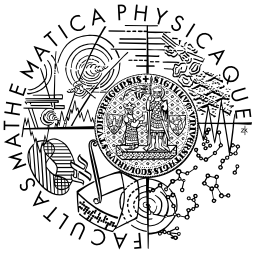


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

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 pouziti modernich webovych a databazovych technologii modernizujeme zastarale knihovni systemy. Snizime naroky na vykon serveru a sitoveho rozhrani. Uzivateli techto systemu poskytneme intuitivni a rychlou webovou aplikaci. Navstevnikovi pak poskytneme nejen onen knihovni system, ale cely balicek ruznych objektuod novinek az po 3D modely historickych objektu. Vyvojari a automatizovane systemy budou mit pristup k API, jez bude poskytovat uchovavana data. Zaroven bude slouzit i jako prostredek pri komunikaci modulu systemu s databazi, napr pro umozneni vyhledavani pomoci neuronove site.

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

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 vytvořeni webového rozhraní pro ukládaní a zobrazovaní historických i současných záznamů z oblasti Krkonoš.

2. Analýza a požadavky

Vybíráme jen z open source produktů, abychom mohli nahlédnout do jejich kódu a lepe 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 systému. Používán např Pedagogickou a Teologickou Fakultou v Praze.
Používá sql databázi, vykreslovaní probíhá na serveru a nemá uživatelsky přívětivé prostředí.

2.3.3 SLIMS

url: https://slims.web.id/
Systém se základní funkcionalitou a přivětivým vzhledem. Není v češtině. Není primárně určen jako knihovní systém, spíš je to univerzální systém na cokoliv, proto není až tak efektivní a jeho nastavovaní 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. Pri zachovaní stejné, možná i lepši 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 aplication**, jež je implementovaná např. v knihovně **React** zaštiťovanou společností Facebook.

Single page aplication

Single page aplication je technologie umožnující vykreslení jiné stránky, bez nutnosti posílaní requestu na server. Uživatel si při prvním spuštěni webu stáhne celý baliček webu a při opětovném načtení vetšinou sahá jen do své lokální cache. Javascriptová knihovna (v tomto případě React) poté stránku překresluje při uživatelské interakci. V případě nutnosti staženi / posílaní 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 aplication technologii. Jedná se o dobře udržovanou knihovnu, jež byla vyvinuta Facebookem, jakožto náhrada zastaralého konceptu renderovaní stránky na serveru. Díky tomu servery nemusejí ztrácet výkon s každou změnou na stránce a výkon k renderovaní se bere z PC uživatele. Jádro této knihovny je velmi dobře optimalizované a poskytuje i řadu debutováních nástrojů, což je pro vetší projekty nepostradatelná výhoda.

Dalsi mozne technologie

Velmi často vykreslovaní stránek probíhá na serveru, se systémy jako jsou WordPress, psaný v PHP. Takovýto system je velmi dobře uživatelsky přivětivý, ale z pohledu výkonu má velmi obrovský overhead. V případě implementace knihovního systému by to znamenalo vykreslovat celou stránku (hlavičku, tělo i zápatí) na serveru, na druhé straně single page aplication 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 odesílaní balíku celé webové stránky. V rámci udržitelnosti jsem se rozhodl využit 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 odesílaní statických stránek (Reactího balíku v našem případe), custom requesty pro rozmanité API a také odesílaní a lokální ukládaní statických souborů, jako obrázků, wordovských i pdf dokumentů atd.

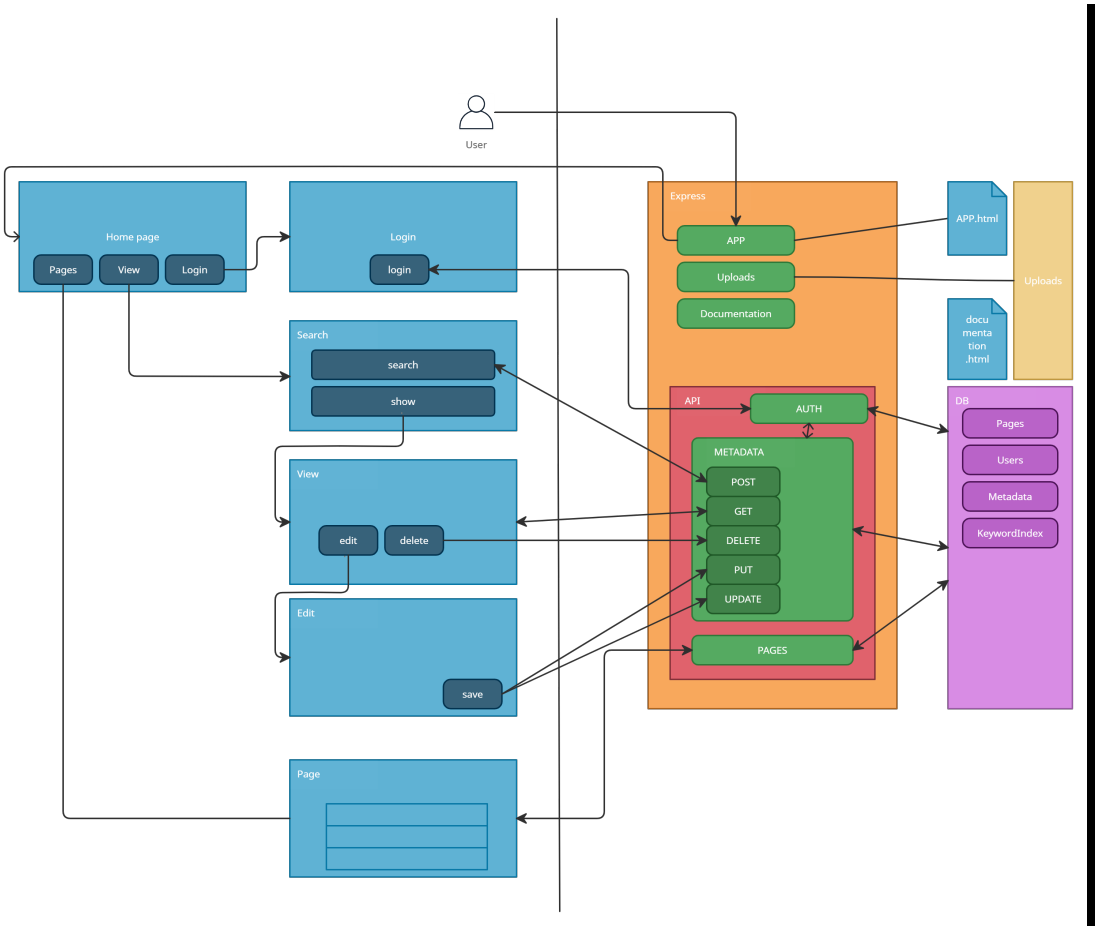
MongoDB

MongoDB je databázový system 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 ukládaní dat a komunikace s DB. Databáze si data uchovává ve formátu BSON (binární JSON rozšířený o datové typy). O data si aplikace žádá pomoci 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 vrací aplikaci.

Další možné technologie

Díky odděleni frontendu a backendu (narozdil např. u WordPressu) je možné na backend nasadit téměř cokoliv co umi posílat requesty. Příkladem tomu můžou byt scripty v jazycích PHP, C#, Python, nebo Perl. Ale vzhledem k tomu, že jedním z modulů bude neuronová síť na pokročilé vyhledávaní, 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 přijímaní požadavku od klienta a odesílaní 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 requesty potřebnými pro fungovaní 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 šifrovaní hesel a hašovaní 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 nahravaní 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

/api/documentation

4.3.1 Vlastní mini knihovna pro dokumentaci

4.4 Routes

4.4.1 Uživatel

4.4.2 Autorizace

4.4.3 Záznam

4.4.4 Stránka

4.4.5 Nahrávaní souborů

4.5 Modely

4.5.1 Záznam

4.5.2 Stránka

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

Zobrazeni stránky

Přihlašování

Kontaktní formulář

Vyhledávátko

Zobrazovávátko

Upravovátko

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. Provázaní Backendu a Frontendu, API

6.1 Dokumentace online

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

Prvni cast popisuje volani API. Kazda metoda (GET, POST atd.) ma svuj vlastni ucel jez je popsan 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 ktera nastala. Obvykle to u kodu 400 byva, ze odesilame duplicitni zaznam.

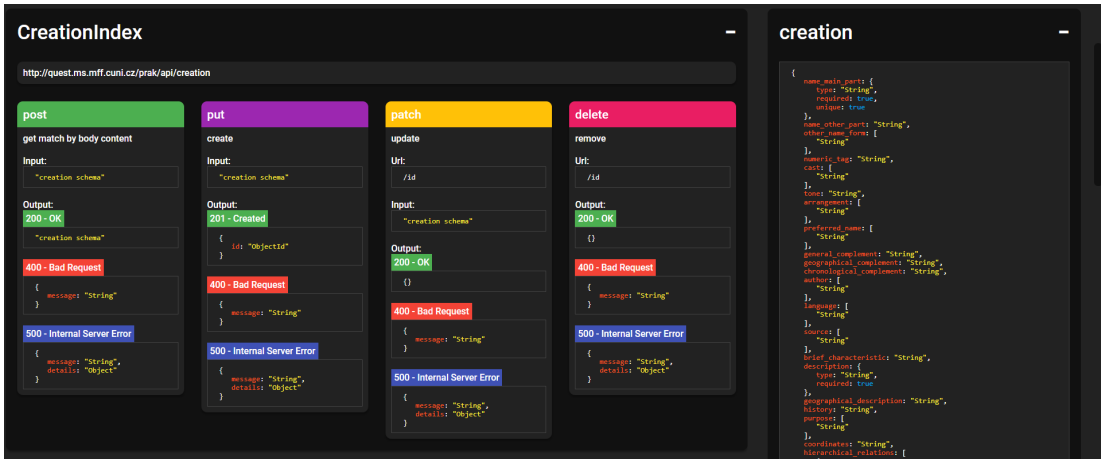
Druha cast popisuje strukturu schemat jednotlivych modelu
Jelikoz databaze ma datove formaty, zatimco JSON soubor ne (nebo alespon ne tak rozsahle) musi i odesilana data mit spravny format, nebo alespon byt validni po automatickem pretypovani. Schema tvori JSON objekt popisujici schema. pokud je datovy 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 ktery 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.

6.1.1 sowtware na online dokumentaci

Pro moznost stazeni dokumentace a prohlizeni offline, je vse zabaleno do jednoho html souboru. Uvnitr je zdrojovy kod programu, jez vykresluje stranku a zaroven data dokumentace samotne.
Program vykresluje veskere polozky s daty dokumentace. Bloky zdrojovych kodu jsou vysazeny monospacem a obarveny, aby uzivateli poskytly rychlejsi orientaci v kodu.



6.2 Backend

Na backendu je spusteny express server (bezici pod nodejs), ktery zachytava requesty s url ...prak/api/... . podle cesty, jez je uvedena za /api se request predava prislosnemu routeru. Vyjimku tvori adresa .../api/documentation, ktera rovnou preposila soubor s dokumentaci a neni tedy pro ziskani dokumentace trba rozumet systemu hloubeji.

Pri prevzeti requestu jednim z mnoha routeru, se porovnava typ requestu (POST, PUT atd.) a pripadne detaily cesty v url. Pri plne nalezeni schody se provede overeni prav pokud je potreba. Pokud request ma pozadovana opravneni je provedena prislusna funkce a uzivateli je vracena odpoved / potvrzeni o uspechu. V pripade ze request nema prislusna opravneni, je vracen kod 401. V pripade ze se na serveru neco pokazi vrati se kod 400, nebo 500, podle typu problemu.

6.3 API

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

6.3.1 Autentifikace

S kazym requestem prichazi v hlavicce i cookies. Pro autentifikaci pouzivam cookie se jmenem "sessionID". Podle nej se najde prislusny uzivatel a porovnaji se jeho predava a prava potrebna pro vykonani funkce. Pokud jsou prava nedostatecna, vrati se odpoved 401, v pripade spravneho opravneni, router vykona

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.

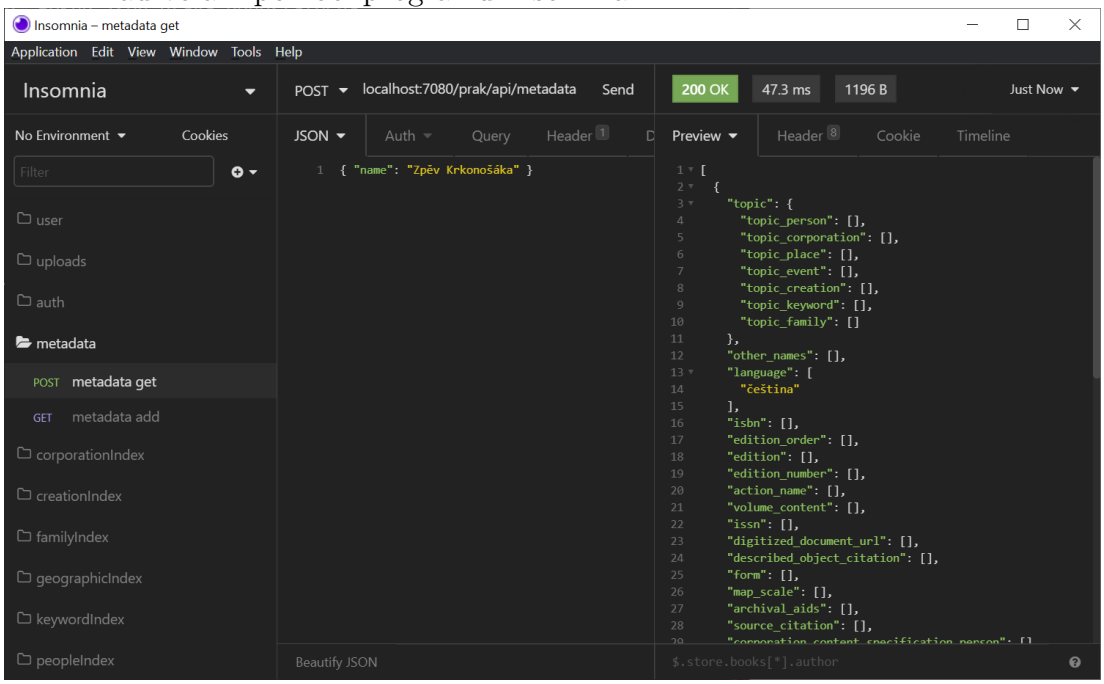
6.4.1 Fetch

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 Existujici programy pro praci a testovani APi

Priklad volani pomoci programu Insomnia:



7. Moduly

7.1 obecne pridavani modulu

7.2 aktualni moduly

7.2.1 hologram

nacitani modulu

Tento modul obsahuje velkou knihovnu a tudiz neni efektivni jej mit v defaultni aplikaci. Protoze by se tyto knihovny nacitaly, i kdyz by uzivatel s timto modulem nemel v planu pracovat. Coz vetsinu casu nebude chtit a tudiz by to pouze vedlo k zpomaleni systemu. Systemem LAZY loading je tedy cely modul odriznut a nacita se az explicitne pri jeho pouziti.

8. instalace a spusteni

9. Reseni

9.1 vysledny web

9.2 uzivatelska dokumentace

Závěr

Seznam použité literatury

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A. Přílohy

A.1 První příloha