel je řada nástrojů, které se používají k převodu kódu ECMAScript 2015 (ECMAScript – JavaScriptový standard) a novějších na zpětně kompatibilní verzi JavaScriptu podporovanou v současných a starších prohlížečích nebo prostředích [24]. Tím umožňuje při vývoji využívat ty nejnovější funkce a způsoby zápisu JavaScriptu a převodem na starší verzi zaručuje, že bude výsledný kód funkční i v prohlížečích, které použité funkce nepodporují.

6.1.2 Serverové technologie

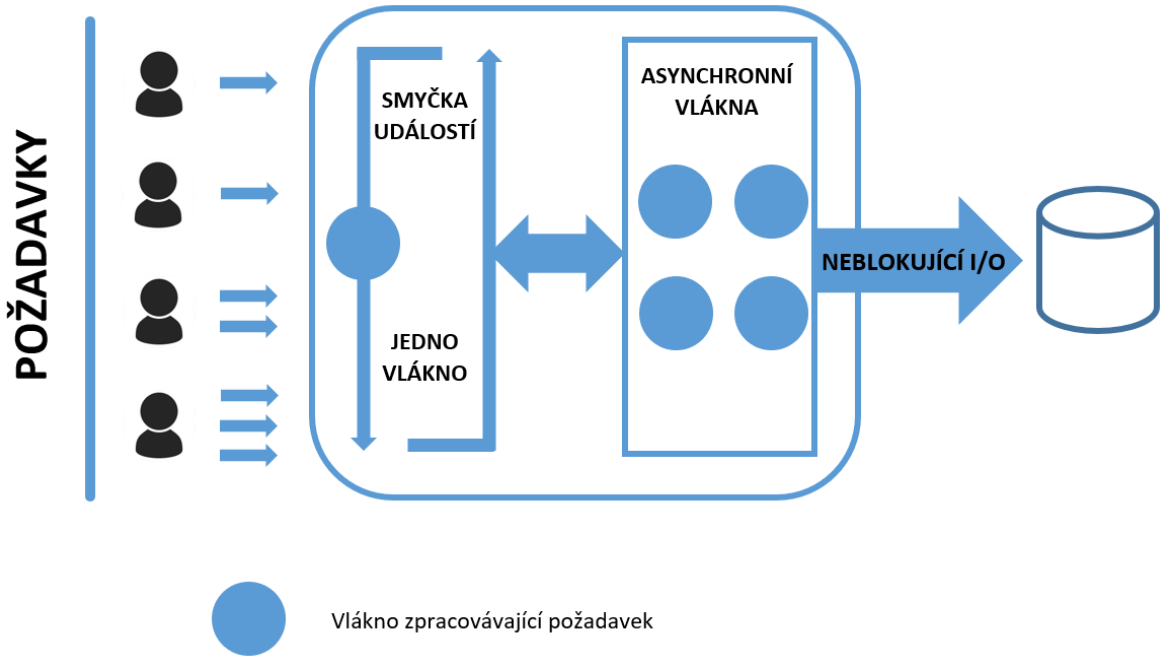
Následující část popisuje nejdůležitější nástroje použité při vývoji serverové části.

6.1.2.1 Node.js

Node.js je open-source back-endové (serverové) prostředí pro JavaScript. Základním „motorem“ je interpret V8 od Google, který dokáže spouštět javascriptový kód na straně sevreru [25].

Používá událostmi řízený (event-driven) model a asynchronní (neblokující) vstupně-výstupní operace. Tento model umožňuje minimalizovat režii procesoru a maximalizaci jeho výkonu. Před zpracováním prvního požadavku si Node.js načte a inicializuje potřebné nástroje a funkce a začne naslouchat příchozím spojením. Jednotlivé požadavky dále směřuje na konkrétní controller (funkce obsluhující požadavek daného typu), který ho dále zpracovává [26]. Na rozdíl od tradičnějších HTTP serverů dokáže Node.js zpracovávat více požadavků najednou. Není totiž omezen dostupným počtem vláken, ale pracuje pouze v jednom. Z tohoto důvodu je nutné při vývoji serveru využívat asynchronních funkcí pro zpracovávání požadavků. Požadavky zpracovávány synchronně by jinak blokovaly celý server, dokud by nebyly dokončeny.

V synchronním programování program klasicky zpracovává kód postupně řádek po řádku a dokud není příkaz dokončen, tak není možné začít vykonávat další. V Node.js je také možné psát tímto způsobem, ale pouze tehdy, nedochází-li ke zpracovávání uživatelských požadavků (např. inicializace prostředí, načítání konfigurace, atd). Pří zpracovávání HTTP požadavků je ovšem nutné programovat asynchronně, jinak by hrozilo zablokování vlákna a tím zablokování celého serveru, dokud by nebyl daný požadavek zpracován, což by způsobovalo dlouhotrvající odezvy serveru. Následující obrázek (Obr. 13) zjednodušeně prezentuje výše popsaný model.



Obrázek 19 *– Jednovláknový neblokující I/O model (upraveno z [26])*

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Listing 6 *– Ukázka vytvoření jednoduchého serveru v Node.js (převzato z [25])*

6.1.2.2 Express.js

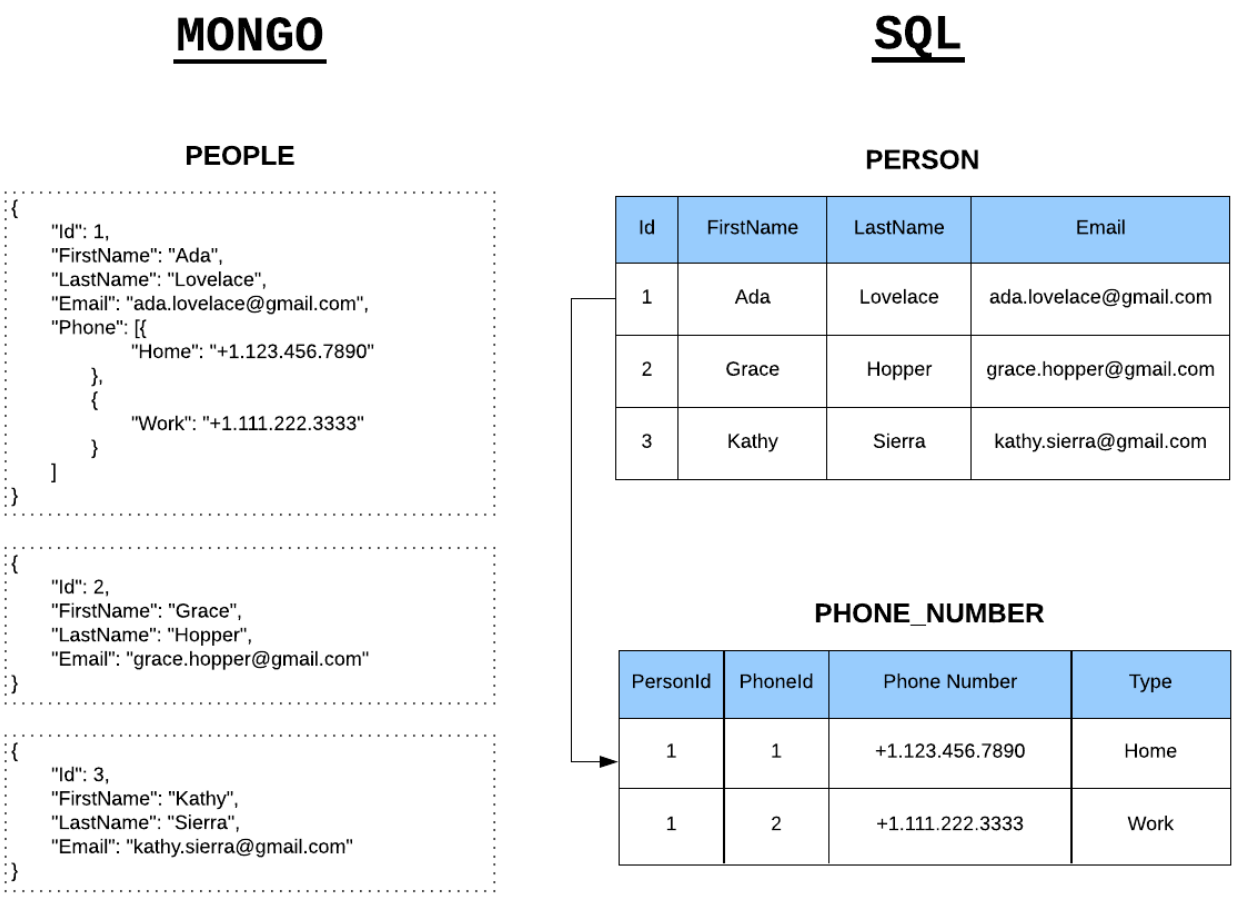
Express.js (Express) je open-source webový framework sloužící jako nadstavba nad prostředím Node.js. Nabízí širokou škálu funkcí a minimální rozhraní potřebné k vytváření aplikací. Mezi hlavní výhody nástroje Express patří nastavení zjednodušené směřování (route) jednotlivých požadavků podle URL (endpoint) k potřebným controllerům, které požadavek dále zpracují [27]. Tato funkce značně usnadňuje tvorbu aplikačního programovacího rozhraní (API).

```
const express = require('express')
const app = express()
const port = 3000

// Define Routes
app.use('/api/user', UserRouter);
app.use('/api/auth', AuthRouter);
app.use('/api/location', LocationRouter);

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

Listing 7 *– Konfigurace jednoduchého serveru pomocí Express.js*

6.1.2.3 MongoDB

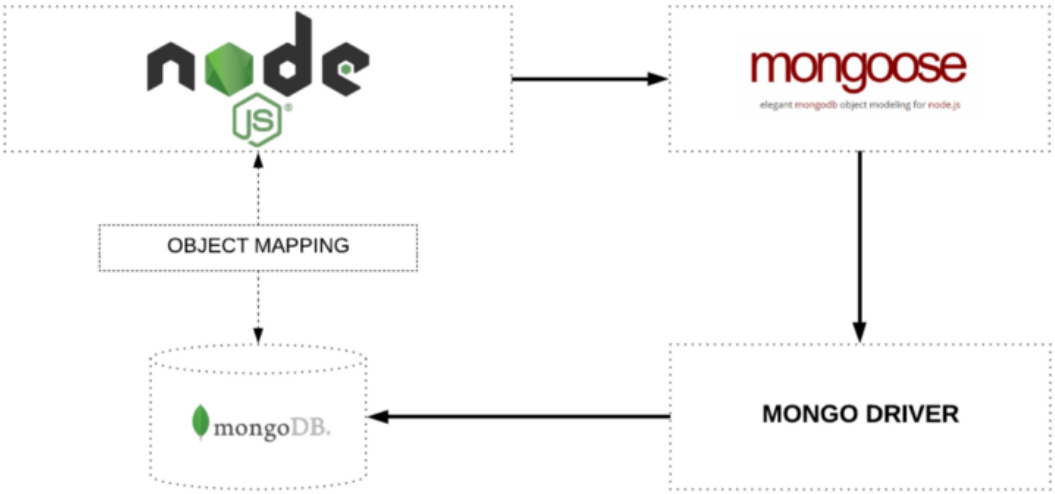
MongoDB je multiplatformní dokumentově orientovaná NoSQL databáze. Oproti tradičním relačním databázím (např. MySQL, MS-SQL) využívajících tabulky používá kolekce formátu BSON. Jedná se o obdobu formátu JSON, který navíc nabízí podporu pro datové typy. Oproti sloupcům a řádkům v SQL databázích tak využívá objekty a pole. Jednou z velkých výhod MongoDB oproti klasickým SQL databázím je, že struktura jednotlivých dokumentů se může lišit a umožňuje vkládání vnořených dokumentů. Struktura totiž není striktní a vynucována jako v případě SQL. Tento přístup zrychluje vývoj a snižuje složitost nasazení [28].



Obrázek 20 *– NoSQL vs. SQL (dostupné z [29])*

6.1.2.4 Mongoose

Mongoose je ODM (Object Data Modeling) knihovna pro MongoDB a Node.js. Spravuje vztahy mezi daty, poskytuje ověřování definovaných schémat a používá se k překladu mezi objekty v kódu a jejich prezentací v MongoDB [29]. Ukázka mapování objektů Node.js a MongoDB spravované pomocí Mongoose je znázorněna na následujícím obrázku (Obr. 21).



Obrázek 21 *– Mapování objektů pomocí ODM Mongoose (dostupné z [29])*

6.1.2.5 Google Cloud Storage

Google Cloud Storage je RESTful (REpresentational State Transfer – architektonický styl využívající podmnožinu HTTP) webová služba sloužící pro ukládání souborů online a přístup k datům v infrastruktuře Google Cloud Platform. Služba kombinuje výkon a škálovatelnost Google Cloudu s pokročilými možnostmi zabezpečení a sdílení. Jedná se o infrastrukturu jako službu (IaaS) srovnatelnou například se službou Amazon S3 [30].

Pro komunikaci se vzdáleným úložištěm je využita knihovna *@google-cloud/storage* sloužící jako Node.js klient. Umožňuje čtení a zápis ze stejnojmenného uložiště. Google Cloud lze využít pro celou řadu scénářů, včetně poskytování obsahu webových stránek, ukládání dat, nebo pro archivaci různého druhu dokumentů. V projektu je tato služba využita jako vzdálené uložiště všech multimediálních objektů (obrázky, 3D modely, atd).

6.1.2.6 Morgan

Morgan slouží jako logovací middleware (funkce, která je při vstupně výstupních operacích prováděna před výkonem samotného požadavku) HTTP požadavků. Nabízí několik možností nastavení podrobnosti zápisu [31]. Sleduje a vypisuje jednotlivé dotazy na server a napomáhá tak

k odhalování případných problémů a chyb při vývoji i při běhu v produkčním prostředí. Na následující ukázce kódu (Listing 8) je názorně ukázáno, jak je logování v aplikaci řešeno.

Z kódu je patrné, že si aplikace pomocí proměnné prostředí (environment variable – env) ověří, zda kód běží v produkčním prostředí. Pokud ano, vytvoří si soubor, do kterého všechny logy zapisuje. Pokud je aplikace spuštěna ve vývojovém prostředí, tak se logy vypisují pouze do terminálu aplikace.

```
morgan.token('date', () => {
    return moment().tz('Europe/Prague').format('DD/MM/YYYY HH:mm:ss');
});
if (process.env.NODE_ENV === 'production') {
    const dir = path.join(__dirname, 'logs');

    try {
        if (!fs.existsSync(dir)) {
            fs.mkdirSync(dir);
        }
    } catch (err) {
        console.log(err);
    }

    const accessLogStream: WriteStream = fs.createWriteStream(
        path.join(dir, 'access.log'),
        {
            flags: 'a'
        }
    );

    app.use(morgan('combined', { stream: accessLogStream }));
} else {
    app.use(morgan('common'));
}
```

Listing 8 *– Logování pomocí middlewaru Morgan*

6.1.2.7 Helmet

Helmet je užitečný Node.js modul, který pomáhá zabezpečit HTTP hlavičky vracené Express aplikací. HTTP hlavičky jsou důležitou součástí protokolu HTTP, ale z pohledu koncového uživatele jsou obecně transparentní. Poskytují důležitá metadata HTTP požadavku, nebo odpovědi, aby mohl prohlížeč, nebo server v dané transakci odeslat potřebné informace.

Vzhledem k tomu, že klasický uživatel HTTP hlavičku nevidí, mají vývojáři tendenci je ignorovat. Hlavičky ovšem obsahují velké množství citlivých informací o serveru a ty mohou být zneužity. Příklad častého úniku citlivé informace z Express aplikace je hlavička X-Powered-By. Jedná se o informativní HTTP hlavičku, která sděluje, na jakém systému, nebo frameworku je server provozován a uvádí i verzi daného systému. Díky této informaci se z takové aplikace může stát snadnější cíl útoku zvenčí. Pomocí Helmet je v základním nastavení tato hlavička zcela skryta a neodesílá se s odpovědí do prohlížeče, případně lze nakonfigurovat jiný druh a verzi systému, která se bude s odpovědí vracet. Aplikace fungující na frameworku Express tak může vracet klamavou informaci o tom, že běží například na PHP serveru verze 5.1.2.

Díky Helmet lze HTTP hlavičky zabezpečit. Neznamená to, že by aplikace byla chráněná proti všem druhům útoků, ale případný útočník bude mít značně ztíženou práci [32].

6.1.2.8 Express Slow Down

Express Slow Down je middleware omezující rychlost reakcí Express serveru na příchozí požadavky. Jedná se o základní ochranu proti DDoS útokům (Denial of service – útok, při kterém se pachatel snaží narušit služby poskytovatele zaplavením cílového serveru nadbytečným množstvím dotazů). Místo toho, aby server požadavky úplně blokoval, tak je pouze omezuje a zpomaluje odpověď serveru na požadavek. Middleware sleduje příchozí spojení a pokud požadavky z jednoho zdroje přesáhnou během určeného času specifikovaný limit, začnou se odpovědi úmyslně zpomalovat. A to s každým příchozím požadavkem v daném čase o definované zpoždění plus hodnotu zpoždění minulého [33]. Je-li například limit počtu požadavků stanoven na 100 během 10 minut, tak se po 10 minutách začne každý další požadavek na server zpomalovat o daný čas – například 500 milisekund. 101. požadavek v deseti minutovém intervalu by tedy na

odpověď čekal o 500 milisekund déle a 120. požadavek v daném intervalu by musel počkat na odpověď už 10 vteřin navíc.

6.1.3 Webové technologie

Následující část popisuje nejdůležitější nástroje použité při vývoji webové části.

6.1.3.1 React.js

React.js (React) je open-source front-endová (webová) JavaScriptová knihovna spravovaná společností Facebook. Usnadňuje tvorbu interaktivních UI (User Interface – uživatelské rozhraní) nebo jeho komponent. React se zabývá pouze správou aktuálního stavu aplikace a jeho vykreslením do DOMu (Document Object Model – umožňuje přistupovat k jednotlivým objektům XML (XHTML) dokumentu a pracovat s nimi) [34]. Pokud se na knihovnu podíváme z hlediska klasické MVC architektury, jedná se právě o část *view*, tedy vrstvu pohledu, která slouží k prezentaci dat.

Základním stavebním kamenem knihovny jsou znovupoužitelné HTML elementy se zapouzdřenou funkcionalitou, které jsou v Reactu označovány jako komponenty. Skládáním těchto komponent vzniká komplexní UI aplikace. Jednotlivé komponenty získávají své vlastnosti (props) a udržují si vlastní stav (state). Jedná se o deklarativní způsob práce s daty, který umožňuje předvídatelnost chování komponent a usnadňuje tak jejich tvorbu.

K běhu React aplikace je zapotřebí základní HTML soubor, do kterého se linkuje výsledný JavaScript a jeden prázdný HTML element (typicky *div*), do kterého se celá React aplikace vykresluje. React si v DOMu daného HTML souboru nalezne obalovací element a pomocí funkce *render* se do něj vykreslí (Listing 9).

```
import React from 'react';
import { render } from 'react-dom';
import { AppContainer } from 'react-hot-loader';
import App from 'app';

const rootElement = document.getElementById('app');

render(<App />, rootElement);
```

Listing 9 *– Vykreslení React aplikace do obalovacího HTML elementu*

K pohodlnějšímu zápisu kódu nabízí React takzvaný JSX. Je to syntax podobný jazyku XML, nebo HTML, který je rozšířen o možnost zápisu ECMAScriptu. Syntax je poté transformován pomocí nástroje Babel do klasických JavaScript objektů. V ukázce kódu níže (Listing 10) je prezentován zápis pomocí JSX oproti čistému zápisu pomocí JavaScriptových objektů [35].

```
const jsxElement = <h1>This is JSX</h1>;
ReactDOM.render(jsxElement, document.getElementById('root'));

const jsElement = React.createElement('h1', {}, 'This is JavaScript');
ReactDOM.render(jsElement, document.getElementById('root'));
```

Listing 10 *– Rozdílné možnosti tvorby React komponenty (převzato z 14.04.2021 14:14:00)*

Manipulace s DOM prohlížeče je zásadní v každém moderním webu. Bohužel tato funkce je jednou z nejpomalejších operací celého JavaScriptu. Tuto skutečnost také zhoršuje fakt, že většina knihoven a frameworků aktualizuje DOM mnohem častěji, než je potřeba. K řešení tohoto problém používá React takzvaný virtuální DOM (VDOM). VDOM je odlehčená virutální kopie reálného DOM a je uložena v paměti prohlížeče ve formě JavaScriptového objektu. Pokaždé, kdy React zaznamená změnu, která je potřeba vykreslit, tak vytvoří nový VDOM. Porovná novou

verzi VDOM s tou předešlou (tento proces se nazývá *diffing*) a zjistí, které prvky se změnili a potřebují znovu vykreslit. Ty se poté vykreslí do skutečného DOM prohlížeče. Omezuje se tak potřeba vykreslování všech elementů znovu, ale překreslují se pouze ty, které to skutečně vyžadují (tento proces se nazývá *reconciliation*). Pokaždé, kdy je zaznamenána změna mezi dvěma virtuálními DOM objekty, tak se celý proces opakuje. Vykreslování změn do reálného DOM prohlížeče zahrnuje parsování stylů, změn uzlů a také různé kalkulace kvůli rozložení jednotlivých elementů. VDOM celý tento proces značně urychluje [36].

6.1.3.2 React Router

React Router je standardní směrovací (routovací) knihovna pro knihovnu React. Umožňuje tvorbu SPA (Single Page Applicaition – jednostránkové aplikace) díky synchronizaci uživatelského rozhraní (UI) s URL adresou prohlížeče. Poskytuje jednoduché API s výkonnými funkcemi. Mezi základní funkce patří přechod mezi stránkami na bez jejich opětovného načítání prohlížečem. Veškeré směrování probíhá stále v jednom okně a komponenty se načítají na základě URL adresy. Ale umožňuje využití například i takzvaného líného načítání (lazy loading) kódu, kdy se komponenta nenachází v hlavním JavaScriptovém balíčku, ale načítá se až při příchodu na komponentu, respektive na stránku, která kód potřebuje. Tyto části kódu nejsou v hlavním balíčku obsaženy a ten je díky tomu velikostně o to menší [37].

6.1.3.3 Redux

Redux je open-source knihovna pro správu stavu aplikace. Jedná se o předvídatelný stavový kontejner obalující celou React aplikaci a všem zanořeným komponentám umožňuje na základě jednoduchého API přístup ke sdíleným informacím a možnost jejich změny. V situaci, kdy potřebuje několik komponent využívat stejná data je mnohem lepší mít globální úložiště v podobě Redux kontejneru než každé komponentě zvlášť předávat potřebná data. Redux také zamezuje situaci, kdy je potřeba předávat data komponentě zanořené několik úrovní. Tyto data je potřeba předat každé jednotlivé komponentě, než se dostanou k té, která je využije (tomuto procesu se říká *prop drilling – Listing. 11*). Díky Redux knihovně tento problém odpadá. Nemusejí se

předávat žádná data a každá komponenta, která potřebuje informace z globálního uložiště si je zavolá jednoduchou funkcí.

```
interface Props {
    message: string;

const GreatGrandParent: React.FC<Props> = ({ message }) => {
    return <GrandParent message={message} />;
};
const GrandParent: React.FC<Props> = ({ message }) => {
    return <Parent message={message} />;
};
const Parent: React.FC<Props> = ({ message }) => {
    return <Children message={message} />;
};
const Children: React.FC<Props> = ({ message }) => {
    return <div>Great grandpa says {message}</div>;
};
```

Listing 11 *– Prop drilling v Reactu (převzato z [38])*

6.1.3.4 Axios

Axios je odlehčený HTTP klient, který poskytuje abstrakci na prohlížečovým objektem XMLHttpRequest (XHR) a používá se k provádění HTTP požadavků. Použitím této knihovny odpadá potřeba parsovat přijatá nebo odesílaná data do formátu JSON. Axios vrací rovnou datový objekt, s kterým je možné pracovat. Umožňuje asynchronní provádění HTTP metod a nabízí funkci přerušení odchozích požadavků [39].

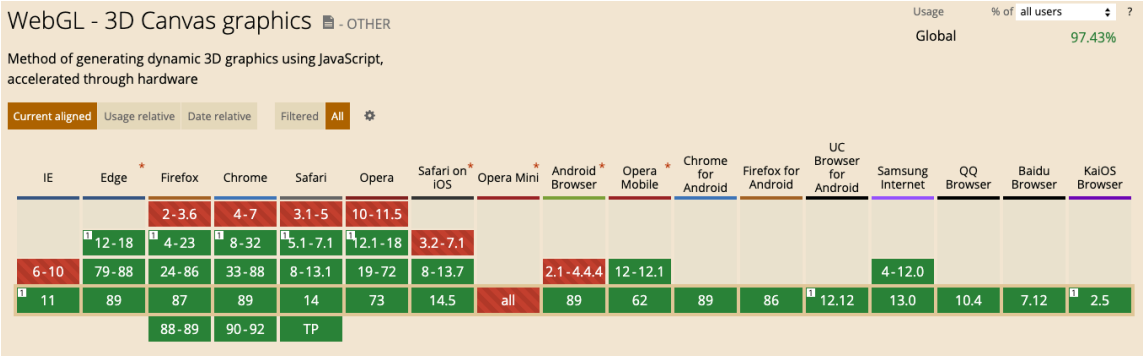
6.1.3.5 AR.js

AR.js je knihovna pro rozšířenou realitu na webu (WebAR) a nabízí možnost vykreslování modelů na základě skenování markeru, obrázku, nebo podle polohy zařízení. WebAR je technologie umožňující překrývání obsahu do skutečného světa. Lze ji využít pro několik typů zařízení, jako jsou mobilní telefony, náhlavní soupravy (headsety), nebo displeje stolních počítačů [40]. AR.js vychází ze dvou knihoven:

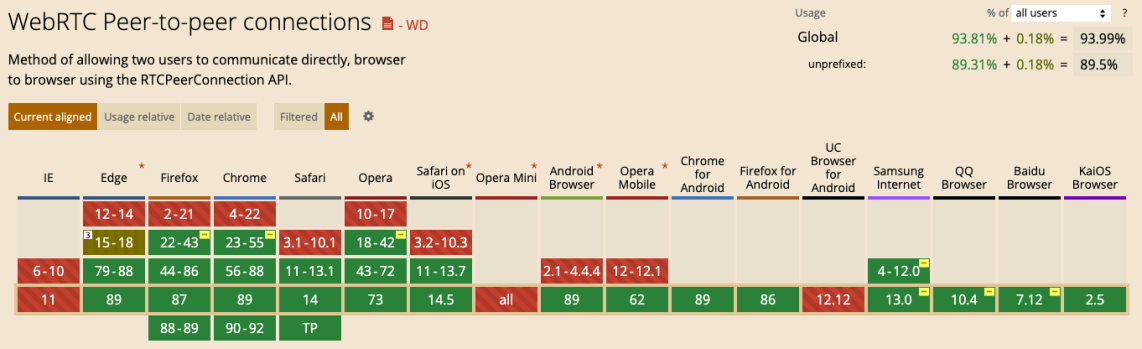
- **a-frame** – open-source framework pro vytváření VR na webu [41]
- **three.js** – JavaScriptová knihovna pro programování rozhraní, které se používají k vytváření a zobrazování animované 3D počítačové grafiky ve webovém prohlížeči [42]

AR.js kompatibilní se všemi webovými prohlížeči, které podporují tyto standardy:

- **WebGL** – JavaScript API pro vykreslování interaktivní 2D a 3D grafiky v prohlížečích, které ho podporují (Obr. 22). A to bez použití přídavných pluginů (modulů). Díky využití GPU (grafická řídící jednotka) zařízení umožňuje akceleraci vykreslení obrazu a efektů [43].
- **WebRTC** – Web Real-Time Communication, neboli webová komunikace v reálném čase. Tento projekt umožňuje webovým prohlížečům a mobilním aplikacím komunikaci v reálném čase (RTC) prostřednictví API. Umožňuje komunikaci zvuku a videa uvnitř webových stránek přímou komunikací typu P2P (peer-to-peer) [44].



Obrázek 22 *– Prohlížeče podporující WebGL (převzato z [45]*



Obrázek 23 *– Prohlížeče podporující WebRTC (převzato z [46])*

6.1.3.6 Styled components

Knihovna umožňující zápis CSS (kaskádové styly) pomocí JavaScriptu. Knihovna poskytuje kombinaci těchto dvou technologií (CSS a JS), díky čemuž není nutné vytvářet několik stylovacích tříd pro HTML elementy, ale stačí předat informaci styled components objektu, který styl podle potřeby upraví. Každý jednotlivý styl elementu slouží jako React komponenta [47]. Níže naleznete ukázku využití styled components (Listing 12).

```
import styled from 'styled-components';

const ReactComponent = () => {
    const [isLightTheme, setIsLightTheme] = useState<boolean>(true);

    return (
        <>
            <StyledText $isLightTheme={isLightTheme}>Themed text!</StyledText>
            <StyledButton
                onClick={() => setIsLightTheme((prevState) => !prevState)}
            >
                Toggle theme
            </StyledButton>
        </>
    );
};

interface TextProps {
    $isLightTheme: boolean;
}

const StyledText = styled.p<TextProps>`
    background-color: ${({ $isLightTheme }) =>
        $isLightTheme ? 'white' : 'black'};
    color: ${({ $isLightTheme }) => ($isLightTheme ? 'black' : 'white')};
    padding: 1rem;
`;

const StyledButton = styled.button`
    background-color: black;
    color: white;
    padding: 1rem;
`;
```

Listing 12 – *Ukázka využití styled components*

6.1.3.7 Material-UI

Knihovna obsahující sadu React komponent založených na Material designu od Google. Slouží pro rychlejší a snadnější vývoj aplikace. Obsahuje řadu předdefinovaných komponent, které lze použít ve výchozím stavu, nebo jejich vzhled upravit pomocí globálního nastavení. Umožňuje také upravovat jednotlivé komponenty, k čemuž slouží knihovna styled components, popsaná výše [48].

6.2 Aplikační rozhraní

V této části bude popsána funkcionalita REST API serveru. Potřebná data jsou načítána z databáze a odesílána do klientské části.

Ještě před tím, než server začne přijímat požadavky od klientů, nastaví si logování pomocí knihovny Morgan. Dále zabezpečí odesílané HTTP hlavičky knihovnou Helmet a nastaví CORS (Cross-origin resource sharing), což je mechanismus, který umožňuje nastavení přijímání požadavků jen z určitých domén [49]. Jak bylo řečeno již výše v dokumentu, komunikace probíhá pomocí datového typu JSON. Dalším krokem je tedy nastavení parsování komunikace právě na tento datový typ. Server dále nakonfiguruje ochranu proti DDoS útokům knihovnou express-slow-down. Poté se už může připojit k databází a začít naslouchat příchozím požadavkům.

Při příchodu požadavku si server zkontroluje adresu (endpoint), na kterou byl požadavek vyslán a směřuje ho ke správnému kontroleru, který začne požadavek zpracovávat. Pokud není dotazovaný endpoint nalezen, aplikace vrátí chybovou hlášku s informací o problému (Listing 14). Ukázka směřování (routování) ke správným kontrolerům je zobrazena níže (Listing 13).

```
const app: Application = express();

app.use('/api/user', UserRouter);
app.use('/api/auth', AuthRouter);
app.use('/api/monument', MonumentRouter);
app.use('/api/monument-asset', MonumentAssetRouter);
app.use('/api/quizlet', MonumentQuizletRouter);
app.use('/api/location', LocationRouter);

app.use(notFound);
app.use(errorHandler);
```

Listing 13 *– Směrování v Express aplikaci*

```
import { Request, Response, NextFunction } from 'express';

const notFound = (req: Request, res: Response, next: NextFunction) => {
    const error: Error = new Error(`Not Found - ${req.originalUrl}`);
    res.status(404);
    return next(error);
};

const errorHandler = (error: any, req: Request, res: Response) => {
    const statusCode: number = res.statusCode === 200 ? 500 : res.statusCode;
    res.status(statusCode);
    return res.json({
        message: error.message,
        stack: process.env.NODE_ENV === 'production' ? '💔' : error.stack
    });
};
```

Listing 14 *– Funkce řešící chybné požadavky na server*

Požadavky od klientů mohou přijít chybné, i když směřují na správný endpoint. Pro kontrolu parametrů a těla požadavku je proto využit middleware express-validator. Díky tomu lze nastavit například potřebné parametry, nebo jejich datové typy pro správné zpracování požadavku. Pokud middleware odhalí chyby, přidá je do těla požadavku a propustí kód dále ke zpracování do kontroleru. Ten si před samotným zpracováváním svých interních funkcí ověří, zda tělo

požadavku neobsahuje chyby přidané pomocí express-validator. Pokud ne, pokračuje ve zpracovávání funkce dál. V opačném případě přeruší zpracovávání požadavku a odešle tuto informaci klientovi zpět (Listing 15).

```
router.get(
    '/:monument_id',
    [
        param('monument_id', 'Wrong monument_id format').custom((monumentId) =>
            Types.ObjectId.isValid(monumentId)
        )
    ],
    async (req: Request, res: Response): Promise<any> => {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json({
                errors: errors.array()
            });
        }

        // ...
    });
```

Listing 15 – *Validace chybného parametru HTTP požadavku*

Pokud server přijme požadavek neobsahující žádné chyby, může ho začít zpracovávat. V následující ukázce kódu (Listing 16) je znázorněno zpracování jedné z nejjednodušších implenetovaných API metod – získání dat o přihlášeném uživateli. Kontroler je zjednodušen díky využití middlewaru *auth* (Listing 17). Ten je přidán do funkce routeru pomocí druhého parametru a má za úkol zpracovat JWT přijmutý s klientským požadavkem, dekódovat ho a získat tak data o uživateli. Tyto data poté vkládá do těla požadavku, což je klasický JavaScriptový objekt a je možné s ním tedy tak pracovat. Pokud middleware v databázi nalezne uživatele podle identifikátoru získaného z JWT, přidá do těla požadavku klíč *user* a nastaví mu hodnotu těla JWT, obsahující dekódované informace o uživateli. Pomocí funkce *next* umožní pokračování následující funkci, v tomto případě už samotnému kontroleru. Pokud by za tímto middlewarem

následoval nějaký další, kód by pokračoval jím. Middlewarů je možné přidávat neomezené množství.

```
// @route   GET api/auth
// @desc    Get data about logged user
// @access  Private
router.get(
    '/',
    auth,
    async (req: Request, res: Response): Promise<any> => {
        try {
            const user = await User.findById(req.user.id);
            return res.json(user);
        } catch (err) {
            return res.status(500).send('Server error');
        }
    }
);
```

Listing 16 *– Kontroler sloužící k získávání dat uživatele na základě jeho identifikátoru*

```
export const auth = (req: Request, res: Response, next: NextFunction) => {
    const token: string | undefined = req.header('x-auth-token');

    if (!token) {
        return res.status(401).json({
            msg: 'No token, authorization denied'
        });
    }
    const JWT_SECRET: string = process.env.JWT_SECRET!;

    try {
        const decoded = jwt.verify(token, JWT_SECRET);
        req.user = decoded.user;
        return next();
    } catch (err) {
        return res.status(401).json({ msg: 'Token is not valid' });
    }
};
```

Listing 17 *– Middleware auth*

V aplikaci se nachází ještě další tři middleware funkce, a to pro ověření role uživatelů. Jedná se o middlewary:

- *monumentAdmin* – používá se při přístupu k zabezpečeným endpointům a ověřuje, jestli je uživatel administrátorem projektu, nad kterým se snaží provádět požadovanou operaci
- monumentManage*r* – totožná funkcionalita jako u middlewaru *monumentAdmin*, ale ověřuje pouze roli managera projektu
- *appAdmin* – ověřuje roli administrátora aplikace

Díky těmto middlewarům lze jednoduše zabezpečit neomezený počet endpointů. V následující ukázce kódu (Listing 18) je znázorněno použití většího počtu middlewarů – ověření, zda je uživatel přihlášen, jeho role, zpracování nahrávaného souboru a ověření důležitých dat těla požadavku. Po tom, co požadavek projde úspěšně všemi middleware funkcumi, začne ho kontroler zpracovávat. V následující ukázce se konkrétně jedná o kontroler sloužícímu pro přidávání mediálních objektů (*asset*) k projektu (*monument*).

```
// @route   POST api/monument-asset/monument/:monument_id
// @desc    Add monument asset
// @access  Private + Organization manager
router.post(
    '/monument/:monument_id',
    [
        auth,
        monumentManager,
        upload.single('file'),
        check('name', 'Asset must have a name').exists(),
        check('assetType')
            .exists()
            .withMessage('You must specify asset type')
            .bail()
            .matches(/\b(?:image|video|sound|arModel)\b/)
            .withMessage('You must specify VALID asset type'),
        check('hidden', 'You must specify if asset will be hidden or not')
            .exists()
            .bail()
            .isBoolean()
    ],
    async (req: Request, res: Response): Promise<any> => {
        // ... controller code
    }
);
```

Listing 18 – *Ukázka využití několika middleware funkcí*

6.3 Schéma dokumentů databáze

ODM knihovna Mongoose slouží ke správě a vytvoření schémat dokumentů MongoDB. Tyto schémata definují strukturu dokumentů a jejich vlastností a definují metody instancí dokumentů [50]. Nad vytvořenou instancí dokumentu lze pracovat pomocí mongoose metod přímo v kontrolerech aplikačního rozhraní. V práci jsou použity metody pro čtení, zápis, změnu, nebo mazání záznamů dokumentů. Tato skupina metod se souhrně označuje jako CRUD (create, read, update, delete). Na následujícím úryvku kódu je znázorněno schéma uživatele (Listing 19).

Vyhledávání v dokumentu pomocí instance schématu můžete nalézt v ukázce kódu uvedené dříve v této práci (Listing 16).

```
import { Schema, Document, model } from 'mongoose';

const UserSchema = new Schema(
    {
        email: {
            type: String,
            required: true,
            unique: true
        },
        password: {
            type: String,
            required: true,
            min: 8,
            select: false
        },
        role: {
            type: String,
            enum: [RoleEnum.ADMIN, RoleEnum.MANAGER, RoleEnum.USER],
            default: 'user'
        },
        verifiedForProjects: {
            type: Boolean,
            default: false,
            requied: true
        }
    },
);
export interface IUser extends Document {
    firstName?: string;
    lastName?: string;
    email: string;
    password: string;
    role: RoleEnum;
    verifiedForProjects: boolean;
    createdAt: Date;
    updatedAt: Date;
}
export default model<IUser>('User', UserSchema);
```

Listing 19 *– Schéma dokumentu uživatele*

6.4 Nahrávání multimediálních objektů

Všechny multimediální objekty (*asset*) jsou z webové aplikace nahrávány na server, kde se zjistí dodatečné informace o typu souboru (MIME Type). Obrázky, videa a zvukové záznamy se nahrávají jako jednoduché soubory. 3D modely typicky obsahují celou řadu souborů a v klientské části je vyžadováno jejich uložení pomocí archivu *zip* (formát podporující bezztrátovou kompresi dat [51]).

Při uložení souboru si ve webové aplikaci uživatel vybere, jaký typ souboru chce nahrát, poté se mu zpřístupní pole pro nahrávání a soubor může být pomocí REST API odeslán na server. Po doručení požadavku o uložení souboru zkontroluje příslušný kontroler, zda se shoduje uvedený datový typ v těle požadavku s datovým typem doručeného souboru. Pokud vše proběhne bez problému, přistoupí se k nahrání do cloudu. V cloudu jsou jednotlivé projekty řazeny do složek s názvy podle vzoru – *název_projektu_identifikátor_projektu*. K pojmenování by postačil identifikátor, ale název projektu je na začátku názvu uváděn pro lepší orientaci pro případ, že by k datům v cloudu bylo potřeba přistoupit přímo, a ne jenom pomocí REST API. Po určení výchozí (root) složky pro soubor se upraví (normalizuje) název souboru tak, aby neobsahoval žádné speciální znaky a mezery. Server nyní disponuje potřebnými informacemi, kam přesně objekt uložit a začne jej nahrávat. V případě problémů s nahráváním vrátí server chybovou hlášku. Pokud vše proběhne v pořádku, uloží se informace o umístění souboru v cloudu do databáze a po dotazu na daný objekt (*asset*) se v odpovědi vrátí také URL souboru v cloudu ve formátu *název_projektu_identifikátor_projektu/typ_souboru/název_souboru.datový_typ*. Například tedy *CeskeBudejovice_38dad831mdja/img/kasna.png*.

V případě ukládání 3D modelu je průběh nahrávání složitější. Ve webové aplikaci se nic nemění. Uživatel vybere, že chce nahrát zip archiv obsahující 3D model a nahraje ho. Na straně serveru je ovšem nutné celý archiv rozbalit a soubory do cloudu nahrávat postupně. Před samotným nahráváním je nutné, aby server provedl kontrolu, zda se skutečně jedná o archiv obsahující data o 3D modelu. Tato kontrola probíhá kontrolou datového typu jednotlivých souborů. Dále je potřeba určit výchozí soubor obsahující odkazy na externí binární soubory a textury. Například soubor typu gLTF (GL Transmission Format – specifikace pro efektivní

přenos a načítání 3D scén a modelů). Po úspěšném nahrání je odkaz na hlavní soubor v cloudu uložen do databáze a odesílán při požadavku na data o objektu.

Hlavní funkce pro ukládání souborů do cloudu je zobrazen v následující ukázce kódu (Listing 20).

```
public async saveAsset(
        asset: any,
        assetName: string,
        assetType: AssetType
    ): Promise<GoogleCloudReturnType | null> {
        let file: any | UnzippedType;
        let coreFileName: string;

        if (assetType === AssetEnum.AR_MODEL) {
            const zipHandler = new ZipHandler(asset.buffer);
            const unzippedAsset = zipHandler.unzipArModel();

            if (
                !unzippedAsset ||
                !unzippedAsset.unzip ||
                !unzippedAsset.coreFileName
            ) {
                return null;
            }

            file = unzippedAsset.unzip;
            coreFileName = GoogleCloud.normalizeFileName(
                unzippedAsset.coreFileName
            );
        } else {
            file = asset.buffer;
            coreFileName = GoogleCloud.normalizeFileName(asset.originalname);
        }

        const destFolder = this.getDestinationFolder(
            false,
            assetName,
            assetType
        );

        if (assetType === AssetEnum.AR_MODEL) {
            file.map(async (unzippedBuffer: any) => {
                const { unzipBuffer, unzipFileName } = unzippedBuffer;
```

```
            const fileType = await fromBuffer(unzipBuffer);
            const mimeType = fileType ? fileType.mime : null;

            try {
                const res = await this.saveSingleFile(
                    unzipBuffer,
                    destFolder,
                    unzipFileName,
                    mimeType
                );

                if (!res) return null;
            } catch (err) {
                console.log(err);
                return null;
            }
        });
    } else {
        try {
            const res = await this.saveSingleFile(
                file,
                destFolder,
                coreFileName
            );

            if (!res) return null;
        } catch (err) {
            console.log(err);
            return null;
        }
    }

    const publicUrl = GoogleCloud.getMainFilePublicUrl(
        destFolder,
        coreFileName
    );

    return {
        mimetype: asset.mimetype,
        url: publicUrl,
        rootDir: destFolder
    };
}
```

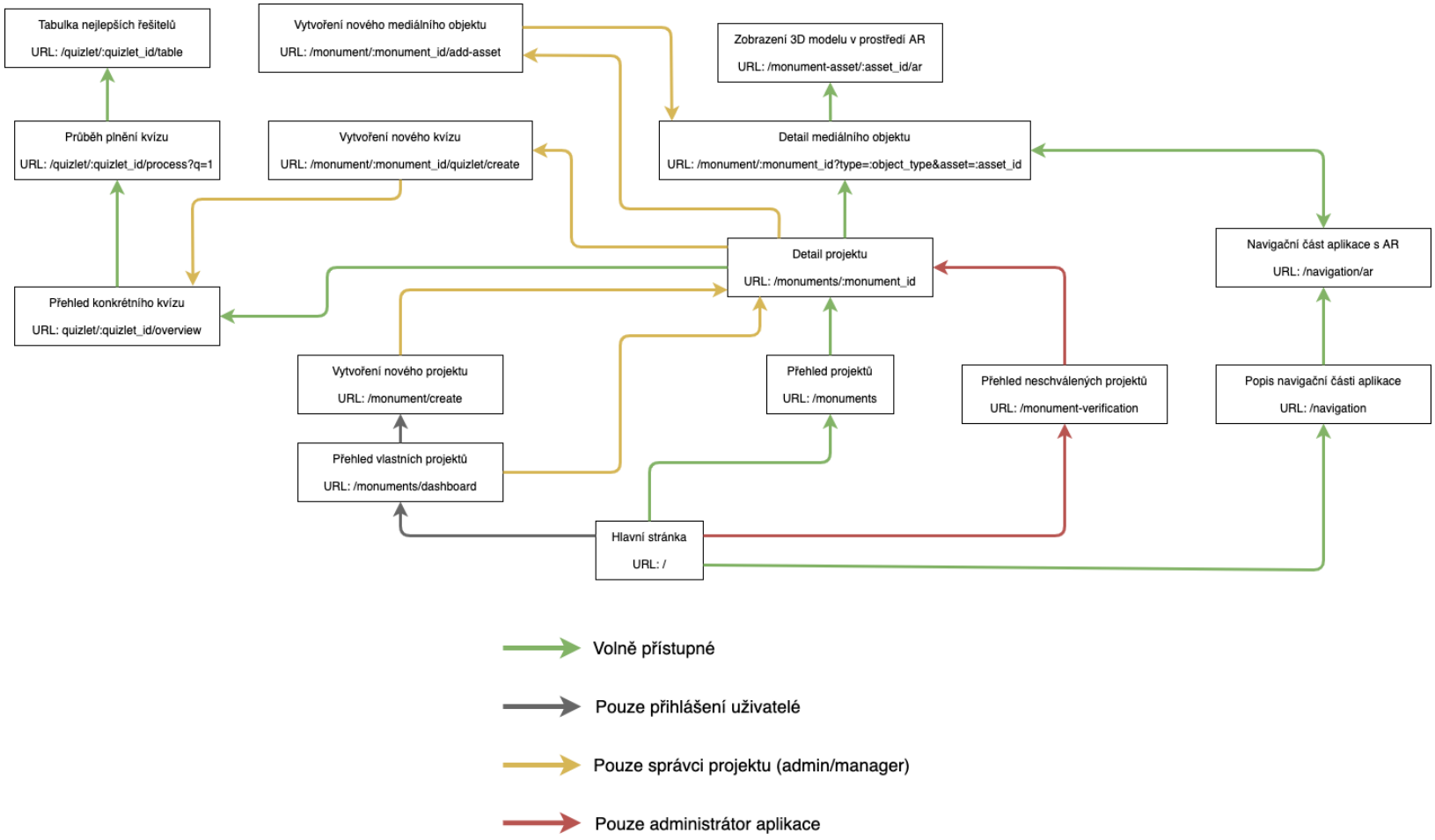
Listing 20 *– Hlavní funkce pro ukládání souborů do cloudu*

6.5 Webová část aplikace

Jak již bylo pospáno dříve, webová aplikace se fakticky skládá z aplikací dvou. Z hlavní aplikace tvořené pomocí knihovny React. A z druhé, která slouží pro vykreslování prvků rozšířené reality.

6.5.1 React aplikace

Aplikace vytvořena pomocí knihovny React tvoří hlavní webové rozhraní, díky kterému uživatel s programem komunikuje. Jedná o Single Page Aplikaci (SPA). Je tedy nutné řešit i věci, které by klasicky obstarával server, jako například přechody (routování) mezi stránkami. Po kliknutí na odkaz se uživateli jeví, že přišel na novou stránku. Ve skutečnosti je ale pouze vykreslena jiná komponenta. Tento přístup tak nevyžaduje žádné aktualizování stránky při načítání jako u vykreslování pomocí serveru. Tuto funkcionalitu umožňuje knihovna react router. Na každou stránku je možné se samozřejmě dostat pomocí příslušného URL nebo přímo odkazy v aplikaci. Přehled všech nabízených stránek aplikace, jejich návaznosti a přístupnosti dle rolí jsou znázorněny v následujícím schématu (Obr 24).



Obrázek 24 – *Schéma routování React aplikace*

O směrování (routování) se stará komponenta *Router*. Ta obsahuje komponentu *Switch* (Listing 21), která obstarává přepínání mezi jednotlivými komponentami, které jsou do ní vložené jako cesty (*Route*). Každá *Route* má dva povinné parametry – cestu (URL adresa) a komponentu, kterou má na dané adrese vykreslit. Pro potřeby chráněných *Route*, na které se dostanou jen správci projektů, nebo pouze administrátor aplikace byly navíc vytvořeny dvě komponenty *AppAdminRoute* a *ProtectedRoute* (Listing 22), které ověřují roli uživatele a na základě role mu dají ke komponentě přístup, nebo ho směřují na stránku pro přihlášení.

```
<Switch>
    <AppAdminRoute
        exact
        path="/monument-verification"
        component={MonumentVerificationPage}
    />

    <ProtectedRoute exact path="/dashboard" component={DashboardPage} />
    <ProtectedRoute exact path="/monument/create" component={MonumentCreatePage} />

    <Route exact path="/" component={MainPage} />
    <Route exact path="/monuments" component={MonumentsPage} />
    <Route exact path="/monument/:monumentId" component={SpecificMonumentPage} />

    <Route path="*" exact component={Error404} />
</Switch>
```

Listing 21 *– Switch komponenta s vnořenými Route komponentami*

```
export const ProtectedRoute = ({ component: Component, ...rest }) => {
    const { isAuthenticated, isLoading } = useSelector(
        (state: RootState) => state.auth
    );

    if (isLoading) return;

    return (
        <Route
            {...rest}
            render={(props) => {
                if (isAuthenticated) {
                    return <Component {...props} />;
                }

                return (
                    <Redirect
                        to={{
                            pathname: '/login',
                            state: {
                                from: props.location
                            }
                        }}
                    />
                );
            }}
        />
```

Listing 22 *– ProtectedRoute komponenta ověřuje, zda je uživatel přihlášen*

V předešlém úryvku kódu (Listing 22) je na druhém řádku volána metoda knihovny Redux *useSelector*. Funkce vrátí požadovaná data z globálního uložiště, tedy zda je uživatel autentizovaný, nebo informaci o tom, že se data načítají. V kódu je názorně zobrazeno jednoduché použití dané funkce. Tato funkce a všechna data obsažena v globálním Redux uložišti jsou obdobným způsobem přístupná napříč celou aplikací.

I když je Router stěžejní komponenta pro chod celé aplikace, není v hiearchii komponent umístěna nejvýše. Celou aplikaci obaluje komponenta *Provider* knihovny Redux. *Provider*

obsahuje veškeré informace z globálního uložiště a díky obalení celé aplikace mají k těmto datům přístup všechny ostatní komponenty. Pod globálním úložištěm se nachází komponenta *StylesProvider* z knihovny Material-UI. Ta zaručuje přístup ke všem komponentám z knihovny ostatním komponentám aplikace. Dále *ThemeProvider* z knihovny styled components obsahující nakonfigurované globální styly a témata, které je opět možné využít v celé aplikaci (Listing 23). Další úroveň tvoří již popsaný *Router*.

```
import PersonAddIcon from '@material-ui/icons/PersonAdd';

export const StyledAddIcon = styled(PersonAddIcon)`
    fill: ${({ theme }) => theme.color.primary.default};
    cursor: pointer;
    transition: 300ms ease fill;

    &:hover {
        fill: ${({ theme }) => theme.color.primary.dark};
    }
`;
```

Listing 23 *– Úprava stylu Material-UI komponenty s využitím styled components*

6.5.2 AR aplikace

AR aplikace byla původně obsažena v hlavní React aplikaci. Kvůli problémům s načítáním potřebných knihoven až na vyžádání (lazy-loading) a kvůli trhanému vykreslování modelů došlo k jejímu odtržení a do hlavní React aplikace se vkládá pomocí rámce *iframe*. Ten umožňuje načtení webové stránky uvnitř dalšího HTML dokumentu. AR aplikace slouží čistě k vykreslování rozšířené reality. Na stránce s AR je možné narazit i na různá tlačítka, nebo načítací obrazovky. Ty ovšem spadají pod hlavní React aplikaci a pouze překrývají *iframe*, v kterém se vykresluje AR prostředí. Obě aplikace jsou stylovány tak, aby běžný uživatel nepoznal rozdíl a aplikace se mu jevila jako jednolitý celek.

Z využité AR.js knihovny byly použity dvě funkce:

- Vykreslení 3D modelů pomocí skenování markeru (Obr 25)
- Navigace pomocí značek rozšířené reality

Obrázek 25 *– HIRO marker použitý pro vykreslování AR obsahu*

Po úspěšném naskenování markeru se na obrazovce zařízení vykreslí 3D model (Obr. 26). Modely mohou být i animované a je možné s nimi pomocí gesta prstů na displeji mobilního telefonu manipulovat. Lze je zvětšovat, zmenšovat, nebo s nimi otáčet. Tyto fuknce (zvětšování, zmenšování, rotace) knihovna sama o sobě nenabízí a bylo nutné je doimplementovat. Následující ukázka kódu znázorňuje šablonu (Listing 24) nutnou pro správné vykreslení 3D modelu pomocí AR.js. Šablona je psaná pomocí nástroje EJS (Embedded JavaScript). V elementu *a-scene* se nachází speciální atributy ve formátu „*<%= klíč %>*“. Tento formát je využit kvůli snazšímu nastavení šablony pomocí nástroje webpack. Ten přidává konfiguraci do šablony podle toho, zda se jedná o vývojové, nebo produkční prostředí. V produkčním prostředí například není vhodné mít zobrazenou tabulku s frekvencí vykreslování modelu, a naopak ve vývojovém prostředí je tato informace velmi důležitá.

Obrázek 26 *– Vykreslený 3D model po naskenování markeru*

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-
scale=1.0">
    <title>HistoryARound </title>

<body style="margin: 0px; overflow: hidden;">
    <a-scene <%=htmlWebpackPlugin.options.debug %>
             stats="<%= htmlWebpackPlugin.options.isDebugOn %>"
             embedded arjs="sourceType: webcam; debugUIEnabled: <%= htmlWebpackPlugin.options.isDebugOn %>;
             trackingMethod: best;"
            renderer="logarithmicDepthBuffer: true;" vr-mode-ui="enabled: false" gesture-detector id="scene">

                <a-marker preset="hiro" raycaster="objects: .clickable" emitevents="true"
                    cursor="fuse: false; rayOrigin: mouse;" smooth="true" id="targetMarker">

                    <a-entity position="0 0 0" scale="0.05 0.05 0.05" class="clickable" gesture-handler
                        id="model-entity">
                    </a-entity>

                </a-marker>
                <a-entity camera></a-entity>
    </a-scene>
</body>
</html>
```

Listing 24 *– EJS šablona pro vykreslení 3D modelu pomocí rozšířené reality naskenováním markeru knihovnou AR.js*

6.5.3 Komunikace mezi React a AR aplikací

Jelikož je AR aplikace do hlavní React aplikace vkládána pomocí iframu, probíhá komunikace mezi nimi pomocí metody PostMessage. Ta umožňuje komunikaci mezi dvěma aplikacemi, respektive mezi dvěma *window* (globální objekt reprezentující okno prohlížeče) objekty.

Při příchodu na stránku, ve které se má načítat aplikace s rozšířenou realitou (vykreslení 3D modelu pomocí markeru, nebo navigační stránka), začne React naslouchat příchozím PostMessage zprávám. Ihned poté přistoupí k načítání AR aplikace uvnitř iframu. Po načtení AR aplikace zažádá uživatele o přístup k videokameře, aby bylo možné poskytnout prostředí rozšířené reality. Zde nastává problém, protože při udělení souhlasu k přístupu ke kameře zařízení nedal uživatel ve skutečnosti povolení k přístupu aplikaci s rozšířenou realitou, ale udělil přístup hlavní aplikaci s Reactem. Je proto nezbytné přidat k iframe elementu atribut *allow*, v kterém vyjmenujeme funkce a zdroje, které chceme, aby měla aplikace uvnitř iframu povolené. V tomto případě kameru zařízení.

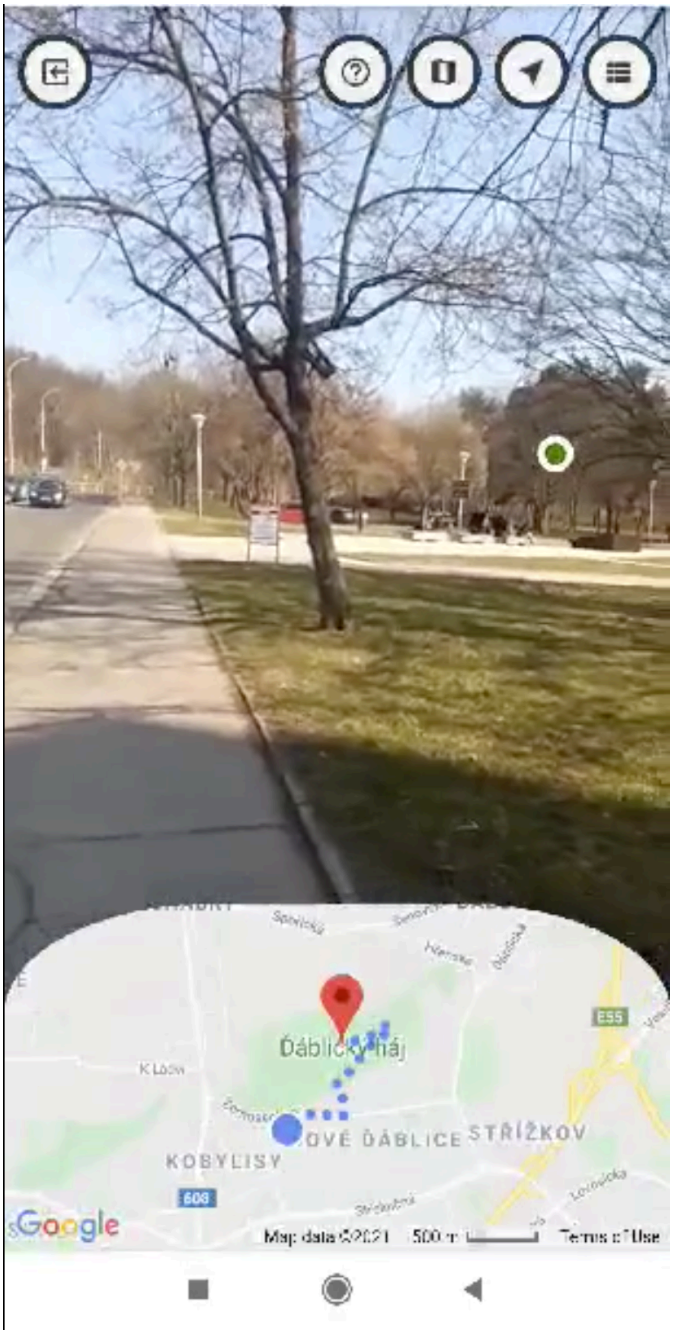
Jako další čeká React aplikace na signál od AR aplikace, že načetla své prostředí a všechny potřebné zdroje a je připravena přijmout další informace. React aplikace tuto zprávu zpracuje a obratem zasílá informace o 3D objektu, zejména pak cílovou adresu směřující do cloudu, kde se model nachází. AR aplikace přijme zprávu s informacemi o objektu a odešle do React okna zprávu, že se pokouší objekt načítat. React aplikace na to zareaguje vykreslením zprávy s informací, že probíhá načítání objektu – uživatel je tak tedy o všem informován. Pokud se model nepodaří načíst, je uživatel opět informován a vyzván k znovunačtení stránky. Pokud je model bez problému načtený, odešle opět AR aplikace zprávu do hlavní React aplikace, že je model připraven a React aplikace vyzve uživatele k naskenování markeru. Poté co uživatel marker naskenuje, informační okno zmizí a uživateli se na displeji zobrazí 3D model, s kterým může interagovat způsoby popsanými dříve v práci.

Komunikace aplikací na stránce „Navigace“ probíhá téměř totožně. Rozdíl je ovšem v tom, že se nepošle jeden model a poté se pouze nekontrolují stavy naskenovaného/nenaskenovaného markeru. V části „Navigace“ React aplikace potřebuje povolení ke geolokaci zařízení. Získaný přístup opět nasdílí AR aplikaci v iframu. Jelikož se uživatel se zařízením pohybuje, tak je nutné informace o jeho poloze často obnovovat do aktuálního stavu. Pokaždé kdy dojde k obnově těchto

údajů, odešle React aplikace tato data aplikaci s rozšířenou realitou. Navigační stránka má několik funkcí:

- Najít nejbližší objekty v okolí – pošle na server data o své poloze a ten na základě této informace prohledá databázi a vrátí informace o všech objektech v pěti kilometrovém radiusu od polohy uživatele
- Najít nejbližší – najde ten nejbližší objekt vzhledem k poloze uživatele
- Navigovat k bodu – uživatel si může vybrat některý z jemu poblíž nalezených bodů a odnavigovat se přímo k němu po pěších stezkách (k této funkci slouží služba Google Directions API)

Stránka navigace zobrazuje v části obrazovky Google mapu, která je běžná z řady klasických webů. Ale v druhé části obrazovky se nachází prostředí rozšířené reality. Pokud si uživatel zobrazí nejbližší bod, nebo body v jeho okolí, tak ho aplikace informuje o tom, zda a případně kolik se jich nachází poblíž. Tyto body se zobrazí na mapě, lze je rozkliknout a přečíst si o nich dodatečné informace. Ale také se tyto body zobrazí v prostředí rozšířené reality, kde je každý bod znázorněn zeleným bodem (Obr 27). Tyto body směřují směrem k místu, kde se objekt v reálném světe podle poskytnutých souřadnic nachází a usnadňují tak celkovou navigaci.



Obrázek 27 *– Ukázka stránky navigace s využitím AR*

7 Závěr

Hlavním cílem diplomové práce byl návrh a implementace webové aplikace pro sdílení digitálních prezentací kulturních objektů s využitím technologie rozšířené reality na webu.

Po úvodní kapitole práce stručně definuje pojem rozšířené reality a popisuje koncept virtuálního kontinua vymezeného P. Milgramem. Následně bylo provedeno zmapování existujících aplikací s rozšířenou realitou v oblasti historie a kultury.

Dále se práce zabývá tvorbou scénářů aplikace pro zvýšení zájmu o kulturní obsah. Oproti zkoumaným aplikacím popsaných ve 3. kapitole, které nabízejí pouze mobilní verzi, vypracovaná aplikace nabízí přístup pomocí webového prohlížeče, což umožní přístup většímu množství zařízení než jen mobilním telefonům. Na scénáře bylo nahlíženo ze dvou úhlů. Z pohledu uživatele přistupujícího k aplikaci pomocí stolního počítače a z pohledu uživatele, který aplikaci používá na mobilním zařízení. Obě možnosti bylo třeba zvážit kvůli skutečnosti, že valná většina zkoumaných aplikací nenabízela možnost získávání informací jinak než prostřednictvím rozšířené reality. Toto zjištění se v plánování aplikace odráží a scénáře použití počítají s oběma typy uživatelů. Kapitola dále shrnuje konkrétní funkcionality aplikace.

Na základě scénářů použití a návrhu funkcionality aplikace z předchozí části došlo k implementaci aplikace. Nejprve bylo nutné vybrat vhodnou sadu technologií jak na webu, tak na serveru. Ty nejstěžejnější technologie jsou popsány v kapitole 6. Technologie byly vybrány na základě požadavků zadání a také na základě zkušeností autora práce s jazykem JavaScript.

Po výčtu technologií práce obsahuje popis postupu implementace a důležité části kódu. Nejprve bylo nutné implementovat aplikační rozhraní serveru pomocí frameworku Express a webovou aplikaci vytvořenou pomocí knihovny React. Doplňuje jí další webová aplikace poskytující rozhraní pro rozšířenou realitu a využívající knihovnu AR.js. Ta je do hlavní aplikace tvořené Reactem vkládána pomocí iframu. Obě webové aplikace se navzájem doplňují a koncovému uživateli se tak jeví jako jeden ucelený program.

Hlavní výhodou aplikace vytvořené v této práci oproti aplikacím popsaných v kapitole 3 je skutečnost, že vytvořená aplikace umožňuje přístup bez nutnosti ji do zařízení fyzicky instalovat. To je pro použití ostatních zmíněných aplikací nezbytné. Popsané aplikace jsou dále použitelné pouze na mobilních zařízeních. Díky volbě aplikace webové není dodaný program tímto směrem

omezen a umožňuje přístup pomocí webového prohlížeče pro širokou škálu zařízení. Další výhoda dodané aplikace je ta, že umožňuje tvorbu projektů a přidávání multimediálních objektů komukoliv pouze s nutností registrace. S tím jde ovšem ruku v ruce problém s ukládáním 3D modelů a jejich následným vykreslováním. Pokud nastane situace, že je do systému vložen 3D objekt, který se poté v prostředí rozšířené reality vykreslí zmenšený, zvětšený, nebo špatně natočený, je to daň za možnost volného přidávání obsahu a systém sám nedokáže nastavit specifická pravidla pro každý jeden objekt. Jedná se o úvodní vykreslení objektu, uživatel poté může s modelem různě manipulovat, takže tato nedokonalost má za následek pouze úvodní nehezké vykreslení některých modelů. Zmiňovaný problém se naopak netýká aplikací popsaných ve 3. kapitole. Ty tyto případy mohou řešit interně a pro každý objekt nastavit vlastní pravidla vykreslování díky tomu, že interní týmy dodávaných aplikací jsou jediné, kdo může 3D objekty do systému vkládat. Tyto aplikace mají dále výhodu v kvalitě vykreslování prostředí rozšířené reality. Díky tomu, že jsou instalovány přímo v zařízení, mají přímý přístup k jeho hardwaru, což umožňuje rychlejší a plynulejší vykreslování modelů. Vypracované aplikaci přístup k hardwaru zařízení zprostředkovává webový prohlížeč. V tomto případě nemůže být dosaženo stejné úrovně jako při použití nativní aplikace. I přes tuto skutečnost jsou ovšem výsledky uspokojující.

Pro budoucí vylepšení aplikace by bylo vhodné zaměřit se na problém s vykreslováním výchozího stavu některých modelů v prostředí rozšířené reality. U přidávaných modelů by se mohla ještě před vložením do systému nastavit jejich úvodní pozice a konfigurovat celá scéna rozšířené reality oproti modelu. Uživatel, který model přidává by tak měl pokaždé možnost ovlivnit úvodní stav vykresleného objektu. Tato data by byla ukládána v databázi společně s modelem a předávána aplikaci, která by model konfigurovala do požadované úvodní formy.

Vzhledem k výše uvedenému lze požadované a navržené funkce systému považovat za splněné, stejně tak i cíle diplomové práce.

8 Seznam obrázků

9 Seznam tabulek

10 Seznam ukázek kódu

11 Citovaná literatura

[1] P. Milgram a F. Kishino, „A Taxonomy of Mixed Reality Visual Displays", *IEICE Trans Inf. Syst.*, roč. E77-D, č. 12, s. 1321–1329, pro. 1994 (viděno dub. 12, 2021).

[2] „Augmented Reality Guide", *AstroReality*. https://blog.astroreality.com/augmented-reality-guide-ar/ (viděno dub. 12, 2021).

[3] S. Yuen, Y. Yuen, G. Yaoyuneyong, a E. Johnson, „Augmented Reality: An Overview and Five Directions for AR in Education", *J. Educ. Technol. Dev. Exch.*, roč. 119, s. 119–140, lis. 2011 (viděno dub. 12, 2021).

[4] „Augmented reality app for Scotland | Scotland.org", *Scotland*. https://www.scotland.org/about-scotland/scotlands-stories/portal-ar (viděno dub. 12, 2021).

[5] S. Jørgensen, „Bedste AR app: Portal AR – Step Into Scotland", *Mobil.nu*, dub. 17, 2018. https://mobil.nu/apps-og-spil/bedset-ar-app-portal-ar-step-scotland-88605 (viděno dub. 12, 2021).

[6] „BBC launches its first augmented reality app - Civilisations AR". bbc.com/mediacentre/latestnews/2018/civilisations-ar-launches/ (viděno dub. 12, 2021).

[7] „Civilisations AR - Apps on Google Play". https://play.google.com/store/apps/details?id=uk.co.bbc.civilisations&hl=en_US&gl=US (viděno dub. 12, 2021).

[8] „PIVOTtheWorld: Engage history, change your point of view.", *pivottheworld*. https://www.pivottheworld.com (viděno dub. 12, 2021).

[9] „How two Palestinian Americans plan to PIVOT the world", *Mondoweiss*, úno. 21, 2015. https://mondoweiss.net/2015/02/working-pivot/ (viděno dub. 12, 2021).

[10] „Chevré 3D, the augmented reality app", *Artefacto*. https://www.artefacto-ar.com/en/reference/chevre-3d-the-augmented-reality-app/ (viděno dub. 12, 2021).

[11] N. Media, „Mobilní průvodce Arthur ukazuje památky a akce v rozšířené realitě", *Médiář*, kvě. 23, 2019. https://www.mediar.cz/mobilni-pruvodce-arthur-ukaze-pamatky-v-rozsirene-realite/ (viděno dub. 12, 2021).

[12] A. Kos, „Arthur je virtuální průvodce městem. Ukáže nejen památky, ale i akce v okolí", *MobilMania.cz*. https://mobilmania.zive.cz/clanky/arthur-je-virtualni-pruvodce-mestem-ukaze-nejen-pamatky-ale-i-akce-v-okoli/sc-3-a-1345859/default.aspx (viděno dub. 12, 2021).

[13] „VIDEO: Chcete cestovat časem? Na Kuksu vás mobilní aplikace přenese o několik století zpět", *Hradec Králové*, kvě. 03, 2019. https://hradec.rozhlas.cz/video-chcete-cestovat-casem-na-kuksu-vas-mobilni-aplikace-prenese-o-nekolik-7908327 (viděno dub. 12, 2021).

[14] F. F. www.fg.cz, 2021 a s, „Visit.More", *Visit.More*. https://www.visitmore.cz/ (viděno dub. 12, 2021).

[15] auth0.com, „JWT.IO - JSON Web Tokens Introduction". http://jwt.io/ (viděno dub. 12, 2021).

[16] auth0.com, „JWT.IO". http://jwt.io/ (viděno dub. 12, 2021).

[17] „What is the MERN Stack? Introduction & Examples", *MongoDB*. https://www.mongodb.com/mern-stack (viděno dub. 12, 2021).

[18] „About npm | npm Docs". https://docs.npmjs.com/about-npm (viděno dub. 12, 2021).

[19] „Yarn: A new package manager for JavaScript", *Facebook Engineering*, říj. 11, 2016. https://engineering.fb.com/2016/10/11/web/yarn-a-new-package-manager-for-javascript/ (viděno dub. 12, 2021).

[20] „Typed JavaScript at Any Scale." https://www.typescriptlang.org/ (viděno dub. 12, 2021).

[21] „About", *ESLint - Pluggable JavaScript linter*. /docs/about/ (viděno dub. 12, 2021).

[22] „What is Prettier? · Prettier". https://prettier.io/index.html (viděno dub. 12, 2021).

[23] „Why webpack", *webpack*. https://webpack.js.org/concepts/why-webpack/ (viděno dub. 12, 2021).

[24] „What is Babel? · Babel". https://babeljs.io/ (viděno dub. 12, 2021).

[25] Node.js, „About", *Node.js*. https://nodejs.org/en/about/ (viděno dub. 12, 2021).

[26] Azat, „You Don’t Know Node: Quick Intro to Core Features". https://webapplog.com/you-dont-know-node/ (viděno dub. 12, 2021).

[27] „Express/Node introduction - Learn web development | MDN". https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (viděno dub. 12, 2021).

[28] „mongodb", bře. 29, 2021. https://www.ibm.com/cloud/learn/mongodb (viděno dub. 12, 2021).

[29] „Introduction to Mongoose for MongoDB", *freeCodeCamp.org*, úno. 11, 2018. https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/ (viděno dub. 12, 2021).

[30] „Google Cloud Storage", *Wikipedia*, [Online]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Google_Cloud_Storage&oldid=1016926842 (viděno: dub. 12, 2021).

[31] „morgan", *npm*. https://www.npmjs.com/package/morgan (viděno dub. 12, 2021).

[32] „Fasten Your Helmet.js (Part 1): Securing Your Express HTTP Headers", *Veracode*. https://www.veracode.com/blog/secure-development/fasten-your-helmetjs-part-1-securing-your-express-http-headers (viděno dub. 12, 2021).

[33] „How to slow down the API Requests in Express", *Reactgo*. https://reactgo.com/express-slow-down-api-requests/ (viděno dub. 12, 2021).

[34] „React – A JavaScript library for building user interfaces". https://reactjs.org/ (viděno dub. 12, 2021).

[35] „Introducing JSX – React". https://reactjs.org/docs/introducing-jsx.html (viděno dub. 12, 2021).

[36] „Virtual DOM and Internals – React". https://reactjs.org/docs/faq-internals.html (viděno dub. 12, 2021).

[37] „Beginner's Guide to React Router", *freeCodeCamp.org*, dub. 05, 2016. https://www.freecodecamp.org/news/beginner-s-guide-to-react-router-53094349669/ (viděno dub. 12, 2021).

[38] „How to avoid prop-drilling in React without using Redux (web development)", *Blog About Web & App Development and Teams*, lis. 21, 2018. https://sunscrapers.com/blog/avoid-prop-drilling-react-without-using-redux-part-1/ (viděno dub. 13, 2021).

[39] E. Kollegger, „What is Axios.js and why should I care?", *Medium*, kvě. 14, 2018. https://medium.com/@MinimalGhost/what-is-axios-js-and-why-should-i-care-7eb72b111dc0 (viděno dub. 12, 2021).

[40] „AR.js Documentation". https://ar-js-org.github.io/AR.js-Docs/ (viděno dub. 12, 2021).

[41] „Introduction", *A-Frame*. https://aframe.io (viděno dub. 12, 2021).

[42] „Three.js Fundamentals". https://threejsfundamentals.org/threejs/lessons/threejs-fundamentals.html (viděno dub. 12, 2021).

[43] „WebGL", *Wikipedia*, [Online]. Dostupné z: https://en.wikipedia.org/w/index.php?title=WebGL&oldid=1016986188 (viděno dub. 12, 2021).

[44] „WebRTC API - Web APIs | MDN". https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API (viděno dub. 12, 2021).

[45] „Can I use... Support tables for HTML5, CSS3, etc". https://caniuse.com/?search=webgl (viděno dub. 12, 2021).

[46] „Can I use... Support tables for HTML5, CSS3, etc". https://caniuse.com/?search=webrtc (viděno dub. 12, 2021).

[47] „A 5-minute Intro to Styled Components", *freeCodeCamp.org*, led. 23, 2017. https://www.freecodecamp.org/news/a-5-minute-intro-to-styled-components-41f40eb7cd55/ (viděno dub. 12, 2021).

[48] „Usage - Material-UI". https://material-ui.com/getting-started/usage/ (viděno dub. 12, 2021).

[49] „Cross-Origin Resource Sharing (CORS) - HTTP | MDN". https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (viděno dub. 12, 2021).

[50] „Mongoose v5.12.3: Schemas". https://mongoosejs.com/docs/guide.html (viděno dub. 12, 2021).

[51] „ZIP (file format)", *Wikipedia*, [Online]. Dostupné z: https://en.wikipedia.org/w/index.php?title=ZIP_(file_format)&oldid=1016707825 (viděno dub. 12, 2021).

Příloha A – Použitý software

Microsoft Word – https://www.microsoft.com/en/microsoft-365/word

Visual Studio Code – https://code.visualstudio.com/

Node.js – https://nodejs.org/en/

Express – https://expressjs.com/

React – https://reactjs.org/

TypeScript – https://www.typescriptlang.org/

NPM – https://www.npmjs.com/

MongoDB – https://www.mongodb.com/2

Postman – https://www.postman.com/

Robo 3T – https://robomongo.org/

GitHub – https://github.com/

Google Cloud Storage – https://cloud.google.com/storage/