



**MATEMATICKO-FYZIKÁLNÍ
FAKULTA**
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

David Nápravník

Softwarové řešení digitálních archivů

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Macková Kateřina

Studijní program: Informatika (B1801)

Studijní obor: IPSS (1801R048)

Praha 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

TODO Podekovani:
Petra Hoffmannová
Kateřina Macková

Název práce: Softwarové řešení digitálních archivů

Autor: David Nápravník

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Macková Kateřina, Ústav formální a aplikované lingvistiky

Abstrakt: Za použití moderních webových a databazových technologií modernizujeme zastaralé knihovní systémy. Snížíme nároky na výkon serveru a síťového rozhraní. Uživatelé těchto systémů poskytneme intuitivní a rychlou webovou aplikaci. Navštěvníkovi pak poskytneme nejen onen knihovní systém, ale celý balíček různých objektů od novinek až po 3D modely historických objektů. Vyvojení a automatizované systémy budou mít přístup k API, jež bude poskytovat uchovávaná data. Zároveň bude sloužit i jako prostředek při komunikaci modulu systému s databází, např. pro umožnění vyhledávání pomocí neuronové sítě.

Klíčová slova: digitální archiv web databáze

Title: Software solution for digital archives

Author: David Nápravník

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Macková Kateřina, Institute of Formal and Applied Linguistics

Abstract: TODO Abstrakt en

Keywords: digital archive web database

Obsah

1	Úvod	3
1.1	Motivace	3
2	Analýza a požadavky	4
2.1	Požadavky na systém	4
2.2	Pro koho je systém určen	4
2.3	Existující produkty	4
2.3.1	KOHA	4
2.3.2	Evergreen	4
2.3.3	SLIMS	5
2.3.4	Souhrn	5
3	Návrh projektu	6
3.1	Výběr technologií	6
3.1.1	Frontend	6
3.1.2	Backend	6
3.2	Diagram systému	8
4	Implementace backendu	9
4.1	Server	9
4.2	Knihovny	9
4.2.1	Knihovna express.js	9
4.2.2	Knihovna mongoose	9
4.3	Dokumentace API	10
4.3.1	Vykreslovací engine pro dokumentaci	10
4.4	Routes	10
4.4.1	Uživatel	11
4.4.2	Autorizace	11
4.4.3	Záznam	11
4.4.4	Stránka	11
4.4.5	Nahrávání souborů	11
4.5	Modely	11
4.5.1	Záznam	11
4.5.2	Stránka	11
4.5.3	Uživatel	12
5	Implementace frontendu	13
5.1	Server	14
5.1.1	Kompilace	14
5.1.2	NPM	14
5.2	Knihovny	14
5.2.1	React	14
5.2.2	Material-ui	14
5.2.3	i18n	14
5.2.4	Babel	14

5.2.5	Webpack	14
5.3	Rozhraní	14
5.3.1	Scény	14
5.3.2	Komponenty	14
5.3.3	Moduly	14
5.4	Lokalizace	14
6	Provázání Backendu a Frontendu, API	15
6.1	Dokumentace online	15
6.1.1	sowtware na online dokumentaci	16
6.2	Backend	16
6.3	API	16
6.3.1	Autentifikace	16
6.4	Frontend a volani API	17
6.4.1	Fetch	17
6.4.2	Existující programy pro práci a testování API	18
7	Moduly	19
7.1	Přidávání nových modulů	19
7.2	Aktuálně nasazené moduly	19
7.2.1	Modul hologram	19
8	instalace a spuštění	21
9	Resení	22
9.1	vysledny web	22
9.2	uzivatelska dokumentace	22
	Závěr	23
	Seznam použité literatury	24
	Seznam obrázků	25
	Seznam tabulek	26
	Seznam použitých zkratk	27
A	Přílohy	28
A.1	První příloha	28

1. Úvod

Cílem této práce je vytvoření webové aplikace pro uchování a následné poskytování informací o historických zdrojích, především z oblasti Krkonoš.

1.1 Motivace

Cílem tohoto projektu je vytvoření webového rozhraní pro ukládání a zobrazování historických i současných záznamů z oblasti Krkonoš.

2. Analýza a požadavky

Vybíráme jen z open source produktů, abychom mohli nahlédnout do jejich kódu a lépe porozumět implementaci jejich komponent. Lépe se pro takový systém vyvíjejí nezávisle moduly a obvykle mají o dost větší komunitu vývojářů a přispěvatelů.

2.1 Požadavky na systém

Systém bude přístupný jako webová aplikace skrze moderní webové prohlížeče. Zadavatel bude moci přidávat, prohlížet, měnit a mazat metadata a k nim příslušné indexy. Redaktor bude moci přidávat a měnit články a novinky na webu. Uživatel bude moci vyhledat záznam podle několika různých kritérií a poté jej zobrazit, též bude moci zobrazit příspěvky na webu, včetně novinek. Externí programátoři budou moci k systému přistupovat přes API a získávat nebo měnit data.

2.2 Pro koho je systém určen

Systém bude sloužit pro historiky jako úložiště dat a pro veřejnost jako jejich zdroj.

2.3 Existující produkty

2.3.1 KOHA



url: <http://www.koha.cz/>

Koha je nejrozšířenější open source systém s širokou komunitou. Byla vyvinuta na Novém Zélandu roku 2000. Ale je stále udržovaná a stále rozšiřovaná (nejnovější update je z konce roku 2020)

Používá SQL databázi. Je psaná v Perlu, na frontendu využívající javascript (ale není jej tolik).

2.3.2 Evergreen



url: <https://eg-wiki.osvobozena-knihovna.cz>

Další z řady knihovních systémů. Používán např. Pedagogickou a Teologickou Fakultou v Praze.

Používá sql databázi, vykreslování probíhá na serveru a nemá uživatelsky přívětivé prostředí.

2.3.3 SLIMS



url: <https://slims.web.id/>

Systém se základní funkcionalitou a přívětivým vzhledem. Není v češtině. Není primárně určen jako knihovní systém, spíš je to univerzální systém na cokoliv, proto není až tak efektivní a jeho nastavování by zabralo mnoho času.

2.3.4 Souhrn

Evergreen a SLIMS jsou systémy, které potřebují silně vyškolenou osobu, aby se o systém starala, narozdíl od KOHY, která je intuitivnější a pro nové uživatele přívětivější. Při zachování stejné, možná i lepší funkcionality.

3. Návrh projektu

3.1 Výběr technologií

3.1.1 Frontend

Trendy v oblasti vývoje webových aplikací se mění rychle avšak velkou změnou, která ovlivnila celý způsob principu jak na stránku nahlížet přinesla technologie **Single page application**, jež je implementovaná např. v knihovně **React** zaštiťovanou společností Facebook.

Single page application

Single page application je technologie umožňující vykreslení jiné stránky, bez nutnosti posílání requestu na server. Uživatel si při prvním spuštění webu stáhne celý balíček webu a při opětovném načtení většinou sahá jen do své lokální cache. Javascriptová knihovna (v tomto případě React) poté stránku překresluje při uživatelské interakci. V případě nutnosti stažení / posílání dat mezi serverem a uživatelem (např. editace záznamu, nebo načtení existujícího záznamu) se volá pouze request k API webové služby a tělo requestu obsahuje pouze užitečné (neredundantní) informace.

React

Knihovna React poskytuje single page application technologii. Jedná se o dobře udržovanou knihovnu, jež byla vyvinuta Facebookem, jakožto náhrada zastaralého konceptu renderování stránky na serveru. Díky tomu servery nemusejí ztrácet výkon s každou změnou na stránce a výkon k renderování se bere z PC uživatele. Jádro této knihovny je velmi dobře optimalizované a poskytuje i řadu debutovaných nástrojů, což je pro větší projekty nepostradatelná výhoda.

Dalsi mozne technologie

Velmi často vykreslování stránek probíhá na serveru, se systémy jako jsou WordPress, psaný v PHP. Takovýto systém je velmi dobře uživatelsky přívětivý, ale z pohledu výkonu má velmi obrovský overhead. V případě implementace knihovniho systému by to znamenalo vykreslovat celou stránku (hlavičku, tělo i zápatí) na serveru, na druhé straně single page application nic nerenderuje, pouze pošle požadované informace.

3.1.2 Backend

Mít single page aplikaci na frontendu znamená, že na backendu musí existovat API, od kterého bude frontend čerpat data. Navíc zde potřebujeme i systém pro statické odesílání balíku celé webové stránky. V rámci udržitelnosti jsem se rozhodl využít jazyk Javascript stejný jako pro frontend. Express.js je knihovna která umožňuje komplexní správu requestu a stala se tudíž jasnou volbou.

Express.js

Express.js poskytuje odesílání statických stránek (Reactího balíku v našem případě), custom requesty pro rozmanité API a také odesílání a lokální ukládání statických souborů, jako obrázků, wordovských i pdf dokumentů atd.

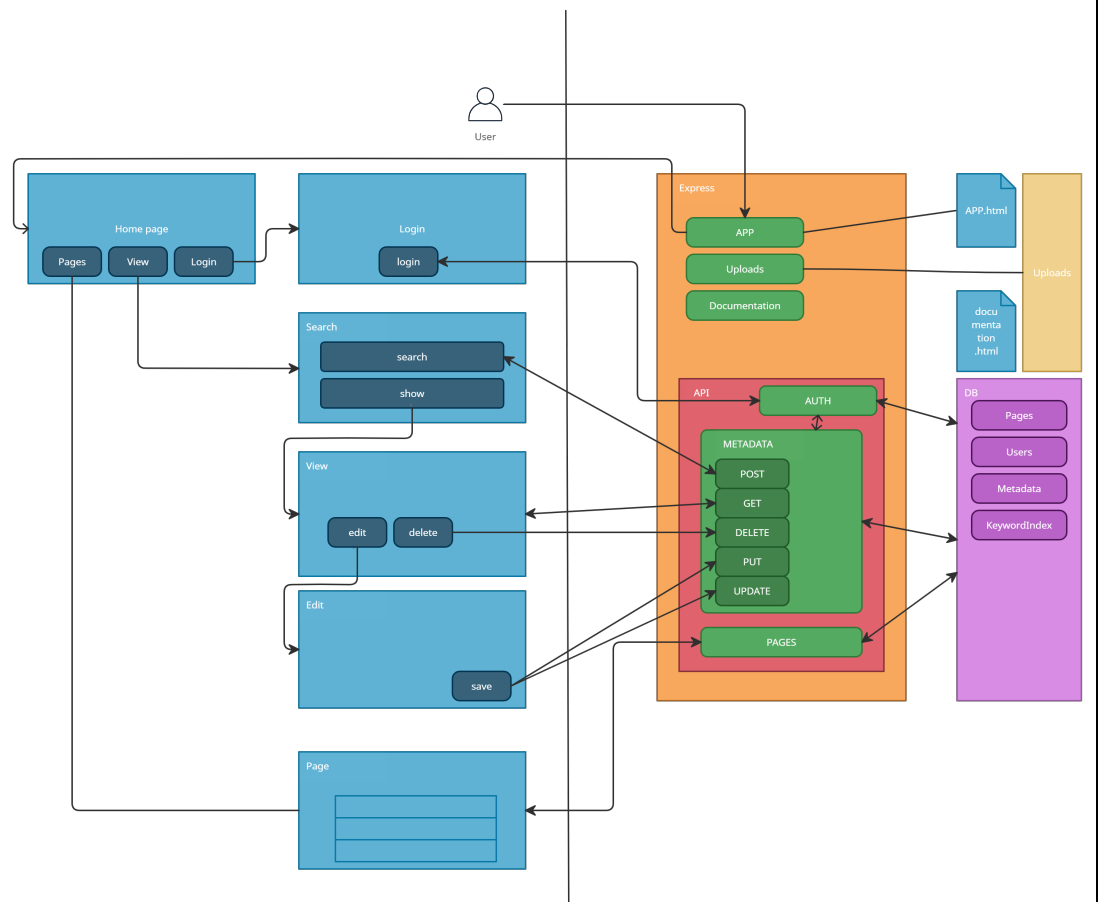
MongoDB

MongoDB je databázový systém typu non-sql. Což primárně znamená, že data neuchovává v tabulkách, ale v tkz. schématech. Což má mnoho výhod, největší je, že nekompletní záznamy nezabírají svými nevyplněnými daty místo v DB a ukládá se opravdu jen to co je potřeba. Další výhodou, je styl ukládání dat a komunikace s DB. Databáze si data uchovává ve formátu BSON (binární JSON rozšířený o datové typy). O data si aplikace žádá pomocí query, která je zcela odlišná od těch u sql-like databází, primárně se zde neposílá query ve formátu string ale jako JSON objekt, díky čemuž např. nenastane známa SQL injection. Znovu ve formátu JSON poté data vrátí aplikaci.

Další možné technologie

Díky oddělení frontendu a backendu (narozdíl např. u WordPressu) je možné na backend nasadit téměř cokoli co umí posílat requesty. Příkladem tomu můžou být skripty v jazycích PHP, C#, Python, nebo Perl. Ale vzhledem k tomu, že jedním z modulů bude neuronová síť na pokročilé vyhledávání, vybíral jsem mezi Pythonem a JavaScriptem, jakožto 2mi jazyky, které mají velmi dobré knihovny pro práci s neuronovými sítěmi.

3.2 Diagram systému



4. Implementace backendu

Backend je část softwaru, která je umístěna na serveru a uživatel z ní vidí jen výstup, jež dostane. Slouží pro přijímání požadavku od klienta a odesílání případné odpovědi nebo provedení nějaké činnosti bez výstupu.

Součástí backendu by mělo být API, které slouží jako toto spojení mezi serverem a klientem. Dále často obsahuje CRON tabulku, která periodicky vykonává nějaký script.

4.1 Server

Stroj na kterém běží aktuální verze systému běží pod systémem Linux (přesněji Ubuntu). Na virtual machine poskytnuté matematicko-fyzikální fakultou UK. Databáze je pak umístěna na serveru u Googlu, kvůli snazší konfiguraci. Není však problém kdykoliv tuto DB přesunout na stejný stroj, na kterém běží zbytek backendu a snížit tím prodlevu způsobenou prací s databází.

4.2 Knihovny

Pro rychlý začátek vývoje byla použita knihovna Express.js, která umožňuje práci s http requestsy potřebnými pro fungování API a to bez většího množství konfigurace ze začátku.

Pro práci s databází byla použita knihovna mongoose, která po rychlé konfiguraci umožňuje práci s databází typu MongoDB.

Dalšími menšími pomocnými knihovnami, jsou **cors** (Cross-Origin Resource Sharing) napomáhající s nastavením hlavičky u API requestu, **md5** pro šifrování hesel a hašování dat a nakonec **cookie-parser** zjednodušující práci s cookies.

4.2.1 Knihovna express.js

Jedná se o minimalistickou a zároveň velmi silnou knihovnu poskytující všestranné prostředky pro web. Obsahuje funkce pro jednoduchou správu HTTP metod a díky tomu je vytváření i větších API jednoduché a přehledné. Má velmi dobrý výkon, který se sice nedá srovnávat s knihovnami psanými v jazyce C, ale z JS knihoven je jeden z nejrychlejších.

Nadstavba pro nahrávání souborů

Nadstavba **express-fileupload** umožňuje v těle requestu rozeznat a zpracovat soubor, který se následně pomocí knihovny **fs** ukládá na server, specificky do složky uploads.

4.2.2 Knihovna mongoose

Ideální knihovna pro práci s databází typu MongoDB. Poskytuje funkce pro snadné připojení k databázi a komunikaci s ní. Hlavní výhodou této knihovny jsou modely, validace a vestavěné přetypování (javascript je dynamicky typovaný

jazyk). S modely se pracuje pomocí tkz. **promise**, která umožňuje řadit akce za sebe, ve stylu:

```
Model.find(body)
  .limit(_limit || 5)
  .exec()
  .then(result => { res.status(200).json(result) })
  .catch(err => { res.status(500).json("something went wrong") })
```

4.3 Dokumentace API

Na adrese <http://quest.ms.mff.cuni.cz/prak/api/documentation> se nachází statický soubor s dokumentací. Celá stránka je zapouzdřena do jediného souboru, který se po načtení vykreslí na straně uživatele, tudíž nezatěžuje server, ale především se dá stáhnout a prohlížet offline.

4.3.1 Vykreslovací engine pro dokumentaci

Aby bylo možné vykreslit stránku až na straně uživatele, je nutné přenášet i script, který to obstará. Tento script se skládá ze dvou částí. Engine na vykreslení a data samotná, která jsou uložena ve formátu JSON.

Script projde veškerá data a podle typu a kontextu postupně vytváří html elementy podle vestavěných šablon a přidává jim příslušné styly a ovládací prvky.

4.4 Routes

Pro rozpoznání, která akce se má vykonat při různých dotazech, se porovnává jak adresa dotazu tak i metoda. Nejdříve se vezme v potaz cesta dotazu za statickou předponou „<http://quest.ms.mff.cuni.cz/prak/api/...>“. Jakmile máme vubranou cestu zjistí se, co vlastně dotaz potřebuje udělat a to jednou z těchto metod:

- POST - většinou obecný dotaz s query v těle
- GET - dotaz požadující 1 záznam, většinou podle ID
- PUT - vytvoření nového záznamu
- PATCH - změna v existujícím záznamu
- DELETE - smazání záznamu

Pokud uživatel má požadované oprávnění, akce se provede. V každém případě uživateli přijde zpětná vazba o úspěchu resp. neúspěchu dotazu. Tělo této zprávy může obsahovat požadovaná data, potvrzení o úspěchu, nebo i chybová zpráva a její příčina.

4.4.1 Uživatel

Slouží pro správu uživatelských účtů. Zakládání nového účtu může zavolat kdokoliv, avšak na zbylé akce, jako mazání, nebo změnu hesla má právo pouze majitel a administrátor. Stejně tak heslo a sessionID nejsou přístupné zvenčí.

4.4.2 Autorizace

Pro ověření, zda přihlášený uživatel má příslušná práva (a to na straně serveru, jelikož lokálně si je může libovoně upravit) se používá **sessionID**, které se poté porovnává v databázi s uživatelem a jeho skutečnými právy.

SessionID získáme po odeslání korektních přihlašovacích údajů. Případně jej ztratíme při odhlášení, nebo expiraci (která je aktuálně nastavena na 1 rok).

4.4.3 Záznam

Pro prohlížení záznamu, resp. jejich vyhledávání není třeba žádné oprávnění. Avšak pro jejich editaci resp. mazání je potřeba mít přiřazena práva pro zápis.

4.4.4 Stránka

Zobrazení stránky též nevyžaduje žádná práva, ale k jejich vytváření je třeba mít roli tkz. „CMS editor“, která jak název napovídá definuje uživatele jakožto editora článků a příspěvků.

4.4.5 Nahrávání souborů

Pro nahrávání souborů je třeba práv pro zápis, stejně jako pro mazání. Zobrazení pak žádná práva nevyžaduje, ani nevyžaduje přístup ze stejné domény.

4.5 Modely

Model nebo též schéma je popis záznamu v databázi, obsahuje datový typ, formát a může obsahovat i referenční cestu. Vpodstatě se jedná o převodní tabulku, aby se Javascript a MongoDB schodli na datovém typu a struktuře (především pro případ reference, nebo pole dat). Část která je pro uživatele nejvíce viditelná je požadavek na unikátní hodnotu pole, nebo požadavek aby hodnota byla nenulová.

4.5.1 Záznam

Každý záznam má přesný popis v online dokumentaci API v pravém sloupci, kde je pravidelně aktualizován.

4.5.2 Stránka

Záznam stránky obsahuje název, jazyk, krátký popis, kategorii a obsah samotný. Pro možnost sledování změn je zde i automaticky generovaný seznam editorů a časů jejich editace.

4.5.3 Uživatel

5. Implementace frontendu

5.1 Server

5.1.1 Kompilace

5.1.2 NPM

5.2 Knihovny

5.2.1 React

5.2.2 Material-ui

5.2.3 i18n

5.2.4 Babel

5.2.5 Webpack

5.3 Rozhraní

5.3.1 Scény

Admin

CMS

Domovská stránka

Zobrazení stránky

Přihlašování

Kontaktní formulář

Vyhledávací

Zobrazovací

Upravovací

5.3.2 Komponenty

ComboBox

Zápatí

Navigační menu

Textové pole s validací

Pole pro nahrávání souborů

Indexy

5.3.3 Moduly

Hologram

5.4 Lokalizace

6. Provázání Backendu a Frontendu, API

6.1 Dokumentace online

Aktuální dokumentace, tak aby mohla být časem aktualizována a mohli do ní být připsány další věci, se nachází na webu quest.ms.mff.cuni.cz/prak/api/documentation. Dokumentace je rozdělena na dvě části.

První část popisuje volání API. Každá metoda (GET, POST atd.) má svůj vlastní účel, je popsán uvnitř URL, na kterou se dotazovat a možné parametry. Spolu s formátem requestu je i zde formát odpovědi. Podle kódu zjistíme, jak úspěšný byl náš požadavek. Kódy jsou standardní podle "http status codes".

- **2xx** Všechno dopadlo dobře
- **4xx** Chyba je na straně klienta
- **5xx** Chyba je na straně serveru

V případě úspěchu (kód 200 - OK) se odesle odpověď na dotaz, nebo nic, pokud request neměl za funkci něco vrátit. V případě neúspěchu pak v odpovědi najdeme zprávu o chybě, která nastala. Obvykle to u kódu 400 bývá, že odesíláme duplicitní záznam.

Druhá část popisuje strukturu schémat jednotlivých modelů. Jelikož databáze má datové formáty, zatímco JSON soubor ne (nebo alespoň ne tak rozsáhlý) musí i odesílaná data mít správný formát, nebo alespoň být validní po automatickém přetypování. Schema tvoří JSON objekt popisující schema. Pokud je datový typ položky objekt, buď se opravdu jedná o objekt, nebo se jedná pouze o upřesnění datového typu. Klíčovými slovy jsou:

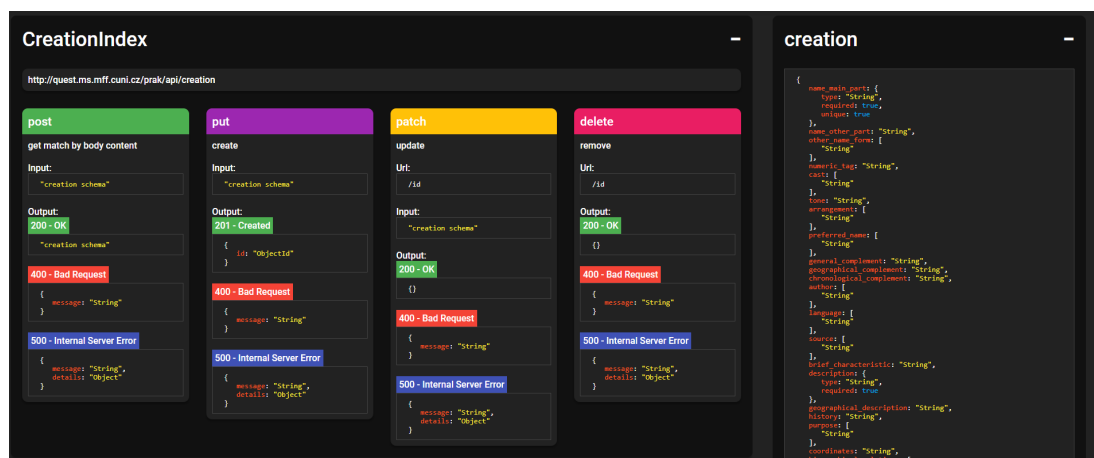
- **type**: datový typ
- **required**: true, pokud je tato položka povinná
- **unique**: true, pokud se zadaná hodnota, nesmí schodovat s již existující položkou v DB
- **ref**: název schématu, na který se ID odkazuje
- **refPath**: speciální ref, umožňující uživateli zadat i název schématu, vůči kterému se odkazuje

Pokud je hodnota pouze string, pak je tato hodnota datovým typem.

6.1.1 software na online dokumentaci

Pro možnost stážení dokumentace a prohlížení offline, je vše zabaleno do jednoho html souboru. Uvnitř je zdrojový kód programu, který vykresluje stránku a zároveň data dokumentace samotné.

Program vykresluje všechny položky s daty dokumentace. Bloky zdrojových kódů jsou vysazeny monospacem a obarveny, aby uživatelé poskytli rychlejší orientaci v kódu.



6.2 Backend

Na backendu je spuštěný express server (bezpečí pod nodejs), který zachytává requesty s url ...prak/api/... . podle cesty, jež je uvedena za /api se request předává příslušnému routeru. Vyjimku tvoří adresa .../api/documentation, která rovnou přeposílá soubor s dokumentací a není tedy pro získání dokumentace třeba rozumět systému hlouběji.

Při převzetí requestu jedním z mnoha routerů, se porovnává typ requestu (POST, PUT atd.) a případně detaily cesty v url. Při plné nalezení schody se provede overení práv pokud je potřeba. Pokud request má požadovaná oprávnění je provedena příslušná funkce a uživateli je vrácena odpověď / potvrzení o úspěchu. V případě že request nemá příslušná oprávnění, je vrácen kód 401. V případě že se na serveru něco pokazí vrátí se kód 400, nebo 500, podle typu problému.

6.3 API

dostupné na adrese quest.ms.mff.cuni.cz/prak/api/

6.3.1 Autentifikace

S každým requestem přichází v hlavičce i cookies. Pro autentifikaci používám cookie se jménem "sessionID". Podle něj se najde příslušný uživatel a porovnají se jeho předava a práva potřebná pro vykonání funkce. Pokud jsou práva nedostatečná, vrátí se odpověď 401, v případě správného oprávnění, router vykoná

funkci jez danemu requestu prislusi a pokud se nepokazi nic jineho, vrati validni odpoved.

6.4 Frontend a volani API

Jelikoz je cely projek zamyslen jako webova aplikace, nejstandartnejsi pouziti je volani API pomoci JS funkce `fetch()`, coz je pouze zastita pro XMLHttpRequest. Ale je mozne jej volat jakkoliv jinak, dokud to bude validni http request.

6.4.1 Fetch

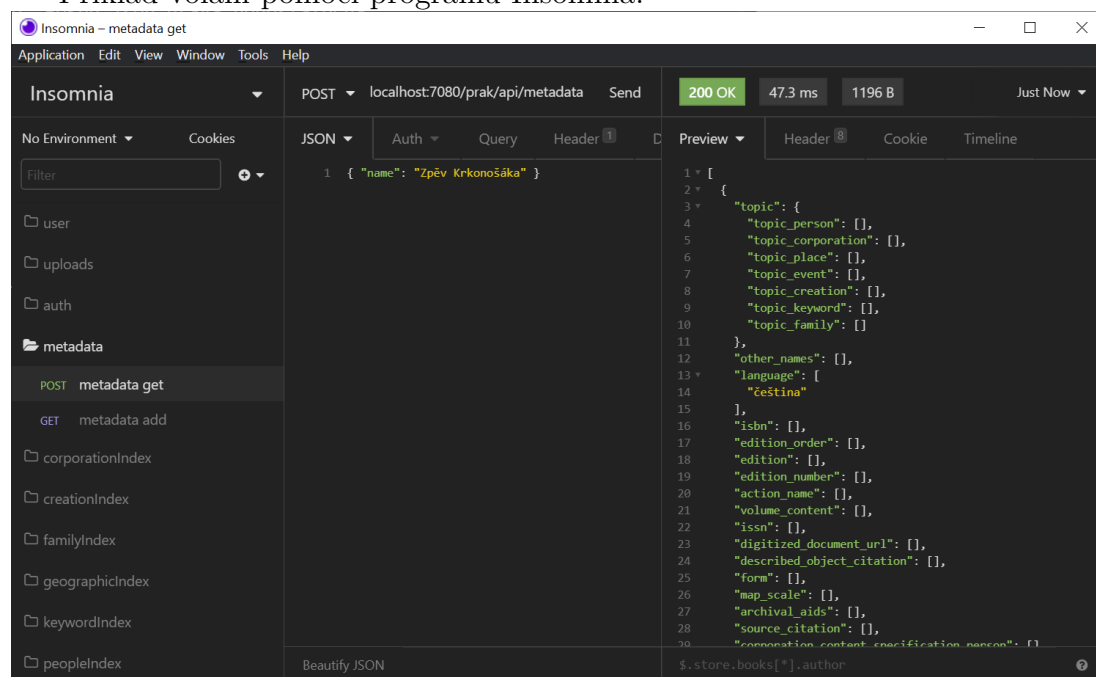
Dokumentace metody: mozilla

Priklad pouziti:

```
const url = "/prak/api/metadata"
fetch(url, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ "name": "... " }),
})
.then(response => {
  if(!response.ok) throw response
  return response.json()
})
.then(response => {
  console.log(response)
})
.catch(error => {
  console.error(error)
})
```

6.4.2 Existující programy pro práci a testování API

Příklad volání pomocí programu Insomnia:



7. Moduly

7.1 Přidávání nových modulů

7.2 Aktuálně nasazené moduly

7.2.1 Modul hologram

Načítání modulu

Tento modul obsahuje velkou knihovnu a tudíž není efektivní jej mít v primární aplikaci. Protože by se tyto knihovny načítaly, i když by uživatel s tímto modulem neměl v plánu pracovat. Což většinu času nebude chtít a tudíž by to pouze vedlo ke zpomalení systému. Systémem LAZY načítání je tedy celý modul odříznut a načítá se až explicitně při jeho použití.

Grafický engine

Pro vykreslování 3D modelu na webu slouží WebGL, grafická knihovna podobná staré verzi OpenGL. Reálně jsou dostupné dvě propracované knihovny pro práci ve WebGL. ThreeJS, které se zaměřuje na statické vykreslování a předvádění. BabylonJS, která naopak napodobuje známe programy jako unity a má podporu fyziky a dalších podknihoven užitečných pro tvorbu her. Pokud pomineme možnost naprogramovat vše od základu, nejvhodnější knihovnou je **ThreeJS**. Z této knihovny využijeme moduly pro načítání modelu a textury, jejich spárování a efektu Peppers Ghost, který je implementován pomocí 4 nezávislých kamer.

Peppers Ghost Effect



source: Wikipedia

Jedná se divadelní trik, jež skládá pozorovateli dva obrazy přes sebe a vytváří iluzi hologramu nebo ducha. Původně byl vyvinut pro divadelní představení, v moderní době však dokáže simulovat i hologram, který známe ze sci-fi.

Modely a textury

Modely jsou z webu <https://3dwarehouse.sketchup.com> a textury z webu <https://www.textures.com/> a <https://texturehaven.com/>.

V programu blender pak byly tyto textury namapovány na objekt a bylo do nich zapéčeno ambient occlusion (stíny vytvořené stísněným prostorem).

8. instalace a spusteni

9. Reseni

9.1 výsledny web

9.2 uživatelska dokumentace

Závěr

Seznam použité literatury

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A. Přílohy

A.1 První příloha