



**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, katedra

Abstrakt: TODO Abstrakt cz

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, department

Abstract: TODO Abstrakt en

Keywords: digital archive web database

Obsah

1	zadani	4
2	existujici produkty	5
2.1	KOHA	5
2.2	Evergreen	5
2.3	SLIMS	5
2.4	Zaverem	5
3	Vyber technologii	6
3.1	Frontend	6
3.1.1	Single page aplicacni	6
3.1.2	React	6
3.1.3	Dalsi mozne technologie	6
3.2	Backend	6
3.2.1	Express.js	7
3.2.2	MongoDB	7
3.2.3	Dalsi mozne technologie	7
4	diagram systemu	8
5	implementace backendu	9
5.1	server	9
5.2	knihovny	9
5.2.1	Express.js	9
5.2.2	mongoose	9
5.3	dokumentace	9
5.3.1	vlastni mini knihovna pro dokumentaci	9
5.4	routes	9
5.4.1	uzivatel	9
5.4.2	autorizace	9
5.4.3	zaznam	9
5.4.4	stranka	9
5.4.5	nahravani souboru	9
5.5	models	9
5.5.1	zaznam	9
5.5.2	stranka	9
5.5.3	uzivatel	9
6	implementace frontendu	10
6.1	server	11
6.1.1	kompile	11
6.1.2	npm	11
6.2	knihovny	11
6.2.1	React	11
6.2.2	Material-ui	11
6.2.3	il8n	11

6.2.4	babel	11
6.2.5	webpack	11
6.3	rozhraní	11
6.3.1	sceny	11
6.3.2	komponenty	11
6.3.3	moduly	11
6.4	Lokalizace	11
7	moduly	12
8	Provazani Backendu a Frontendu, API	13
8.1	Dokumentace online	13
8.1.1	sowtware na online dokumentaci	14
8.2	Backend	14
8.3	API	14
8.3.1	Autentifikace	14
8.4	Frontend a volani API	15
8.4.1	Fetch	15
8.4.2	Existující programy pro práci a testování API	16
9	instalace a spuštění	17
10	výsledný web	18
11	využití	19
	Závěr	20
	Seznam použité literatury	21
	Seznam obrázků	22
	Seznam tabulek	23
	Seznam použitých zkratk	24
A	Přílohy	25
A.1	První příloha	25

Úvod

1. zadani

Cilem tohoto projektu je vytvoreni webového rozhrani pro ukladani a zobrazovani historickych i soucasnych zaznamu z oblasti Krkonos.

2. existující produkty

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

2.1 KOHA

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

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

Používá SQL tabulky. Psána v Perlu, na frontendu využívající javascript (ale není ho tolik).

2.2 Evergreen

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

Další z řady knihovnických systémů. Používán např. Pedagogickou a Teologickou fakultou v Praze.

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

2.3 SLIMS

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

Systém se základní funkcionalitou a přívětivým vzhledem. Není v cestě. Není primárně určen jako knihovnický systém, spíše je to univerzální systém na jakýkoliv systém, proto není až tak efektivní a jeho nastavování by zabralo mnoho času.

2.4 Závěrem

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

3. Vyber technologii

3.1 Frontend

Vzhledem k rychle se menicim trendum v oblasti webovych technologii, jsem se rozhodl jit cestou kterou vyvynul Facebook a jeho tym programatoru. Jedna se o technologii **Single page application**, jez je implementovana v knihovne **React**.

3.1.1 Single page application

Single page application je technologie umoznujici vykresleni jine stranky, bez nutnosti posilani requestu na server. Uzivatel si pri prvnim spusteni webu stahne cely balicek webu a pri opetovnem nascteni vetsinou saha jen do sve cache. Javascriptova knihovna (v tomto pripade React) pote stranku prekresluje pri uzivatelske interakci. V pripade nutnosti stazeni / posilani dat mezi serverem a uzivatelem (napr. editace zaznamu, nebo nacteni existujiciho zaznamu) se vola pouze request k API webové služby a telo requestu obsahuje pouze uzitecne informace.

3.1.2 React

Knihovna React je knihovna poskytujici single page application technologii. Jedna se o dobre udrzovanou knihovnu, jez byla vyvinuta Facebookem, jakozto nahrada zastaraleho konceptu renderovani stranky na serveru. Diky tomu servery nemuseli ztratet vykon s kazdou zmenou na strance a vykon k renderovani se bere z PC uzivatele. Jadro teto knihovny je velmi dobre optimalizovate a poskytuje i radu debugovacich nastroju, coz je pro vetsi projekty nepostradatelna vyhoda.

3.1.3 Dalsi mozne technologie

Velmi casto vykreslovani stranek probiha na serveru, se systemy jako jsou WordPress, psany PHP. Takovyto system je velmi dobre uzivatelsky privetivy, ale z pohledu vykonu ma velmi obrovsky overhead. V pripade implementace knihovniho systemu by to znamenalo vykreslovat celou stranku (hlavicku, telo i zapati) na serveru, na druhe strane single page application nic nerenderuje, pouze posle informaci o knize.

3.2 Backend

Mit single page aplikaci na frontendu znamena, ze na backendu musi existovat API, od ktereho bude frontend cerpat data. Navic zde potrebujeme i system pro staticke odesilani baliku cele webové stránky. V ramci udrzitelnosti jsem se rozhodl vyuzit jazyk Javascript stejny jako pro frontend. Express.js je knihovna ktera umoznuje komplexni spravu requestu a stala se tudiz jasnou volbou.

3.2.1 Express.js

Express.js poskytuje odesilani statickych stranek (Reactiho baliku v nasem pripade), custom requesty pro rozmanite API a take odesilani a lokalni ukladani statickych souboru, jako obrazku, word i pdf dokumentu atd.

3.2.2 MongoDB

MongoDB je databazovy system typu non-sql. Coz primarne znamena, ze data neuchovava v tabulkach, ale v tkz. schematech. Coz ma mnoho vyhod, nejvetsi je, ze nekompletni zaznamy nezabiraji svymi nevyplnenymi daty misto v DB a uklada se opravdu jen to co je potreba. Dalsi vyhodou, je styl ukladani dat a komunikace s DB. Databaze si data uchovava ve formatu BSON (binarni JSON roz sireny o datove typy). O data si aplikace zada pomoci query, ktera je zcela odlišna od tech u sql-like databazi, primarne se zde neposila query ve formatu string ale JSON, diky cemuz napr. nenastane znama SQL injection. Znovu ve formatu JSON pote data vraci aplikaci.

3.2.3 Dalsi mozne technologie

Diky oddeleni frontendu a backendu (narozdil napr. u WordPressu) je mozne na backend nasadit temer cokoliv co umi posilat requesty. Prikladem tomu muzou byt scripty v jazicich PHP, C#, Python, nebo Perl. Ale vzhledem k tomu, ze jednim z modulu bude neuronova sit na vyhledavani, vybiral jsem mezi Pythonem a JavaScriptem, jakozto 2mi jazyky, ktere maji velmi dobre knihovny pro praci s neuronovymi sitemi.

4. diagram systemu

5. implementace backendu

5.1 server

5.2 knihovny

5.2.1 Express.js

nadstavba pro nahravani souboru

5.2.2 mongoose

5.3 dokumentace

/api/documentation

5.3.1 vlastni mini knihovna pro dokumentaci

5.4 routes

5.4.1 uzivatel

5.4.2 autorizace

5.4.3 zaznam

5.4.4 stranka

5.4.5 nahravani souboru

5.5 models

5.5.1 zaznam

5.5.2 stranka

5.5.3 uzivatel

6. implementace frontendu

6.1 server

6.1.1 kompilace

6.1.2 npm

6.2 knihovny

6.2.1 React

6.2.2 Material-ui

6.2.3 i18n

6.2.4 babel

6.2.5 webpack

6.3 rozhrani

6.3.1 sceny

amin

cms

homepage

page

login

kontakt

search

show

edit

6.3.2 komponenty

KomboBox

Zapati

Navigacni menu

validationTextField

Uploadfile

Indexy

6.3.3 moduly

hologram

6.4 Lokalizace

7. moduly

8. Provazani Backendu a Frontendu, API

8.1 Dokumentace online

Aktualni dokumentace, tak aby mohla byt casem aktualizovana a mohly do ni byt pripisovany dalsi veci, se nachazi na webu quest.ms.mff.cuni.cz/prak/api/documentation. Dokumentace je rozdelena na dve casti.

Prvni cast popisuje volani API. Kazda metoda (GET, POST atd.) ma svuj vlastni ucel jez je popsán uvnitr, url, na kterou se dotazovat a mozne parametry. Spolu s formatem requestu je i zde format odpovedi. Podle kodu zjistime, jak uspesny byl nas pozadavek. Kody jsou standartni podle "http status codes".

- **2xx** Vsechno dopadlo dobre
- **4xx** Chyba je na strane klienta
- **5xx** Chyba je na strane serveru

V pripade uspechu (kod 200 - OK) se odesle odpoved na dotaz, nebo nic pokud request nemel za funkci neco vracet. V pripade neuspechu pak v odpovedi najdeme zpravu o chybe která nastala. Obvykle to u kodu 400 byva, ze odesilame duplicitni zaznam.

Druha cast popisuje strukturu schemat jednotlivych modelu. Jelikož databaze ma datove formaty, zatimco JSON soubor ne (nebo alespon ne tak rozsahle) musi i odesilana data mit spravny format, nebo alespon byt validni po automatickem pretypovani. Schema tvori JSON objekt popisujici schema. pokud je datovy typ polozky objekt, bud se opravdu jedna o objekt, nebo se jedna pouze o upresneni datoveho typu. Klicovimy slovy jsou:

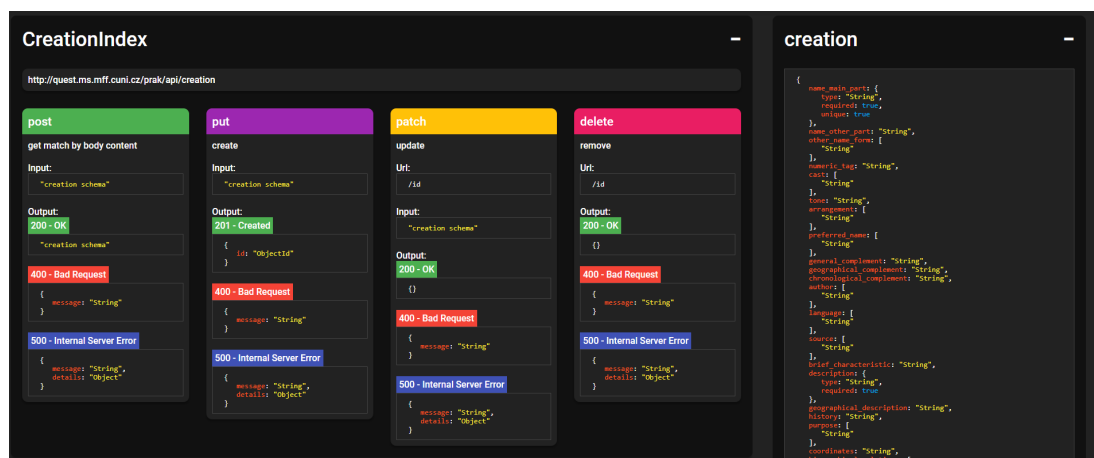
- **type**: datovy typ
- **required**: true, pokud je tato polozka povinna
- **unique**: true, pokud se zadana hodnota, nesmi schodovat s jiz existujici polozkou v DB
- **ref**: nazev schematu, na který se ID odkazuje
- **refPath**: specialni ref, umoznujici uzivateli zadat i nazev schematu, vuci kteremu se odkazuje

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

8.1.1 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.



8.2 Backend

Na backendu je spuštěný express server (běžící 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.

8.3 API

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

8.3.1 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 práva 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.

8.4 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.

8.4.1 Fetch

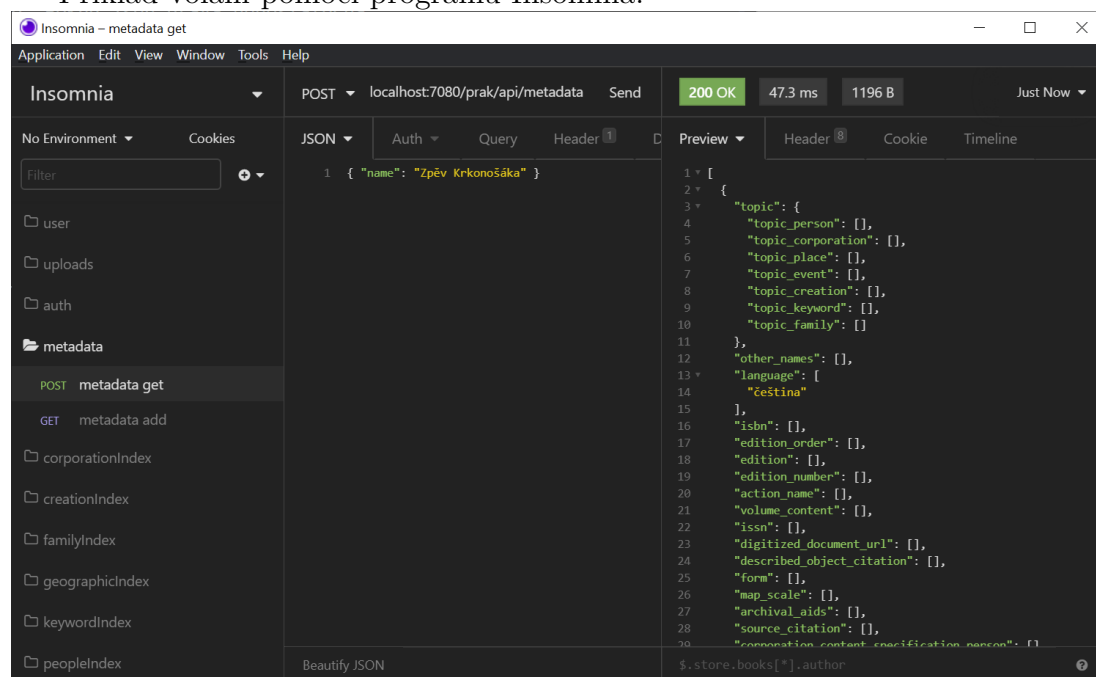
Dokumentace metody: mozilla

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)
})
```

8.4.2 Existující programy pro práci a testování API

Příklad volání pomocí programu Insomnia:



9. instalace a spusteni

10. vysledny web

11. využití

Závěr

Seznam použité literatury

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A. Přílohy

A.1 První příloha