Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Webová aplikace pro sběratele za použití Vue.js frameworku

Bakalářská práce

Autor: Michal, Dolenský
Studijní obor: Aplikovaná Informatika

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové leden 2022

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 15.11.2021 Michal Dolenský

Poděkování:

Děkuji vedoucí bakalářské práce Mgr. Daniele Ponce, Ph.D. za ochotu, trpělivost, cenné rady a metodické vedení práce.

Anotace

Cílem této bakalářské práce je v teoretické části představit základní pojmy, následně framework Vue.js a analyzovat jeho popularitu s dalšími hlavními frameworky na trhu. Dále se práce zabývá popisem a analýzou existujících řešení pro sběratele. Z těchto existujících řešení (zejména z jejich nedostatků) a poznatků od sběratele je vytvořen seznam požadavků a návrhy částí aplikace. Praktická část se práce zabývá implementací navrhnuté aplikace pro sběratele předmětů ve formě jednostránkové aplikace za použití frameworku Vue.js. Výstupem je zhodnocení frameworku Vue.js na základě zkušeností autora při realizaci zmíněné aplikace. Na straně serveru byl použit framework Nest.js společně s TypeORM a databází PostgreSQL. Komunikace mezi klientem a serverem je zajištěna pomocí GraphQL a REST API.

Annotation

Title: Web application for collectors using Vue.js framework.

The aim of this bachelor thesis is to introduce the basic concepts in the theoretical part, then the Vue.js framework and analyze its popularity with other major frameworks on the market. Furthermore, the thesis deals with the description and analysis of existing solutions for collectors. From these existing solutions (especially from their shortcomings) and insights from the collector, a list of requirements and suggestions for parts of the application are made. The practical part of the thesis deals with the implementation of the proposed application for object collectors in the form of a single page application using the Vue.js framework. The output is an evaluation of the Vue.js framework based on the author's experience in implementing said application. On the server side, the Nest.js framework was used along with TypeORM and PostgreSQL database. Communication between the client and server is provided using GraphQL and REST API.

Obsah

Seznam obrázků

Seznam grafů

Seznam ukázek kódů

Seznam použitých zkratek

API - Application Programming Interface

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

IDE – Integrated Development Environment

JSON - JavaScript Object Notation

ORM - Object–relational mapping

REST - Representational State Transfer

SEO – Search engine optimization

SPA – Single Page Application

XML - Extensible Markup Language

1 Úvod

Sběratelství je velmi rozšířený koníček na světě, který se zaměřuje na sbírání předmětů a jejich organizací do sbírek.
Tyto sbírky s nástupem moderních technologií lze převést do moderní podoby a umožnit sběratelům svoje předměty vystavit na internetu pomocí vytvoření vlastního katalogu. Těchto řešení je poskrovnu a mají své nedostatky.

Tato práce se zabývá vytvoření ukázkové aplikace pro sběratele na základě požadavků cílového uživatele a analýzy existujících řešení. V rámci práce je vypracován návrh vlastního řešení, následně popsána a zhodnocena implementace za použitím JavaScriptového frameworku Vue.js a Node.js frameworku Nest.js.

2 Cíl práce

Cílem práce je experimentálně vyhodnotit Vue.js framework, pro tento účel bude vypracována ukázková aplikace.

Aplikace bude vytvořena na základu požadavků od uživatele a analýzy existujících řešení (zejména jejich nedostatků). Součástí vypracování je práce i návrh a implementace aplikace, včetně popisu vybraných technologií a řešení, která umožní uživatelům, sběratelům zakládat si své předměty svých sbírek.

Práce je uzavřena zhodnocením frameworku Vue.js na základě zkušeností autora získaných při realizaci této aplikace.

3 Vymezení pojmů

V této kapitole budou vymezeny pojmy používané později v práci.

3.1 DOM

DOM (Document Object Model) je programovací rozhraní, které definuje logickou strukturu HTML a XML dokumentů, které jsou využívány prohlížečem pro zobrazení stránek. [1]

3.2 Virtuální DOM

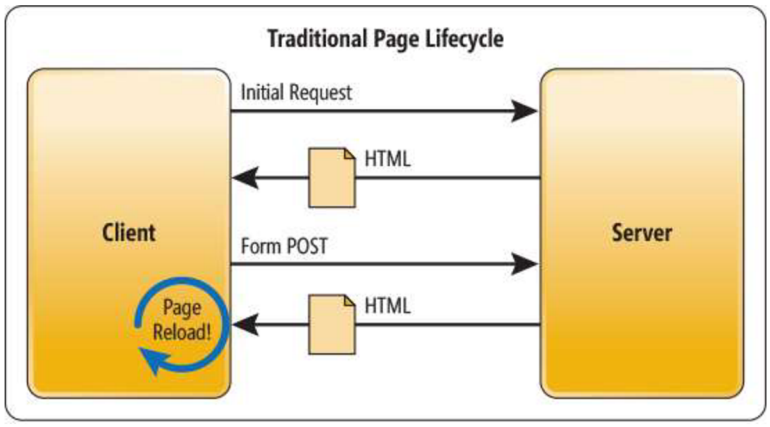
Virtuální DOM je kopie DOM udržována v paměti za účelem zvýšení výkonu zobrazení, jelikož použití JavaScriptu k vyhledávání v DOM je náročné na výpočet. Je reprezentován jednoduchým JavaScriptovým objektem, který je vytvořený funkcí vykreslení.
V případě nutnosti úpravy DOM objektu se nejprve provedou všechny změny na Virtuální DOM a následně se provede porovnání mezi reálným a Virtuálním DOM objektem a poté se aktualizují pouze změněná pole. [2], [3]

3.3 Jednostránková aplikace (Single-page application)

Jednostránková aplikace (SPA) je webová aplikace, která je spuštěna v prohlížeči. Pro svůj chod vyžaduje načtení, jak už název vypovídá, pouze jedné HTML stránky. O aktualizaci obsahu se pak stará JavaScript, který po požadavku uživatele vyšle požadavek na server, následně zpracuje obdržená data a výsledek zobrazí klientovi bez nutnosti znovunačtení stránky. [4]

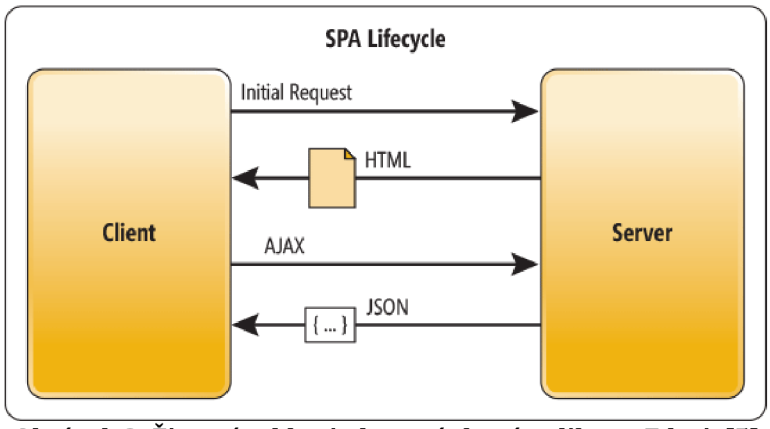
3.3.1 Porovnání tradiční vícestránkové a jednostránkové webové aplikace

Cyklus tradiční více stránkové webové stránky začíná požadavkem klienta na server o HTML dokument, který je následně serverem zprostředkován a prohlížeč následně podle tohoto dokumentu aktualizuje obsah. V případě, že klient si vyžádá další stránku nebo zašle formulář na server, tak mu bude vrácen zpracovaný dokument pouze v HTML. [5]



Obrázek 1: Životní cyklus tradiční aplikace. Zdroj: [5]

Cyklus jednostránkové aplikace začíná opět požadavkem o HTML dokument, který server následně poskytne. Další obsah je poté dynamicky poskytován na základě volání AJAX. Vrácená data ze serveru jsou obvykle ve formátu JSON, nebo XML který JavaScript zpracuje a aktualizuje podle nich stránku. [5]



Obrázek 2: Životní cyklus jednostránkové aplikace. Zdroj: [5]

3.3.2 Výhody jednostránkových aplikací

Výhody jednostránkových aplikací jsou následující:

- *Rychlejší odezva* – velká část zdrojů je načtena při prvním spuštění stránky.
- *Lepší uživatelský zážitek* – uživatel nemusí čekat na znovu načtení celé stránky a aktualizuje se pouze její část.

- *Snížené požadavky na server* – server se nemusí zabývat generováním obsahu pro zobrazení. Klient si po inicializaci vyměňuje pouze data, a nikoliv celou stránku s obsahem.
- *Snížení počtu přenesených dat mezi klientem a serverem* – jak už bylo výše zmíněno klient se serverem si vyměňuje pouze data. [6]

3.3.3 Nevýhody jednostránkových aplikací

Nevýhody jednostránkových aplikací jsou následující:

- *Optimalizace pro vyhledávání vyhledávačem (SEO)* - někteří vyhledávací roboti nemusí, nebo neumí spouštět JavaScript při prolézání jednotlivých webových stránek a následně si uloží do indexu pouze stránku s prázdným HTML tagem body. Světově nejpopulárnější vyhledávač Google s JavaScriptem umí pracovat a jeho robot GoogleBot zapíná JavaScript před indexací stránky [7] (bohužel nelze se na to spoléhat). Populární český vyhledávač Seznam.cz pravděpodobně indexaci obsahu s JavaScriptem neumí. [8]
- *Vyšší nároky na výkon zařízení klienta* – zpracování dat a následné vykreslování bylo přesunuto ze strany serveru na stranu klienta.
- *Klient musí mít povolený JavaScript v prohlížeči* – načtení a aktualizace obsahu stránky probíhá skrze něho. V případě, pokud klient nemá JavaScript povolený tak se mu zobrazí prázdná stránka.
- *Prvotní načtení může být pomalé* – při inicializaci musí prohlížeč načíst často velké JavaScriptové skripty, kde je uložena část, nebo celá aplikace. Po spuštění těchto skriptů si většinou musí vyžádat ze serveru data, ty zpracovat a následně zobrazit klientovi. [9]

4 Představení frameworku Vue.js

Následující kapitola se zabývá představením frameworku Vue.js a jeho oficiálního doplňku Vue Router. Dále bude srovnána popularita s ostatními JavaScriptovými frameworky React a Angular.

4.1 Úvod

Vue.js, nebo také Vue je progresivní JavaScriptový framework s otevřeným kódem v licenci MIT určený pro vytváření uživatelských rozhraní webových aplikací pomocí znovupoužitelných komponent. Vue bylo od začátku navrženo, aby ho šlo využít při začátku vývoje nové aplikace a také umožňuje postupné začlenění do již stávajících aplikací. Jeho jádro je zaměřenou pouze na zobrazovací vrstvu. [10] Samotné jádro lze rozšířit pomocí oficiálních knihoven jako jsou Vuex a Vue Router. S vývojem pomocí Vue aplikace může začít každý, díky jeho podrobné dokumentaci na oficiálních stránkách [10], kdo má základní až střední schopnosti HTML, CSS a JavaScriptu.

Vue je narozdíl od frameworků React.js od firmy Facebook a Angular (Google) vytvořen jedním člověkem Evan You a v současnosti ho s ním vyvíjí komunita vývojářů na jejich stránce Github. [11]

4.2 Historie

Při vytváření Vue se inspiroval Evan You při jeho práci v Google od frameworku Angular. Pro jeho práci potřeboval nástroj pro rychlé prototypování, který mu v té době Angular nedokázal poskytnout. Jeden důvodů bylo, že Angular ho nutil aplikovat struktury a koncepty, které ho zbytečně zdržovali.

Z tohoto důvodu se v roce 2013 rozhodl experimentovat a vytvořit framework, který by od Angularu převzal deklarativní datovou vazbu a mnoho dalších funkcí. Z toho pokusu vzniklo Vue.js. V září 2014 se podělil o Vue se světem na jeho Github účtu. [12]

Vue od této doby prošlo řadou vylepšení a aktuální verze je 3.

4.2.1 Vue 3

Vue ve verzi 3 bylo vydáno v září roku 2020. Verze 3 přinesla:

- Composition API, díky kterému lze lépe organizovat zdrojový kód.
- Zvýšení výkonu až o 55 % rychlejší vykreslení při prvotním spuštění, až o 133 % rychlejší při aktualizacích a snížila až o 54% využití paměti oproti Vue 2.
- Lepší podporu jazyka TypeScript, jelikož jeho zdrojový kód byl do něho přepsán.
- Byla vypuštěna podpora prohlížeče Internet Explorer 11, jelikož Vue 3 používá na pozadí z ECMAScript verze 2015 (ES6) Proxy API, které tento prohlížeč nepodporuje. [13]

4.3 Instance Vue

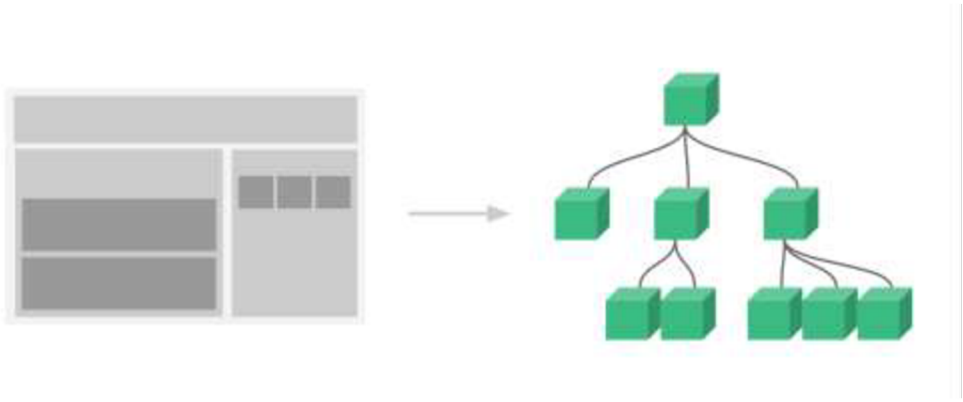
Instance Vue je srdcem celé aplikace. Vytvoříme jí zavoláním funkce *Vue.createApp()*. Poté lze definovat jednotlivé komponenty.

```
const app = Vue.createApp({})
```

Ukázka kódu 1: Vytvoření instance Vue. Zdroj: [Autor]

4.4 Komponenty

Vue aplikace je sestavena pomocí komponentního systému. Tento systém se skládá z komponent, které se dají opakovaně použít.



Obrázek 2: Komponentový systém. Zdroj: [14]

Jednotlivé komponenty lze definovat pomocí *app.component().* Tento způsob definice komponenty nejsou vyžadovány další nástroje a lze je přímo použít pro spuštění aplikace v prohlížeči.

Bohužel tento způsob definice příliš není vhodný pro střední a velké aplikace z důvodu nepřehlednosti, nemožnosti zobrazit zvýraznění syntaxe ve vývojovém prostředí a CSS styly jsou odděleny od komponent v externím souboru. Tyto problémy řeší Single File Components neboli jednosouborové komponenty. [10]

```
const app = Vue.createApp({})
app.component('hello-word', {
  template: `<h1>Hello World</h1>`
})
app.mount('#app')
```

Ukázka kódu 2: Vytvoření komponenty pomocí app.component(). Zdroj: [Autor]; Zpracováno dle: [10]

4.4.1 Single File Components

Single File Component (SFC) je technika, která umožňuje jednotlivě definovat komponenty v souboru s příponou *„.vue“.* Tento jeden soubor je přehledně rozdělen do tří sekcí pomocí tagů:

- <template> obsahující část s HTML,
- <script> obsahující část s JavaScript,
- <style> obsahující část s CSS. [15]

Tyto komponenty jsou následně zkompilovány pomocí například pomocí webpack [16] pluginu vue-loader [17], nebo nového nástroje od tvůrců Vue.js Vite [18].

```
<template>
    <p> {{greetings}} World! </p>
</template>

<script setup>
import { ref } from 'vue';
const greetings = ref("Hello");
</script>

<style scoped>
p{
    font-size:2em
    text-align: center;
}
</style>
```

Ukázka kódu 3: Ukázka struktury jednosouborové komponenty. Zdroj: [15]

4.4.2 Direktivy komponent

Direktivy jsou speciální atributy s předponou „v-“ tagů HTML, tyto atributy umožňují přidat komponentám další funkce, jako například: předání dat další komponentě, tvorbu listů za pomocí iterace polí, zobrazení/schování komponenty, reakci na eventy a vazbu dat.

4.4.2.1 Zobrazení textu z proměnné v-text a v-html

Pro zobrazení obsahu Vue využívá pro zobrazení surového textu direktiva v-text, nebo častější způsob vložení ve tvaru *{{nazev_promene}}* v těle. Vue tento obsah zobrazí jako text, nikoliv jako HTML. Pro tento účel slouží direktiva v-html, která vloží hodnotu proměnné, pomocí JavaScriptové funkce *innerHTML*, přímo do stránky.

```
<template>
<span v-text="html"></span> // <b>Hello World</b>
<span>{{html}}</span> // <b>Hello World</b>
<span v-html="html"></span> // Hello World
</template>
```

Ukázka kódu 4: Direktivy v-text a v-html. Zdroj: [Autor]

4.4.2.2 Podmíněné zobrazení v-if a v-show

Direktiva v-if umožňuje podmíněně zobrazit a schovat komponentu při zadání argumentu s true, nebo false. Pokud argument nesplňuje podmínku je možné zobrazit alternativní komponentu pomocí v-else, nebo v případě podmíněnou alternativu v-else-if. Neplatně podmíněná komponenta bude vymazána z těla HTML. Pro zobrazení, nebo schování komponenty je také možné použít direktivu v-show, která na rozdíl od direktivy v-if na dané komponentě aplikuje pouze CSS styl *display:none;*, který komponentu skryje uživateli, ale ve zdrojovém kódu HTML bude zobrazen nadále.

```
<h1 v-if="show">Zobrazí se pokud show bude true</h1>
<h1 v-else> Zobrazí se pokud show bude false </h1>

<h1 v-show="show">Bude schován pokud bude show true</h1>
```

Ukázka kódu 5: Direktivy v-if, v-else, v-show. Zdroj: [Autor]

4.4.2.3 Iterace proměnné v-for

Direktiva *v-for* se používá nejčastěji pro vytvoření seznamů, řádků tabulek a dalších opakujících komponent. Zde lze využít znovupoužitelnosti komponent a opakovat pole objektů. Tento objekt se dá poté vložit do komponenty pomocí direktiv *v-model*, nebo *v-bind*.

```
<ul v-for="todo in todos">
  <li>{{ todo.name }}</li>
</ul>
```

Ukázka kódu 6: Direktiva v-for. Zdroj: [Autor]

4.4.2.4 Předávání dat do komponenty v-bind

Pro předávání dat do komponent slouží direktiva v-bind, která lze zkrátit pomocí „:“. Umožňuje definovat například zdroj obrázku, nebo předat informaci další komponentě pomocí props.

```
<template>
  <img :src="imageSrc"/>
  <OtherComponent :text="textString">
</template>

<script setup>
import { ref } from 'vue'

const imageSrc = ref("example.com/image.jpg")
const textString = ref("Hello")
</script>

//OtherComponent
<template>
  <div>{{props.text}}</div>
</template>

<script setup>
const props = defineProps({
        text: String
    }
)
</script>
```

Ukázka kódu 7: Použití v-bind a ukázka předání dat do další komponenty pomocí props. Zdroj: [Autor]

4.4.2.5 Zpracování eventů v-on

Pro zpracování eventů z komponenty využívá Vue direktivu *v-on*, která sleduje, zda uživatel například stiskl tlačítko (v-on:click). V případě stisku zavolá metodu, která je definována jako její hodnota. Tuto direktivu lze také zkrátit pomocí @click.
Takto lze reagovat na změny hodnot formuláře, stisknutí klávesy a mnoho dalších JavaScriptových eventů.

```
<template>
    <button @click="buttonPressed">Press me</button>
</template>
<script setup>
const buttonPressed = ()=>{
  //obsah funkce
}
</script>
```

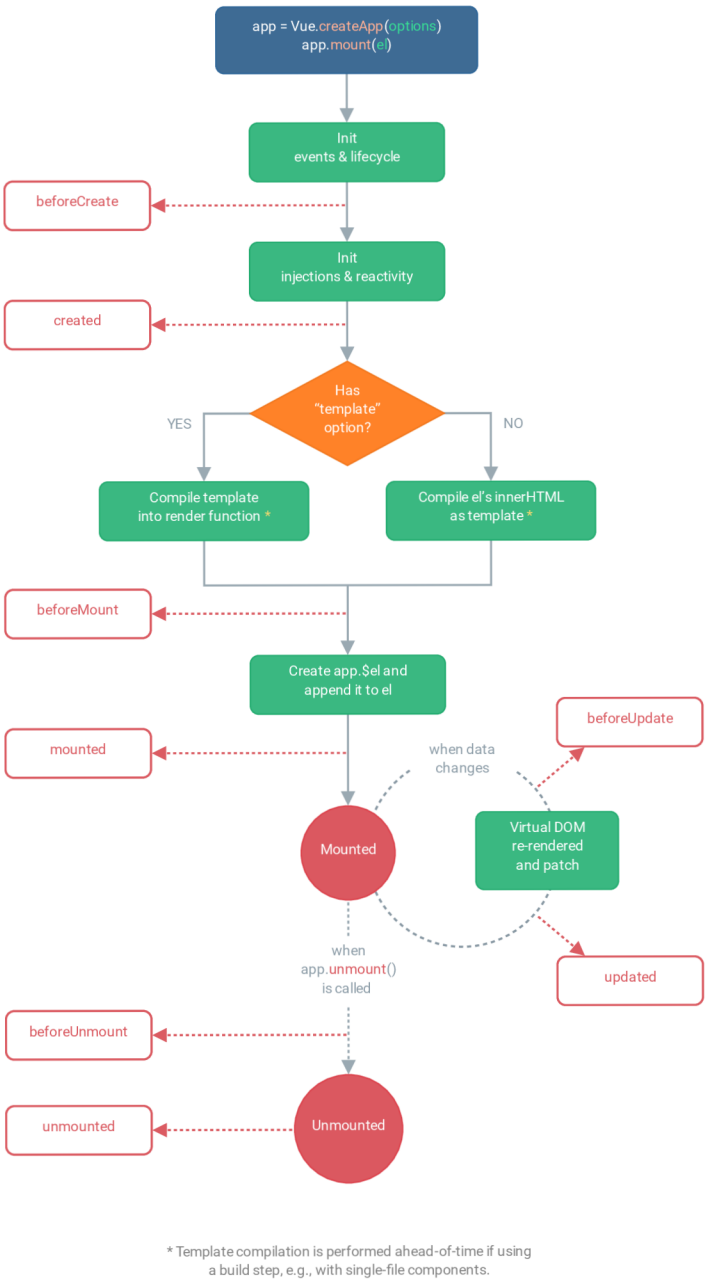
Ukázka kódu 8: Reakce na event. Zdroj: [Autor]

4.4.3 Životní cyklus komponenty

Při inicializaci nové komponenty se provede řada inicializačních metod.
Tyto metody se nazývají lifecycle hooks neboli háčky a jsou zavolány v určité fázi života komponenty od její inicializace, po její život, až po její zničení. [19]

Tyto háčky jsou volány v tomto pořadí:

1. *beforeCreate* – zavolána ihned po inicializaci instance, kdy ještě nebyly definovány eventy a pozorování dat
2. *created* – V této fázi bylo dokončeno zpracování nastavení. Připojení do DOM ještě nezačalo.
3. *beforeMount* – zavolána těsně před zahájeném připojení do DOM.
4. *mounted* – zavolána po vykreslení šablony do DOM.
5. *beforeUpdate* – zavolána před úpravou DOM při úpravě dat.
6. *updated* – zavolána po úspěšné změně a znovu vykreslení DOM.
7. *beforeUnmount* – zavolána těsně před odstraněním instance. Instance je v této fázi plně funkční.
8. *unmounted* – zavolána po zničení instance. V tomto okamžiku jsou všichni data, potomci, eventy, dané instance zničeny a instance byla odpojena.



Obrázek 3: Životní cyklus instance Vue. Zdroj: [19]

4.5 Vue Router

Vue Router je oficiální plugin pro Vue. Je hluboce integrován s jádrem Vue a umožňuje vytvářet jednostránkové aplikace. Záměrem tohoto pluginu je umožnit plynulý přechod, pomocí tras, mezi jednotlivými stránkami aplikace bez nutnosti znovunačtení stránky ze serveru. K tomu využívá dvou komponent router-link a router-view. Komponenta router-link funguje jako „*a* tag“ v HTML. Pomocí této komponenty lze vytvořit interní odkaz, který změní adresu URL bez nutnosti znovunačtení stránky. Komponenta router-view zobrazí komponentu podle aktuální URL adresy definované při konfiguraci routeru. [20]

4.5.1 Inicializace a konfigurace tras routeru

Pro přidání pluginu routeru do aplikace využijeme funkci *use(router)*, kde do parametru zadáme výsledek importované funkce *createRouter()*.

```
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

createApp(App)
  .use(router)
  .mount('#app');
```

Ukázka kódu 9: Příklad přidání routeru do Vue aplikace. Zdroj: [Autor]

4.5.2 Definice cest

Jednotlivé cesty jsou definovány pomocí pole *routes* jsou definovány objekty s hodnotami:

- *path* – obsahuje část, nebo celou URL adresu,
- *name* – jednotlivé trasy lze pojmenovat pro jednodušší přechod
- *component* – odkaz na danou komponentu, která bude zobrazena pod rodičovskou komponentou. Tuto komponentu lze také načíst za běhu aplikace přes funkci *import()* pro snížení času nutného k načtení a snížení požadavků na klienta.

Vue Router také umožňuje vytvářet dynamické trasování pomocí přidání *:nazev_promene* do parametru *path.*

```
import { createRouter, createWebHashHistory } from 'vue-router';
import Home from '../views/Home.vue';
import NotFound from '../views/NotFound.vue';
import Page from '../views/Page.vue';

const router = createRouter({
  history: createWebHashHistory(),
  routes:[
    { path: '/', name: 'Home', component: Home, },
    {
      path: '/about',
      name: 'About',
      component: () => import('../views/About.vue'),
    },
    { path: '/page/:pageId', name: 'Page', component: Page, },
    { path: '/:pathMatch(.*)*', name: 'NotFound', component: NotFound, },
  ],
});

export default router;
```

Ukázka kódu 10: Konfigurace Vue routeru. Zdroj: [Autor]

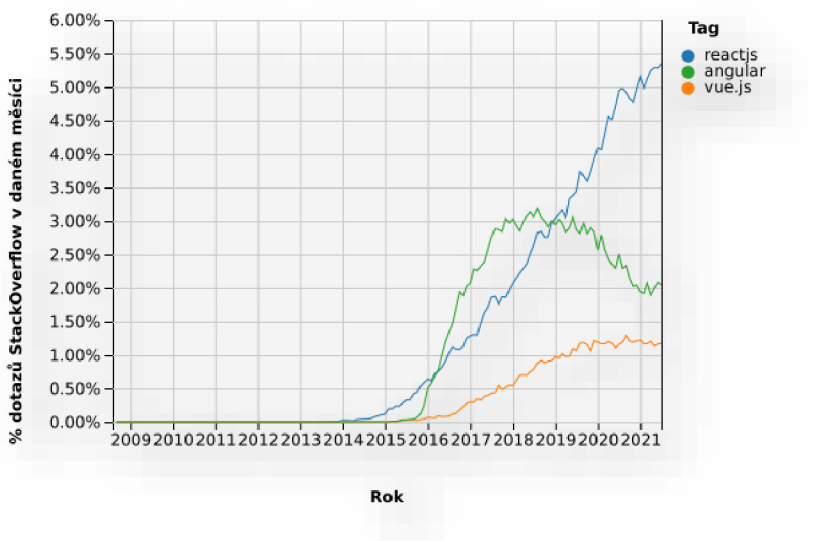
4.6 Srovnání popularity s ostatními frameworky

4.6.1 Stack Overflow

Stack Overflow [21] je populární stránka, kde uživatelé mohou pokládat dotazy s jejich problémy.

4.6.1.1 Stack Overflow Trends

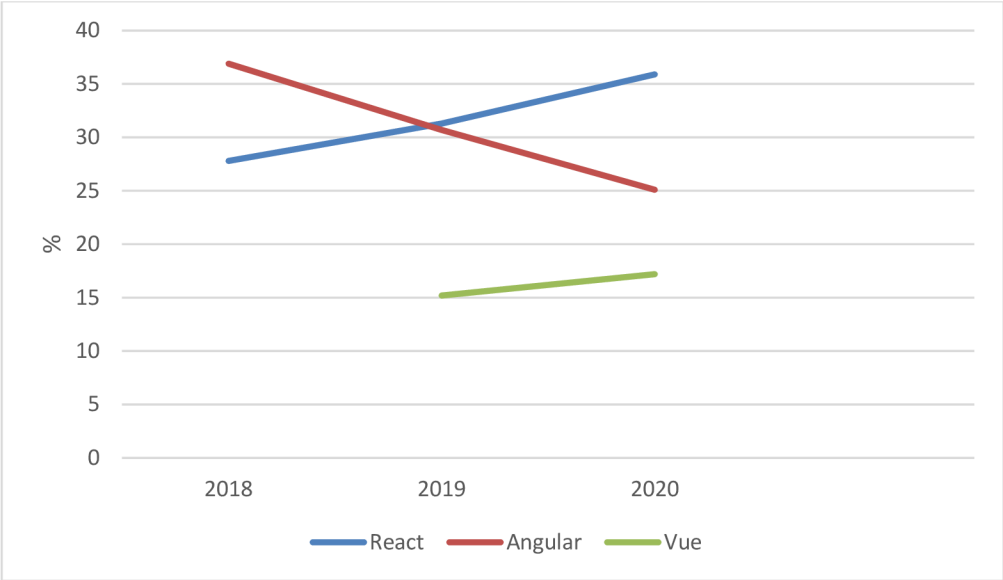
Stack Overflow Trends ukazuje počet procent z celkového počtu dotazů ohledně React.js, Angular, Vue.js. Tento graf může značit jejich oblíbenost, ale může značit počet problému, které uživatelé mají s každým frameworkem.



Graf 1: Dotazy v procentech, které byly položeny v daném měsíci. Zdroj: [22]

4.6.1.2 Stack Overflow Developer Survey

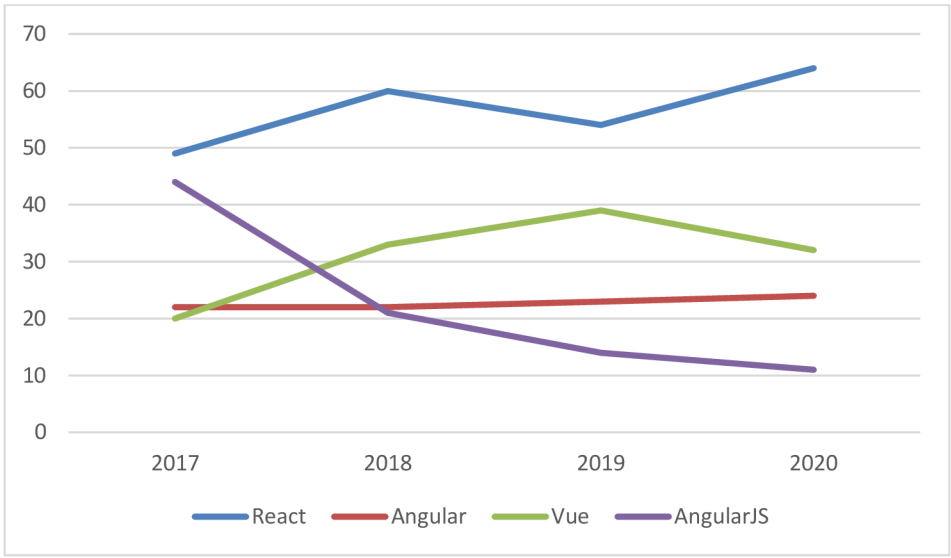
V roce 2018 (51 620 respondentů), 2019 (63 585 respondentů) a 2020 (42 279) odpovědělo na dotaz „Jaká je vaše nejoblíbenější technologie?“.



Graf 2: Popularita Frameworků podle Stack Overflow Developer Survey. Zdroj: [23], [24],[25]

4.6.2 Průzkum společnosti JetBrains

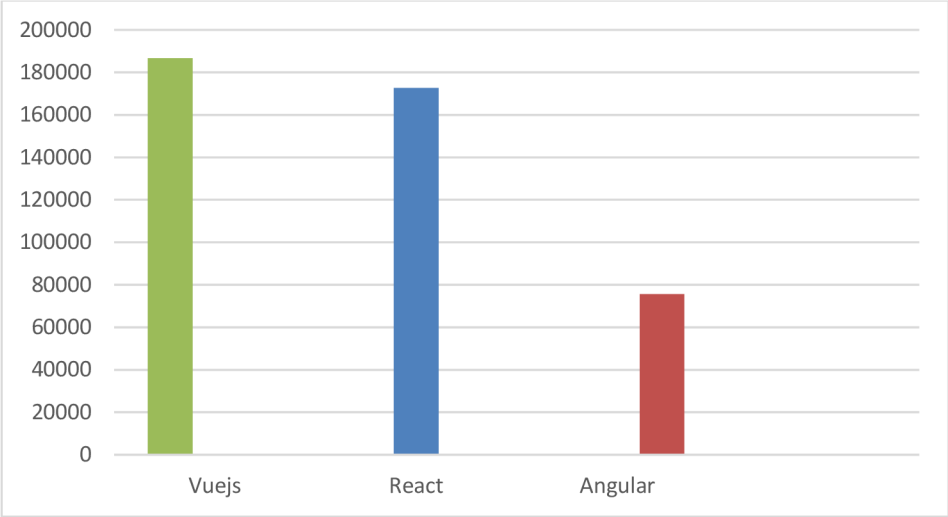
JetBrains [26] je česká společnost, která se zabývá vývojem vývojářských nástrojů. Na otázku: „Které frameworky často používáte?“ odpovědělo od roku 2017 do roku 2020 následující počty: 5 000, 6 000, 7 000 a 19 969 vývojářů.



Graf 3: Průzkum JetBrains: Popularita Frontendových Frameworků v rocích 2017-2020. Zdroj: [27], [28], [29], [30]

4.6.3 Github

Github [31] je populární služba na internetu mezi programátory, poskytující systém pro správu verzí zdrojových kódů, založené na programu Git. Uživatelé stránky mohou dát projektům hvězdičky, které lze interpretovat jako povědomí o projektu, případně slouží pro vytvoření záložek a tyto hvězdičky data lze využít pro měření popularity jednotlivých frameworků.



Graf 4: Počet hvězdiček na Githubu Zdroj: [11], [32], [33]

4.6.4 Shrnutí popularity

V předchozí grafech můžeme vidět, že si Vue udržuje druhé až třetí místo v pořadí mezi nejoblíbenějšími frameworky. V případě stránky Github je Vue dokonce na prvním místě.

5 Existující řešení daného problému

Na internetu je dostupných pár služeb specializující pro potřeby sběratelů pro zveřejnění svých sbíraných předmětů. V této kapitole budou představeny čtyři existující řešení, kde budou popsány klady a zápory každého řešení.

5.1 Pinterest.com

Pinterest [34] je sociální síť umožňující vytvářet kolekce videí, obrázků a nápadů, které si uživatelé mohou prohlédnout a inspirovat jimi.

Pinterest umí zařazovat jednotlivé předměty do nástěnek, bohužel neumožňuje přidání více obrázků u jednotlivých předmětů a popis detailních informací. Také je složité najít jednotlivé předměty, které jsou sice zařazeny do nástěnky, což při množství stovek až tisíc obrázků předmětů je zcela nemožné (chybí následná kategorizace podle typu předmětu).

5.2 GULDINER

GULDINER [35] je počítačový program pro Windows, od české společnosti, speciálně určený pro sběratele. Umožňuje sběrateli si vytvořit sbírku, do které lze evidovat jednotlivé předměty. Jeho hlavní nevýhodou je práce pouze v offline režimu a neumožňuje sběrateli tak svoji sbírku veřejně vystavit na internetu. Dále má také tato aplikace zastaralý design.

5.3 MyCollections

MyCollections [36] je multiplatformní aplikace pro Android, iOS a Windows, která umožňuje uživateli si založit vlastní sbírku s předměty, které lze kategorizovat a nahrávat k nim obrázky. MyCollections neumožňuje sběrateli veřejně zobrazit svoji sbírku na internetu, ale pouze soukromě. Dále mu chybí česká lokalizace.

5.4 Řešení pomocí galerií vytvořených pomocí blogu nebo webových stránek

Velká část sběratelů v České republice své osobních sbírky řeší pomocí poskytovatelů webových stránek a blogů zdarma. Po založení webové stránky (nebo i blogu) si sběratelé vytvoří vlastní webovou prezentaci a následně pro jednotlivé kategorie vytvoří galerii zvlášť na stránku. Do galerie přidávají svoji sbírku ve formě obrázků.

Tento způsob je dosti nevyhovující, jelikož sběrateli umožňuje většinou obrázek pouze pojmenovat a již nelze zadat podrobnosti o předmětu ve sbírce, nebo k němu přidat další obrázky.

6 Analýza aplikace

Tato kapitola se zabývá analýzou požadavků pro aplikaci, která je založena na nedostatcích existujících řešení a poznatků sběratele v předchozí kapitole.

6.1 Funkční požadavky

Funkční požadavky můžeme rozdělit do několika kategorií (Správu předmětů, Správa kategorií atd...) a konkrétně se jedná o tyto:

- Správu předmětů
    - Přidání předmětu
    - Úprava předmětu
    - Smazání předmětu
    - Přidání více obrázků k předmětu
    - Možnost přidat předmět do kategorie
    - Možnost přidat detailní popis předmětu
    - Možnost zobrazit počet předmětů ve sbírce
- Správu kategorií předmětů
    - Přidání kategorie
    - Smazání kategorie
    - Úprava kategorie
- Správu stránek
    - Vytvořit vlastní stránku
    - Upravovat obsah vlastní stránky
    - Smazat stránku
- Aplikace by neměla povolit neoprávněnému uživateli vstup do administrace stránky

6.2 Nefunkční požadavky

Dále také byly definovány následující nefunkční požadavky:

- Aplikace musí být připravená v budoucnu na jednoduché rozšíření o další funkce.
- Musí podporovat možný rozvoj na další platformy.
- Aplikace musí být jednoduchá na použití.
- Aplikace bude podporovat možnost rozšíření o další jazyky.
- Aplikace musí využívat databázi PostgreSQL.

6.3 Případ užití

V této části jsou popsány případy užití aplikace a jednotlivé role uživatelů.

6.3.1 Role

Aktéři jsou uživatelé aplikace, kteří budou rozděleny do čtyř odstupňovaných rolí podle práva uživatele. Aktéři jsou následující:

Nepřihlášený uživatel

Každému návštěvníkovi webové aplikace bude umožněno:

- Procházení cizích stránek
- Vyhledávat stránky
- Vytvořit si vlastní uživatelský účet
- Přihlásit se do svého účtu

Přihlášený uživatel

Nepřihlášený uživatel se stává po přihlášení přihlášeným uživatelem. Ten bude moct provádět tyto operace:

- Upravit osobní profil
- Vytvořit stránky
- Vyhledávat stránky
- Prohlížení stránek

Správce stránky

Po vytvoření stránky se uživatel automaticky stává správcem dané stránky a tím získá práva na:

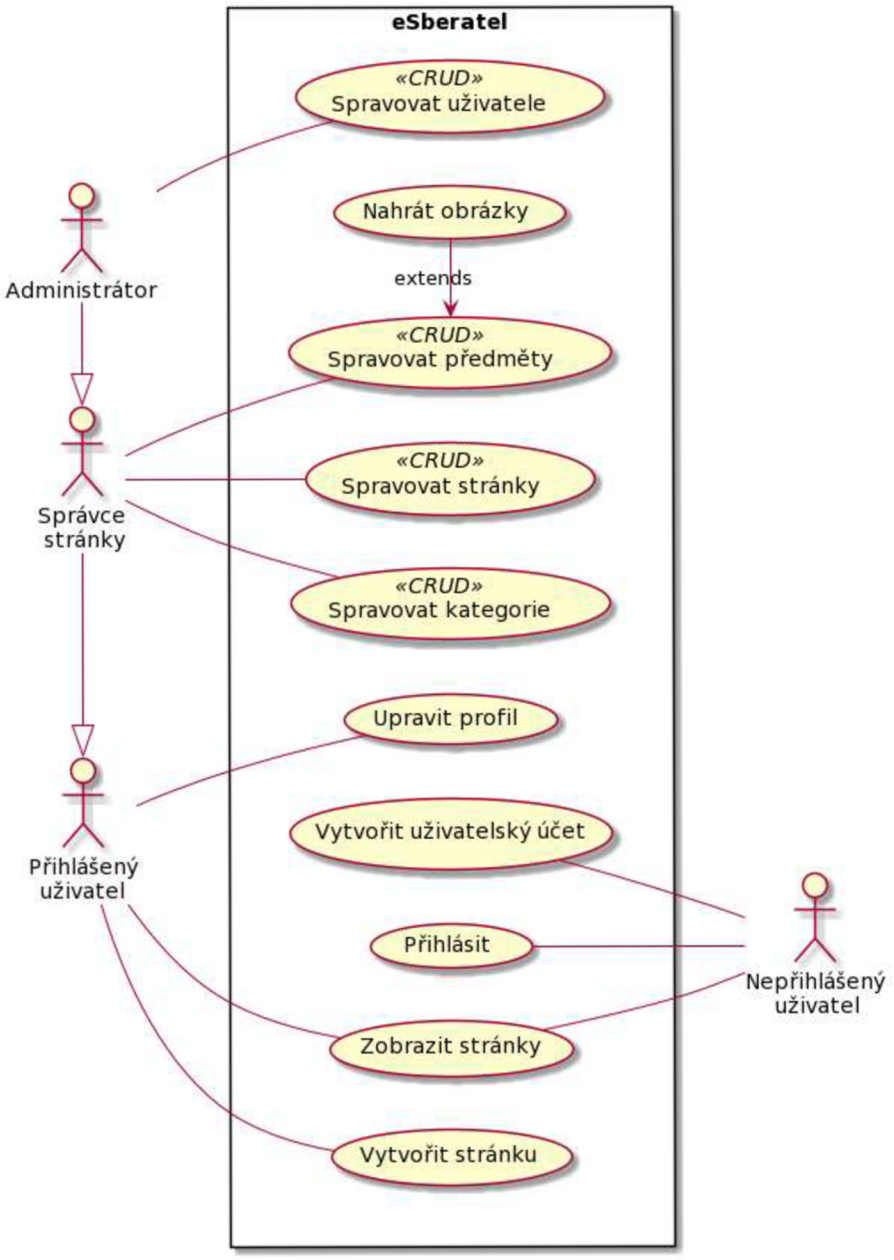
- Spravovat předměty
- Přidávat, upravovat a mazat položky
- Detailně vyhledávat svoje předměty
- Spravovat svoje stránky
- Spravovat kategorie

Administrátor

Administrátor stránky má nejvyšší povolení, který dědí práva správce stránky a může navíc:

- Spravovat uživatele
- Spravovat všechny stránky

Na obrázku níže jsou znázorněny jednotlivé role uživatelů, které jsou následně zobrazeny.



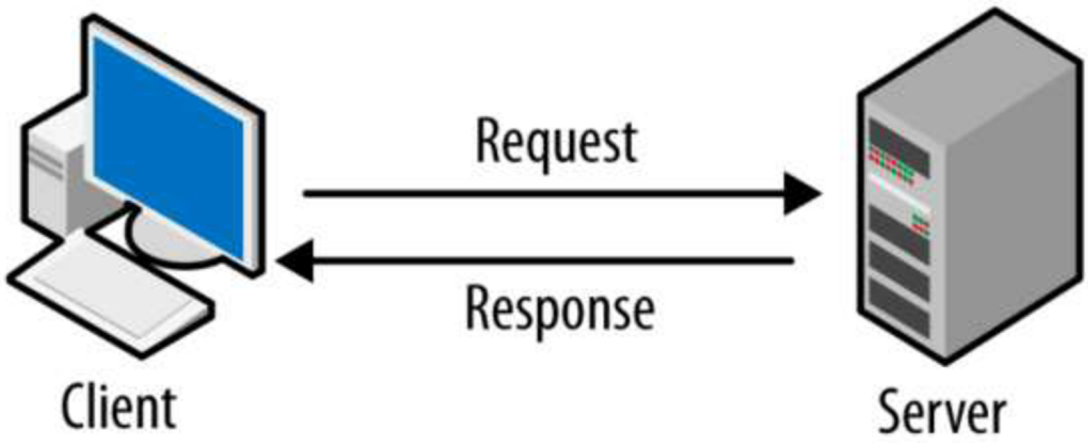
Obrázek 4: Diagram užití. Zdroj: [Autor]

7 Návrh aplikace

Tato kapitola se zabývá návrhem a popisem vybraných technologií pro implementaci webové aplikace s názvem eSberatel.

7.1 Rozdělení aplikace

Aplikace je navržena pomocí architektury klient-server, která se skládá ze dvou základních oddělených částí klienta a serveru, kteří mezi sebou komunikují přes počítačovou síť. Klient se často stará o uživatelské rozhraní a vytváření požadavků, které jsou odeslány na server. Server požadavky od klientů zpracuje a následně jim vrátí odpověď. [37]



Obrázek 5: Architektura Klient-Server. Zdroj:[37]

7.2 Komunikace klienta se serverem

Pro splnění požadavku na jednoduchou **rozšiřitelnost** aplikace byl zvolen dotazovací jazyk GraphQL. GraphQL byl interně vyvíjen společností Facebook od roku 2012 a veřejně uvedený v roce 2015. Jeho hlavní výhodou je, že umožňuje klientovi sdělit jakou strukturu dat vyžaduje od serveru, a ten mu vrátí strukturovaný JSON soubor podle jeho požadavků.

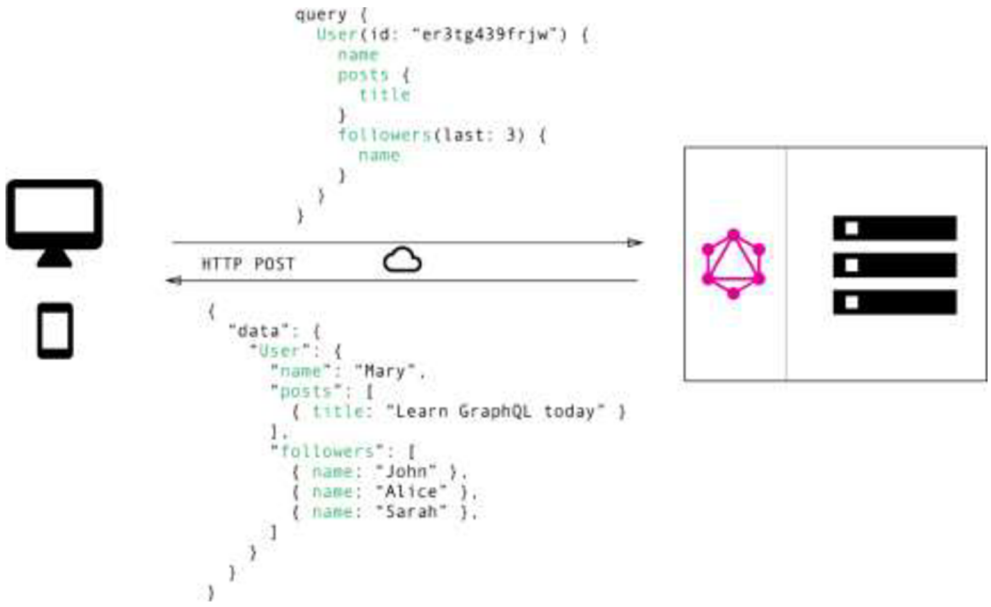
GraphQL pro komunikaci využívá převážně pouze HTTP metodu POST, ale lze případně použít metodu GET a definuje tři operace, které se dají využít při tvorbě dotazu:

- *Query* – slouží pro dotazování dat.
- *Mutation* – slouží pro úpravu dat na serveru a následně vrátí požadovaná data na zpět.

- *Subscription* – využívá se pro komunikaci v reálném čase, jako například chatu, pomocí web socketů. Server informuje klienta o vytvoření, nebo úpravě objektů.

GraphQL také řeší dva problémy, které jeho alternativu REST API. Jedním z nich je *overfetching,* kdy klient obdrží od serveru data, která nevyužije. Druhý problém je *underfetching*, kdy klient musí zaslat tři požadavky na server, aby získal data, která potřebuje, ale přitom také obdrží data, která nepotřebuje. Tyto problémy zobrazuje Obrázek 6. Kde GraphQL umožnuje tyto úkony provést najednou jsou znázorněny v obrázku 8.

Obrázek 6: Získání dat pomocí REST API. Zdroj: [38]



Obrázek 7: Získání dat pomocí GraphQL. Zdroj: [38]

7.3 Návrh databázového modelu

Na základě analýzy požadavků z předchozí kapitoly byla zvolena následují struktura databáze, která se bude skládat z těchto entit:

User – Uživatel

Entita User bude uchovávat jméno a příjmení, email, zaheslované heslo a také jeho roli, která bude definována v závislosti na roli uživatele za pomocí výčtového typu (enum). Tato hodnota bude výchozím stavu nastavená jako User, další možná hodnota je Admin.

Site – Stránka

Entita Site si uchovává data o jejím správci v relačním vztahu N:1. Dále informace o stránce samotné, které si uživatel může upravit a personalizovat tak svojí stánku.

Category – Kategorie

Entita Category uchovává její vlastnosti. Obsahuje relační vztah N:1 se stránkou a s předmětem ve vztahu 1:N.

Item – Předmět

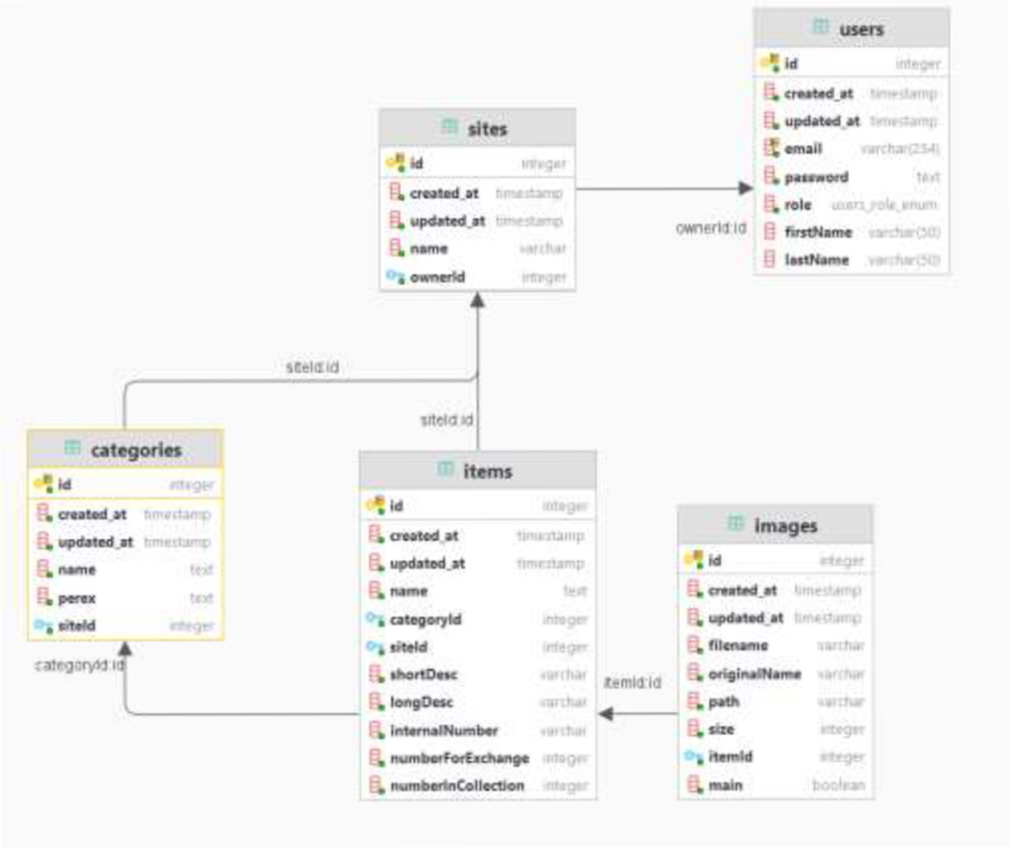
Entita Item bude obsahovat vlastnosti jako název, popis předmětu, přidělenou kategorii a obrázky. Obsahuje relační vztahy se stránkou a kategorií ve vztahu N:1 a s obrázkem ve vztahu 1:N.

Image – Obrázek

Entita Image bude uchovávat informace o uložení na disku, dále informaci, zda se má zobrazit jako hlavní obrázek při hromadném prohlížení. Entita má s entitou Předmět vztah N:1.

7.3.1 ERD diagram

Z uvedených entit byl navržen tento diagram:



Obrázek 8: ERD diagram databáze. Zdroj: [Autor]

8 Implementace aplikace

Následující kapitola představuje vybrané technologie a části implementace aplikace eSberatel.eu.

8.1 TypeScript

Při implementaci byl použit jazyk TypeScript.
TypeScript [39] je programovací jazyk s otevřeným kódem, který vytvořený společností Microsoft v roce 2012. TypeScript rozšiřuje jazyk JavaScript o statické typování jako je v jazycích Java, C#, C++ a C.

Statické typování umožňuje popsat typ proměnné a tvar objektu. Díky tomu poskytuje lepší dokumentaci a čitelnost zdrojového kódu. Dále upozorňuje na chyby spojené se záměnou typů při vývoji ještě při kompilaci, které by byli odhaleny až při spuštění.

Jak už bylo výše naznačeno Typescript se musí kompilovat zpět, pomocí kompilátoru, do JavaScriptu a nelze ho spustit přímo v prohlížeči, nebo na severu pomocí Node.js. Jelikož Typescript je nadmnožina Javascriptu, tak každý validní JavaScriptový kód se stává validním TypeScript kódem. Díky tomu umožňuje postupný přechod z JavaScriptu do TypeScriptu. [39]

Anotace typů

Typescript umožňuje deklaraci typů probíhá pomocí *:typ_promene*, tedy v případě řetězce *:string* za jménem proměnné.

V ukázce níže je ukázána anotace kódu v Typescriptu, kde je definován rozhraním objekt User a jeho vlastnosti. Následně je objekt naplněn a v případě zadání nesprávné hodnoty Typescript automaticky ohlásí chybu při kompilaci, nebo přímo v IDE. V tomto případě vypíše chybu, že *age* musí být číslo.

```
interface User{
    firstName: string;
    lastName: string;
    hasDrivingLicense: boolean;
    age:number;
}

const user1: User = {
    firstName: "John",
    lastName: "Doe",
    hasDrivingLicense: true,
    age: 25 //TS3222: Type 'number' is not assignable to type 'number'.
};

function sayHello(user: User):string{
    return `Hello ${user.firstName} ${user.lastName}`;
}
```

Ukázka kódu 11: Anotace kódu v TypeScriptu. Zdroj: [Autor]

8.2 Implementace klienta

Tato část se zabývá implementací vybraných řešení v klientské části aplikace za pomocí frameworku Quasar.

Quasar Framework neboli Quasar je open-source framework pod licencí MIT, který je založen na Vue. Poskytuje knihovnu s již připravenými komponentami, které jsou stylovány na základě Material designu verze 2.0 [40] od společnosti Google.

Pomocí tohoto frameworku lze vytvořit webové, mobilní, progresivní a multiplatformní desktopové aplikace skrze Electron [41]. Jednou z jeho výhod jsou předpřipravené vývojářské nástroje, které umožňují začít vyvíjet aplikace bez nutnosti konfigurace podpůrných nástrojů. Práci je využita verze 2, která používá nejnovější verzi Vue 3.

8.2.1 Komunikace se serverem

Vzhledem k vybranému API serveru GraphQL byla zvolena knihovna Vue Apollo [42], která je byla vytvořena pro komunikaci Vue se serverem pomocí jazyka GraphQL.

Pro získání dat ze serveru GraphQL je použita funkce useQuery s parametry dotazu, a proměnnými. Obdržený výsledek lze použít přímo v komponentách.

```
const queryResult = useQuery<ItemData, CatalogItemVars>(GET_CATALOG_ITEMS, {
  siteId: currentSiteId.value,
  categoryId: getParsedInt(<string>route.params.categoryId),
});
```

Ukázka kódu 12: Provedení dotazu pro získání předmětu pro katalog.

```
export const GET_CATALOG_ITEMS = gql`
  query items($categoryId:Int,$siteId:Int!){
  items( categoryId: $categoryId, siteId: $siteId) {
    id
    name
    numberForExchange
    numberInCollection
    internalNumber
    createdAt
    shortDesc
    images(main: true){
      path
      main
    }
  }
}`;
```

Ukázka kódu 13: Dotaz pro získání předmětů z kategorie. Zdroj: [Autor]

8.2.2 Kontrola autorizace na straně klienta

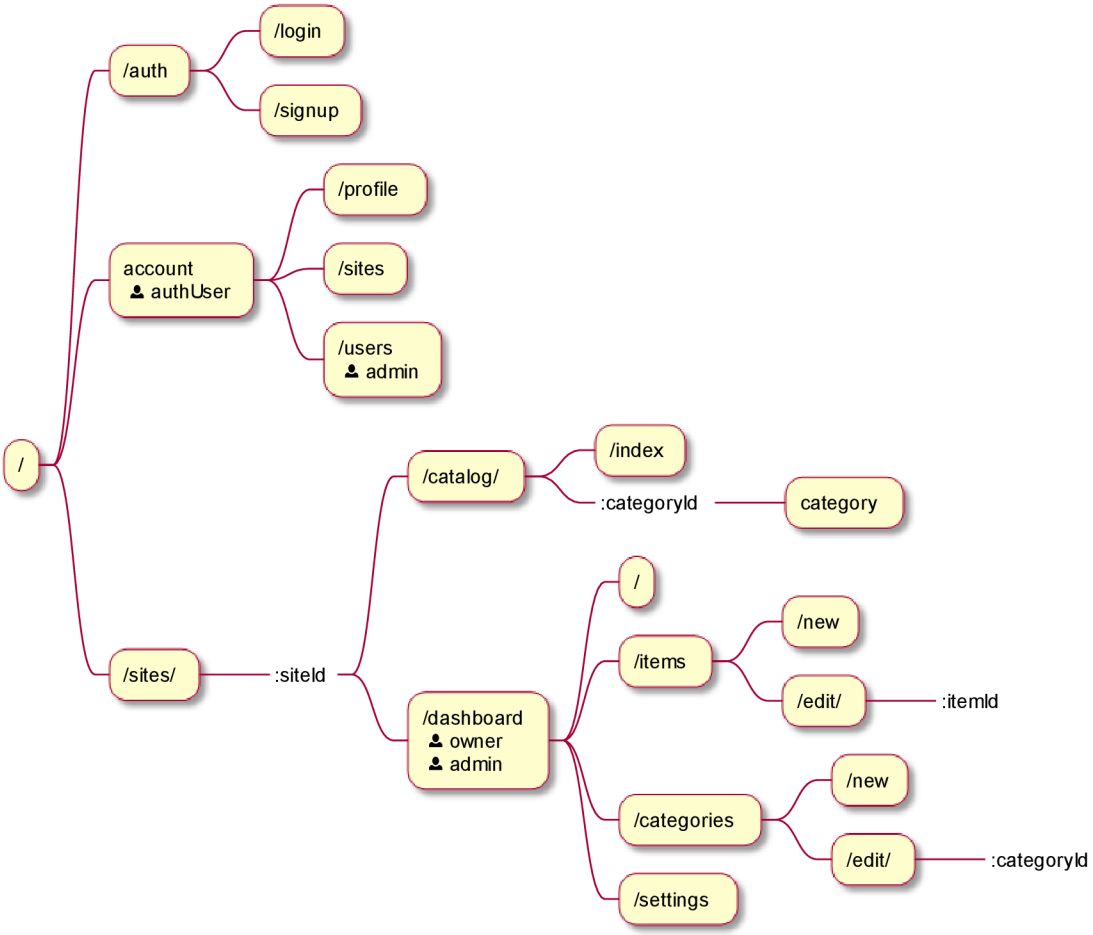
Vue Router poskytuje u každého atribut *beforeEnter,* do které lze přiřadit funkci, nebo pole funkcí, které můhou obsahovat ověřování podle zadaných parametrů a funkce *next()* přesměrovat *na* vybranou stránku.

V ukázce níže je zobrazeno ověření, zda uživatel je přihlášen a je zároveň vlastníkem stránky. Pokud je vlastníkem, tak může dále pokračovat, a pokud není, bude přesměrován na stránku s přihlášením.

```
const redirectToLogin = (to:RouteLocationNormalized) => ({
  name: 'login',
});

const requireOwner:NavigationGuard = (to, from, next) => {
  if (state.user?.sitesIds.includes(getParsedInt(to.params.siteId)) ||
    isAdmin()) {
    next();
  } else {
    next(redirectToLogin(to));
  }
};

const requireAuth:NavigationGuard = (to, from, next) => {
  if (state.isLoggedIn) {
    next(redirectToLogin(to));
  }
  next();
};

const routes = [{
  path: 'site/:siteId/dashboard',
  component: () => import('layouts/DashboardLayout.vue'),
  beforeEnter: [validateSiteId, requireAuth, requireOwner],
  meta: {
    showDrawer: true,
  },
}]
```

Ukázka kódu 14: Zabezpečení cest. Zdroj: [Autor]

Následující diagram zobrazuje jednotlivá rozcestí adres a práva přístupu uživatelů, které byli implementovány s pomocí Vue Routeru.



Obrázek 9: Návrh struktury adresy klientské aplikace. Zdroj: [Autor]

8.2.3 Lokalizace aplikace

Pro zajištění podpory více jazyků v aplikaci byla zvolena knihovna Vue I18n [43]. Ta umožňuje definovat strukturu (pomocí formátu JSON nebo JavaScriptových objektů), která je stejná pro všechny jazyky a následně k nim přiřadí vhodnou lokalizaci podle aktuálně vybraného jazyka.

Pro překlad textu byla použita funkce $t() s řetězcovým parametrem, který obsahuje cestu v objektu pomocí tečkové notace, a díky které lze jednoduše definovat jednotlivé řetězce a přistupovat k nim jako při procházení k objektu.

```
<template>
<q-btn
  :label="$t('auth.signup.button')"
  :to="{ name: 'signup' }"
  color="white"
  text-color="black"
/>
<q-btn
  :label="$t('auth.login.button')"
  :to="{ name: 'login' }"
  color="white"
  text-color="black"
/>
</template>
```

Ukázka kódu 15: Ukázka použití I18n v komponentě. Zdroj: [Autor]

```
export default {
  auth: {
    login: {
      login: 'Přihlášení',
      email: 'Email',
      password: 'Password',
    },
  },
};
```

Ukázka kódu 16: Část definice objektu české lokalizace . Zdroj: [Autor]

8.3 Implementace serveru

V následující části je popsána implementace frameworku Nest.js a jeho vybraných částí.

8.3.1 Nest.js

Nest.js [44] je Node.js framework pro vytváření výkonných a škálovatelných serverových aplikací. Byl vytvořen a umožňuje vývoj pomocí TypeScriptu. Kombinuje prvky objektově orientované, funkcionálního a funkčně reaktivního programování. Na pozadí využívá po Nest.js využívá jako výchozí populární HTTP Server framework Express [45]. Jeho hlavní výhodou je modulárnost a rychlost vývoje.

8.3.2 Struktura aplikace

Zdrojový kód aplikace serveru se nachází ve složce *api*, ve které byla vygenerována nová aplikace za pomocí balíčku @nestjs/cli.

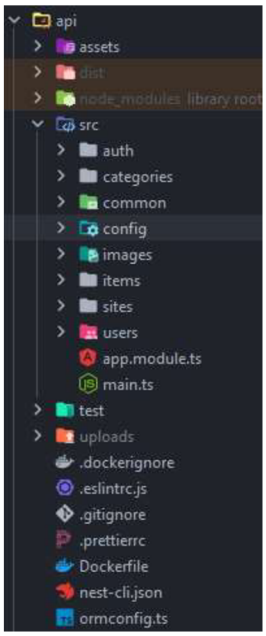
Ve složce *src* se nachází inicializační soubor pro spuštění serveru *main.ts* a kořenový modul *app.module.ts*, do kterého lze následně importovat další jednotlivé moduly. Tyto moduly reprezentují jednotlivé části schopností aplikace.

Pro potřebu implementované aplikace byli vytvořeny následující moduly:

- *auth* – modul, který řeší autentizaci a autorizaci
- *categories* – modul spravující kategorie
- *images* – modul spravující obrázky
- *items* – modul spravující předměty
- *sites* – modul spravující stránky
- *users* – modul spravující uživatele

Dále složka *src* obsahuje pomocné složky *config*, která obsahuje konfiguraci databáze a *common* pro sdílení tříd a funkcí, které jsou využity ve více modulech.

Pro potřeby nahrávání obrázků od uživatelů byla vytvořena složka *uploads* v kořenovém adresáři.



Obrázek 10: Struktura složek serveru. Zdroj: [Autor]

8.3.3 Moduly

Modul je základní svatební kámen jednotlivých funkcí, které jsou obsaženy v projektu:

- *Controller* – Definuje REST Api rozhraní.
- *Resolver* – Definuje jednotlivé operace pro GraphQL.
- *Service* – Slouží pro abstrakci logiky od Resolverů a Controllerů.
- *Guard* – Rozhoduje, zda daný uživatel má přístup ke zdroji.
- *Entity* – Definuje strukturu objektu v případě použití ORM.

8.3.4 Přístup do databáze

Pro uchování dat aplikace byla zvolena databáze PostgreSQL [46]. PostgreSQL je open-source objektově relační databázový systém, který je vyvíjen komunitou vývojářů.
Pro přístup do této databáze byl zvolena knihovna TypeORM, která umožňuje jednoduchou rozšiřitelnost aplikace.

TypeORM [31] je nástroj pro Node.js, vytvořený pro komunikaci a práci s databází, při které využívá techniky objektově relačního mapování (ORM). ORM usnadňuje přístup a úpravu dat v databázi přes API, díky tomu není potřeba psát dotazy v čistém SQL kódu a o generování SQL stará ORM na pozadí.
Struktura databáze je definována pomocí objektů entit, které reprezentují jednotlivé tabulky databáze. V TypeORM jsou tyto objekty definovány jako JavaScriptové objekty, nebo při využití TypeScriptu jako třídy a její vlastnosti určují jednotlivé sloupce. Tento způsob definování byl použit v navrhované aplikaci.

Vývoj TypeORM byl inspirován dalšími známými frameworky jako Hibernate pro jazyk Java. [47]

8.3.4.1 Entity

Entity je třída, která je označena dekorátorem @Entity() a reprezentuje jednu tabulku. Vlastnosti třídy, které jsou označeny dekorátorem @Column() definují jednotlivé sloupce tabulky. Primární klíč je definován pomocí @PrimaryGeneratedColum() a relace jsou definovány pomocí dekorátoru @OneToOne, @OneToMany().

```
@Entity({ name: 'items' })
export class Item{
  @PrimaryGeneratedColumn()
  id: number;

  @Column('text')
  name: string;

  @Column({ default: '' })
  shortDesc: string;

  @ManyToOne(() => Site, (site) => site.categories)
  @JoinColumn({ name: 'siteId' })
  site: Site;


  @OneToMany(() => Image, (image) => image.item, {
    cascade: ['update', 'remove', 'insert'],
  })
  images: Image[];
}
```

Ukázka kódu 17: Ukázka implementace části entity předmět. Zdroj: [Autor]

8.3.4.2 Práce s databází

Práce s databází a zpracování dat je prováděn za pomoci service. Service je reprezentována třídou, která má v konstruktoru definován jeden, nebo více repozitářů vytvořeného z definice entity. Následně pomocí tohoto repozitáře lze pracovat s databází pomocí jeho metod.

```
@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User) private usersRepository: Repository<User>,
  ) {}

  public async create(createUserInput: CreateUserInput): Promise<User> {
    const newUser = this.usersRepository.create(createUserInput);

    try {
      await this.usersRepository.insert(newUser);
      return newUser;
    } catch (error) {
      throw new ConflictException();
    }
  }

  public async findOne(id: number): Promise<User | undefined> {
    return this.usersRepository.findOneOrFail(id);
  }

  public async update(
    userId: number,
    updateUserInput: UpdateUserInput,
  ): Promise<User> {
    const user = await this.findOne(userId);

    if (!user) throw new BadRequestException('Invalid user');
    Object.assign(user, updateUserInput);
    await this.usersRepository.update(userId, user);
    return user;
  }
}
```

Ukázka kódu 18: Část service modulu uživatele. Zdroj: [Autor]

8.3.5 Komunikace s klientem

Pro komunikaci s klientem za použití GraphQL má Nest.js vlastní modul, který po jeho instalaci lze jednoduše naimportovat a nakonfigurovat.

8.3.5.1 Objektový typ

Objektový typ je třída s dekorátorem @ObjectType(), která definuje strukturu poskytovaného objektu přes GraphQL API. V případě této aplikace se jedná o třídu entity.

Pro definici jednotlivých vlastností typů je využit dekorátor @Field() s parametrem funkce s návratovou hodnotou typu, nebo objektu. Pro označení pole objektových typů, nebo základních typů se tento návratová hodnota funkce vloží do hranatých závorek.

```
@ObjectType('Item')
export class Item extends BaseEntity {
  @Field()
  name: string;

  @Field()
  shortDesc: string;

  @Field()
  longDesc: string;

  @Field()
  internalNumber: string;

  @Field(() => Int)
  numberForExchange: number;

  @Field(() => Int)
  numberInCollection: number;

  @Field(() => Category)
  category: Category;

  @Field(() => Site)
  site: Site;

  @Field(() => [Image], { nullable: 'itemsAndList' })
  images: Image[];
}
```

Ukázka kódu 19: Ukázka objektového typu předmět. Zdroj: [Autor]

8.3.5.2 Resolver

Pro definici jednotlivých operací pro daný modul je využit resolver. Resolver je třída označená dekorátorem *Resolver()* s parametrem funkce vracející příslušný Objektový typ.

Jednotlivé operace resolveru jsou definovány jako metody, které jsou dekorovány pomocí @Query(), @Mutation().

Dále metoda může mít dekorátor @ResolveField(), který podle kterého se dají slučovat objekty, které jsou propojené v objektovém typu.

```
@Resolver(() => Item)
export class ItemsResolver {
  constructor(
    private readonly itemsService: ItemsService,
    private readonly categoriesService: CategoriesService,
    private readonly imagesService: ImagesService,
  ) {}

  @Query(() => [Item], { name: 'items' })
  findAll(@Args() args: GetItemsArgs) {
    return this.itemsService.findAll(args);
  }

  @Query(() => Item, { name: 'item' })
  findOne(@Args('id', { type: () => Int }) id: number) {
    return this.itemsService.findOne(id);
  }

  @Mutation(() => Item)
  updateItem(@Args() { updateItemInput }: UpdateItemArgs) {
    return this.itemsService.update(updateItemInput.id, updateItemInput);
  }

  @ResolveField()
  async category(@Parent() item: Item): Promise<Category> {
    return this.categoriesService.findOne(item.categoryId);
  }
}
```

Ukázka kódu 20: Ukázka části resolveru modulu předmět. Zdroj: [Autor]

8.3.6 Autorizace

Pro ověření, zda daný požadavek od uživatele má být zpracován se v Nest.js používá dekorátor *@UseGuards().* Ten lze použít pro třídu revolveru a controlleru, nebo také jednotlivě pro jejich metody. Přijímá jako parametry třídy s implementovaným rozhraním *CanActivate*. Rozhraní definuje metodu CanActivate, která po implementaci rozhodne, zda povolí přístup na základě poskytnutého kontextu, který je jí předán při autentizaci.

V ukázce níže je implementace, která ověřuje, zda daný uživatel je vlastník stránky, nebo administrátor.

```
@Injectable()
export class RoleGuard implements CanActivate {
  canActivate(
    context: ExecutionContext,
  ): boolean | Promise<boolean> | Observable<boolean> {
    const ctx = GqlExecutionContext.create(context);
    const { req } = ctx.getContext();

    let allowAccess = false;
    //  Check if user is owner of site
    if (req?.user?.sitesIds.includes(req?.body?.variables?.siteId))
      allowAccess = true;
    //  Check if user is admin
    if (req.user && req.user.role === UserRole.ADMIN) allowAccess = true;

    return Boolean(allowAccess);
  }
}
```

Ukázka kódu 21: Implementace RoleGuard. Zdroj: [Autor]

8.3.7 Validace objektů

O validaci příchozích dat od klienta se stará class-validator, ten poskytuje jednotlivé validační dekorátory, které lze aplikovat na vlastnosti. Jako například zda vlastnost objektu je číslo, řetězec, email, má určitou délku, není prázdný.

Pro zapnutí validace je nutné přidat následující řádek do souboru *main.ts*.

```
app.useGlobalPipes(  new ValidationPipe());
```

Ukázka kódu 22: Aplikace validace objektů v rámci celé aplikace. Zdroj: [Autor]

```
export class CreateUserInput {
  @IsEmail()
  @IsNotEmpty()
  email: string;

  @IsString()
  @IsNotEmpty()
  @MinLength(8)
  password: string;

  @IsString()
  @MaxLength(50)
  @IsNotEmpty()
  firstName: string;

  @IsString()
  @MaxLength(50)
  @IsNotEmpty()
  lastName: string;
}
```

Ukázka kódu 23: Validace objektu pro vytvoření uživatele. Zdroj: [Autor]

9 Zhodnocení frameworku Vue.js

Framework Vue.js o sobě tvrdí, že je přístupný všem, kteří umí základy jazyků HTML, CSS a Javascriptu. Toto tvrzení mohu potvrdit,

Jelikož Vue má vynikající dokumentaci, která uživatele podrobně provede od základů až do jeho pokročilých funkcí. Nechybí zde také příklady kódu, část z nich lze volně upravovat a zkoušet si tak koncepty bez nutnosti instalace. Pro zkušené uživatele nabízí podrobný API rejstřík.

Druhou velkou částí, která přispívá přístupnosti je strukturalizace kódu s využitím jednosouborových komponent, které jsou rozděleny do tří jasně definovaných tagů template(šablony) pro definici HTML, tagu script(skriptu) pro definici Javascriptu a tagu style (stylu).

Další výhoda tohoto frameworku jsou nástroje @vue/cli [48], nebo nově Vite [18], kteří umožňují na základě konfigurace uživatele vygenerovat startovací projekt, který je ihned připraven pro vývoj. Při využití nástroje @vue/cli lze po inicializaci jednoduše přidat další oficiální či komunitní doplňky, které rozšíří startovací aplikaci o další funkce. Pro Vue byl vytvořen doplněk do prohlížeče pro umožnění ladění a zobrazení stavů aplikace.

Vue na svých stránkách také píše, o své univerzálnosti. Tento výrok mohu potvrdit, jelikož Vue je všestranné. Jeho kód může být přímo vložen a spuštěn v HTML souboru, bez nutnosti úpravy dalšími nástroji. Umožňuje také přidat pouze interaktivní prvek do části stránky, nebo ji celou nahradit a po přidání pluginu vue-router tak vytvořit jednostránkovou aplikaci. Tímto způsobem lze také plynule převést stránku z html do Vue. Dále umí, po přidání dalších doplňků vykreslovat stránky na straně serveru (SSR), kdy klient, který nemá zapnutý JavaScript v prohlížeči uvidí její obsah na místo prázdné stránky.

Práce se samotným frameworkem Vue byla velmi příjemná, díky jeho intuitivnosti, vynikají dokumentaci a organizaci komponent.
Tento framework bych doporučil dalším ostatním programátorům, pokud uvažují o vytvoření jednostránkové webové aplikace. Dle mého zjištění je Vue vhodné i pro větší projekty. Pokud bych měl znovu vytvářet webovou aplikaci, zvolil bych si znovu tento framework Vue.js.

10 Shrnutí výsledků

V bakalářské práci jsem se zabýval představením a zhodnocením frameworku Vue.js na základě vytvoření ukázkové aplikace pro sběratele.

V první části jsem vymezil pojmy DOM a Virtuální DOM, dále jednostránkových aplikací a jejich rozdílu s vícestránkovými aplikacemi, kde vedl jsem jejich výhody a nevýhody.

Ve druhé části jsem představil framework Vue.js a jeho knihovnu Vue Router. Porovnal jeho popularitu mezi ostatními hlavními frameworky React.js a Angular.

Ve třetí až páté části jsem vytipoval vhodné existují řešení, kterým jsem následně analyzoval jejich silné a slabé stránky. Následně jsem sesbíral požadavky na aplikaci od uživatele. Analyzoval jsem danou problémovou oblast s přihlédnutím k těmto předchozím krokům. Vybral jsem vhodné rozhraní pro komunikaci mezi klientem a serverem. Navrhnul jsem vhodnou strukturu databáze.

V šesté části jsem implementoval a popsal použité technologie pro klientskou a serverovou část experimentální aplikace. Kde jsem implementoval jsem vhodný způsob autentizace, autorizace, validace a lokalizace. Přístup do databáze PostgreSQL byl řešen pomocí nástroje TypeORM.

V poslední části jsem zhodnotil Vue.js framework na základě mých předchozích a nově nabytých zkušeností při vývoji ukázkové aplikace a tvrzeních o Vue.js, které uvádí na svých stránkách.

11 Závěry a doporučení

Cílem této práce bylo experimentálně vyhodnotit Vue.js framework, pro za tímto účelem byla vypracována ukázková aplikace pro sběratele eSberatel, která je dostupná na adrese esberatel.eu.

Framework Vue.js jsem zhodnotil a vypracoval ukázkovou aplikaci, kde jsem implementoval většinu požadavků. Požadavky, které nebyli implementovány jsou administrátorské prostředí, česká lokalizace a detail předmětů, hlavní strana se soupisem stránek uživatelů.

Možné rozšíření

V retrospektivě by bylo vhodné provést strukturalizaci projektu ve formě monorepa, například pomocí Yarn workspaces. Za jeho pomocí by byla vytvořena knihovna, která by umožnovala jednoduché sdílení rozhraní a pomocných funkcí TypeScriptu mezi klientskou a serverovou částí aplikace.

Další verze aplikace by mohla nabízet funkce spravování stránky více uživateli, pokročilejší vyhledávání předmětů, zadání vlastních parametrů u předmětů, přidání dalších možností ověření účtu pomocí emailu, nebo přihlášení za pomocí OAuth.

Mobilní aplikace

Na základě nefunkčního požadavku umožnění rozšíření na další platformy. Lze implementovat mobilní aplikaci, která by uživateli umožnovala rychlí přehled nad jeho sbírkou a pořízení obrázku předmětů přímo z fotoaparátu zařízení.

12 Seznam použité literatury

[1] JONATHAN ROBIE. What is the Document Object Model? *W3.org* [online]. [cit. 2021-03-20]. Dostupné z: https://www.w3.org/TR/WD-DOM/introduction.html

[2] V3.VUEJS.ORG. *Rendering Mechanisms and Optimizations | Vue.js* [online]. [cit. 2021-08-11]. Dostupné z: https://v3.vuejs.org/guide/optimizations.html#virtual-dom

[3] REACTJS.ORG. *Virtual DOM and Internals – React* [online]. [cit. 2021-08-11]. Dostupné z: https://reactjs.org/docs/faq-internals.html

[4] FLANAGAN, David. *JavaScript: the definitive guide*. 6th ed. Beijing ; Sebastopol, CA: O'Reilly, 2011. ISBN 978-0-596-80552-4.

[5] MIKE WASSON. *ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET* [online]. [cit. 2021-08-07]. Dostupné z: https://docs.microsoft.com/en-us/archive/msdn-magazine/2013/november/asp-net-single-page-applications-build-modern-responsive-web-apps-with-asp-net

[6] STACKOVERFLOW.COM. Javascript - Single Page Application: advantages and disadvantages. *Stack Overflow* [online]. [cit. 2021-08-08]. Dostupné z: https://stackoverflow.com/questions/21862054/single-page-application-advantages-and-disadvantages

[7] GOOGLE.COM. Understand JavaScript SEO Basics | Google Search Central. *Google Developers* [online]. [cit. 2021-08-08]. Dostupné z: https://developers.google.com/search/docs/advanced/javascript/javascript-seo-basics?hl=cs

[8] SIMONA HAJŠMANOVÁ. Odhalte včas problémy a řešení SEO u obsahu tvořeného JavaScriptem. *Blog ACOMWARE* [online]. 25. leden 2019 [cit. 2021-08-08]. Dostupné z: https://blog.acomware.cz/seo-javascript-problemy/

[9] OZITAG.COM, OZiTAG. *Single-page applications vs. multiple-page applications: pros, cons, pitfalls* [online]. [cit. 2021-08-08]. Dostupné z: https://ozitag.com/blog/spa-advantages

[10] V3.VUEJS.ORG. *Introduction | Vue.js* [online]. [cit. 2021-08-12]. Dostupné z: https://v3.vuejs.org/guide/introduction.html

[11] *vuejs/vue* [online]. JavaScript. B.m.: vuejs, 2021 [cit. 2021-08-10]. Dostupné z: https://github.com/vuejs/vue

[12] VIVIAN CROMWELL. Between the Wires: An interview with Vue.js creator Evan You. *freeCodeCamp.org* [online]. 30. květen 2017 [cit. 2021-01-30]. Dostupné z: https://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/

[13] VUEJS TEAM. Release v3.0.0 One Piece · vuejs/vue-next. *GitHub* [online]. [cit. 2021-08-08]. Dostupné z: https://github.com/vuejs/vue-next/releases/tag/v3.0.0

[14] *Components Basics | Vue.js* [online]. [cit. 2021-08-15]. Dostupné z: https://v3.vuejs.org/guide/component-basics.html

[15] V3.VUEJS.ORG. *Single File Components | Vue.js* [online]. [cit. 2021-08-09]. Dostupné z: https://v3.vuejs.org/guide/single-file-component.html#introduction

[16] webpack. *webpack* [online]. [cit. 2021-08-12]. Dostupné z: https://webpack.js.org/

[17] VUE-LOADER.VUEJS.ORG. *Introduction | Vue Loader* [online]. [cit. 2021-08-12]. Dostupné z: https://vue-loader.vuejs.org/

[18] *Vite* [online]. [cit. 2021-11-11]. Dostupné z: https://vitejs.dev/

[19] *Application & Component Instances | Vue.js* [online]. [cit. 2021-08-15]. Dostupné z: https://v3.vuejs.org/guide/instance.html#lifecycle-diagram

[20] ROUTER.VUEJS.ORG. *Getting Started | Vue Router* [online]. [cit. 2021-07-04]. Dostupné z: https://next.router.vuejs.org/guide/

[21] STACKOVERFLOW.COM. Stack Overflow - Where Developers Learn, Share, & Build Careers. *Stack Overflow* [online]. [cit. 2021-08-12]. Dostupné z: https://stackoverflow.com/

[22] STACKOVERFLOW.COM. *Stack Overflow Trends* [online]. [cit. 2021-08-09]. Dostupné z: https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular

[23] STACKOVERFLOW.COM. Stack Overflow Developer Survey 2018. *Stack Overflow* [online]. [cit. 2021-08-09]. Dostupné

z: https://insights.stackoverflow.com/survey/2018/?utm_source=so-owned&utm_medium=social&utm_campaign=dev-survey-2018&utm_content=social-share

[24] STACKOVERFLOW.COM. Stack Overflow Developer Survey 2019. *Stack Overflow* [online]. [cit. 2021-08-09]. Dostupné z: https://insights.stackoverflow.com/survey/2019/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2019

[25] STACKOVERFLOW.COM. Stack Overflow Developer Survey 2020. *Stack Overflow* [online]. [cit. 2021-08-09]. Dostupné z: https://insights.stackoverflow.com/survey/2020/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2020

[26] JetBrains: Essential tools for software developers and teams. *JetBrains* [online]. [cit. 2021-08-12]. Dostupné z: https://www.jetbrains.com/

[27] JETBRAINS.COM. JavaScript in 2017 - The State of Developer Ecosystem by JetBrains. *JetBrains* [online]. [cit. 2021-08-10]. Dostupné z: https://www.jetbrains.com/research/devecosystem-2017/javascript/

[28] JETBRAINS.COM. JavaScript in 2018 - The State of Developer Ecosystem by JetBrains. *JetBrains* [online]. [cit. 2021-08-10]. Dostupné z: https://www.jetbrains.com/research/devecosystem-2018/javascript/

[29] JETBRAINS.COM. JavaScript 2019 - The state of Developer Ecosystem in 2019 Infographic. *JetBrains* [online]. [cit. 2021-08-10]. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2019/javascript/

[30] JETBRAINS.COM. JavaScript Programming - The State of Developer Ecosystem in 2020 Infographic. *JetBrains: Developer Tools for Professionals and Teams* [online]. [cit. 2021-08-10]. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2020/

[31] GITHUB.COM. Build software better, together. *GitHub* [online]. [cit. 2021-08-12]. Dostupné z: https://github.com

[32] MIKE WASSON. *React ·* [online]. JavaScript. B.m.: Facebook, 2021 [cit. 2021-08-10]. Dostupné z: https://github.com/facebook/react

[33] GITHUB.COM. *Angular - The modern web developer's platform.* [online]. TypeScript. B.m.: Angular, 2021 [cit. 2021-08-10]. Dostupné z: https://github.com/angular/angular

[34] PINTEREST.COM. Pinterest. *Pinterest* [online]. [cit. 2021-08-12]. Dostupné z: https://cz.pinterest.com/

[35] PTERANODONSOFT.CZ. *PTERANODONSOFT* [online]. [cit. 2021-08-12]. Dostupné z: https://www.pteranodonsoft.cz/

[36] MYCOLLECTIONS.ALTOVA.COM. *MyCollections* [online]. [cit. 2021-08-12]. Dostupné z: https://mycollections.altova.com/

[37] ALI MADOOEI. *Client-server Application - OOSE* [online]. [cit. 2021-08-16]. Dostupné z: https://madooei.github.io/cs421_sp20_homepage/client-server-app/

[38] HOWTOGRAPHQL.COM. *GraphQL vs REST - A comparison* [online]. [cit. 2021-08-11]. Dostupné z: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

[39] TYPESCRIPTLANG.ORG. Typed JavaScript at Any Scale. *Typescript* [online]. [cit. 2021-08-08]. Dostupné z: https://www.typescriptlang.org/

[40] MATERIAL.IO. Material Design. *Material Design* [online]. [cit. 2021-08-12]. Dostupné z: https://material.io/design

[41] *Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.* [online]. [cit. 2021-08-12]. Dostupné z: https://www.electronjs.org/

[42] APOLLO.VUEJS.ORG. *Vue Apollo* [online]. [cit. 2021-08-12]. Dostupné z: https://apollo.vuejs.org/

[43] *Home | Vue I18n* [online]. [cit. 2021-08-14]. Dostupné z: https://vue-i18n.intlify.dev/

[44] NEST.JS. NestJS - A progressive Node.js framework. *NestJS - A progressive Node.js framework* [online]. [cit. 2021-08-12]. Dostupné z: https://nestjs.com

[45] NEST.JS. *Documentation | NestJS - A progressive Node.js framework* [online]. [cit. 2021-01-24]. Dostupné z: https://docs.nestjs.com/

[46] SKUPINA, PostgreSQL Global Development. PostgreSQL. *PostgreSQL* [online]. 12. srpen 2021 [cit. 2021-08-12]. Dostupné z: https://www.postgresql.org/

[47] *Connection | TypeORM* [online]. [cit. 2021-08-04]. Dostupné z: https://typeorm.io/#/

[48] *Vue CLI* [online]. [cit. 2021-11-11]. Dostupné z: https://cli.vuejs.org/

13 Přílohy

Aplikace je dostupná na stránce esberatel.eu.

Zdrojový kód aplikace je také dostupný na adrese:

https://github.com/michaldolensky/ecollector

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Michal Dolenský**
Studium:	I1700073
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Webová aplikace pro sběratele za použití Vue.js frameworku**
Název bakalářské práce AJ:	Web application for collectors using Vue.js framework.

Cíl, metody, literatura, předpoklady:

Cílem práce je experimentálně ověřit Vue.js framework pro vytvoření webové aplikace pro sběratele. Osnova: 1. Úvod 2. Představení frameworku Vue.js 3. Analýza aplikace 4. Návrh aplikace 5. Implementace aplikace 6. Závěr 7. Literatura a použité zdroje

FILIPOVA, Olga. Learning Vue.js 2: Learn how to build amazing reactive web applications easily with Vue.js. Birmingham, U.K.: Packt Publishing, [2016]. ISBN 978-1-78646-994-6.

PASSAGLIA, Andrea. Vue.js 2 Cookbook: Build modern, interactive web applications with Vue.js. Birmingham, U.K.: Packt Publishing, [2017]. ISBN 978-1-78646-809-3.

MACRAE, Callum. Vue.js: up and running: building accessible and performant web apps. Sebastopol, California: O'Reilly Media, [2018]. ISBN 978-1-491-99724-6.

Garantující pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Mgr. Daniela Ponce, Ph.D.
Oponent:	Mgr. Hana Rohrová

Datum zadání závěrečné práce: 21.10.2014