VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE PRO SBĚRATELE

WEB APPLICATION FOR COLLECTORS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JANA BALKOVICOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Prof. Ing. TOMÁŠ HRUŠKA, CSc.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Studentka: **Balkovicová Jana**
Program: Informační technologie
Název: **Webová aplikace pro sběratele**
Web Application for Collectors
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s tvorbou single-page aplikací za pomocí JavaScript frameworků.
2. Analyzujte dostupná řešení aplikace pro sběratele a identifikujte nedostatky. Zaměřte se na univerzálnost aplikační domény.
3. Navrhněte architekturu webové aplikace a uživatelské rozhraní splňující MVP.
4. Navržené řešení implementujte pomocí vhodných technologií.
5. Proveďte uživatelské testování aplikace na rozmanité skupině sběratelů a na základě výsledků doporučte další rozvoj aplikace.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Emmit A. Scott, Jr.: SPA Design and Architecture, Manning, 2015
- Michael S. Mikowski: Single Page Web Applications, Manning, 2013
- Richardson: RESTful Web APIs, O'Reilly Media, Inc., 2013
- Shannon Bradshaw: MongoDB: The Definitive Guide, O'Reilly Media, Inc., 2019

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hruška Tomáš, prof. Ing., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Táto práca sa zaoberá návrhom a implementáciou webovej aplikácie vhodnej pre zberateľov rôznorodých predmetov. Cieľom je vytvoriť aplikáciu atraktívnu pre používateľov rôznych vekových kategórií za použitia moderných technológií a architektonických vzorov pri jej implementácii. Používateľ si v aplikácii vie vytvoriť kolekcie a ľubovoľne definovať atribúty, ktoré bude v položkách daných kolekcií zaznamenávať. Dizajn používateľského prostredia je moderný a zrozumiteľný pre všetkých používateľov. V tejto práci sa podarilo navrhnúť a implementovať MVP verziu danej aplikácie, pričom táto verzia prešla prvým používateľským testovaním, na základe čoho boli definované zmeny, ktoré by sa implementovali v ďalších iteráciách vývoja.

Abstract

This thesis describes design and implementation of a web application suitable for people who collect items of various kind. The goal is to create a nice-looking application for users of any age with a use of modern technologies and architectural principles. The user is able to create collections and define custom properties, which they will use in order to track different attributes of collection items. The user interface design is attractive and clear for every user. In this thesis we managed to design and implement a minimal viable product of the application. This version passed user testing and based on users' comments additional changes were defined. These changes are about to be implemented in the next iterations of the development.

Kľúčové slová

webová aplikácia, web, MVP, zbierky, SPA, JavaScript, React, Redux, Node, REST, Express, Docker, MongoDB, autorizácia, Google Auth, používateľské testovanie

Keywords

web application, web, MVP, collections, SPA, JavaScript, React, Redux, Node, REST, Express, Docker, MongoDB, authorization, Google Auth, user testing

Citácia

BALKOVICOVÁ, Jana. *Webová aplikace pro sběratele*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Tomáš Hruška, CSc.

Webová aplikace pro sběratele

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána prof. Ing. Tomáša Hrušky, CSc. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

.....
Jana Balkovicová
4. mája 2021

Poďakovanie

Rada by som poďakovala vedúcemu mojej bakalárskej práce, prof. Ing. Tomášovi Hruškovi, CSc., za ochotu prijať moju vlastnú tému, jeho cenné rady a pripomienky.

Obsah

Kapitola 1

Úvod

V dnešnej ére nám technologické novinky uľahčujú a skvalitňujú život. Používame ich každodenne – komunikujeme cez mobilnú sieť alebo internet, nakupujeme v internetových obchodoch, používame „smart“ zariadenia na domáce práce, učíme sa z webinárov, hráme hry. Zdá sa, že sa všetko presúva do sveta jednotiek a núl, kde je všetko virtuálne. Avšak ešte stále existujú ľudia, ktorí majú radšej veci hmotné. Napríklad takí zberatelia, ktorým radosť robia kolekcie známok, izbových rastlín, kníh alebo pohľadníc.

Je prirodzené obsah svojich zbierok zaznamenávať. Zberatelia si mnohokrát vedú katalógy svojich predmetov či už ručne na papieri alebo digitálne v počítači. Digitálna podoba má omnoho viac výhod – je možné v nej vyhľadávať, filtrovať, položky jednoducho zoradiť alebo upraviť. Štatistiky alebo výpočty sú nad týmito dátami počítané automaticky, a automaticky sa aktualizujú po vykonaní zmien. V digitálnom svete sa taktiež lepšie dáta zálohujú a ak sa nachádzajú na internete, sú dostupné prakticky vždy a všade.

Ak by zberatelia cítili potrebu v dnešnej dobe svoje zbierky katalogizovať, sú odkázaní na mobilné aplikácie so zlým používateľským rozhraním, a počítačové alebo webové aplikácie, ktoré sú určené skôr na inventarizáciu. Inventarizácia vyžaduje implementáciu atribútov a funkcií ako napríklad informácie o počte kusov daného produktu, notifikácie, ak množstvo daného produktu presiahne minimálnu hranicu, prípadne vyhľadávanie položiek pomocou čiarového kódu. Tieto funkcie ale zberatelia väčšinou nevyužijú. Existujúce riešenia pre zberateľov sú málo populárne, väčšinou zamerané na určitý operačný systém a na konkrétny zberateľský druh predmetu. Používatelia-zberatelia sú tým pádom rozptýlení medzi množstvo aplikácií, ktorých vývoj nie je poháňaný silnou komunitou, keďže taká komunita sa prirodzene nevie vytvoriť. Podľa mňa by malo existovať riešenie, ktoré spojí zberateľov rôznych predmetov a rôznych vekových kategórií, a bude dostupné pre všetkých na webe. Počas svojho prieskumu som na podobnú aplikáciu nenarazila, čo vidím ako dieru na trhu.

V kapitole 2 sú popísané tri existujúce riešenia, ktoré ma zaujali funkcionalitou alebo dizajnom. Teoretické znalosti o histórii, architektúre a bezpečnostných rizikách webových aplikácií sú popísané v kapitole 3. V tejto kapitole môžeme nájsť aj porovnanie rôznych druhov databázových systémov ako sú relačné, NoSQL a NewSQL úložiská. V návrhu aplikácie som sa snažila zvoliť moderné technológie, ktoré sa dnes bežne využívajú. Funkcie, dátový model a architektúru je možné vidieť v kapitole 4. Kapitola 5 detailne popisuje server, klient a ich komunikáciu v implementovanej MVP verzii aplikácie. Po nasadení tejto verzie na produkčný server bola aplikácia podrobená používateľskému testovaniu. Kapitola 6 sa venuje pripomienkam používateľov, ktorí aplikáciu testovali, a popisuje zmeny, ktoré by boli implementované v ďalších iteráciách vývoja aplikácie.

Kapitola 2

Aplikácie pre zberateľov

Na internete existuje množstvo webových stránok, dostupných programov a služieb, no tých, čo by mohli zberatelia využiť na inventarizáciu svojich zbierok, je prekvapivo málo. Z aplikácií, ktoré sú vyslovene pre zberateľov, ani jedna nespĺňa požiadavky dané modernou dobou.

Táto kapitola zhodnotí existujúce riešenia a na základe chýbajúcich vlastností navrhne aplikáciu, ktorá bude vyhovovať potrebám zberateľov.

2.1 Existujúce riešenia

Tabuľky, či už databázové alebo softvérové, sú prvá vec čo človeku napadne, keď myslí na dáta a ich katalogizáciu. Od vzniku databáz (úložísk dát, viz. kapitola 3.3) sa stali tabuľky ich interpretáciou. Pre ľudí je prirodzené sa v nich orientovať. Aj tí, čo nie sú zdatní v práci s počítačom, vedia tabuľky ovládať intuitívne. Vizuálnu štruktúru poznajú z výpisov, katalógov a záznamových kníh.

Používateľské prostredia pre prácu s tabuľkami ponúkajú veľa užitočných funkcií – filtrovanie, zoradenie, vyhľadávanie. Niektoré majú dokonca vývojové prostredie a so znalosťou daného vývojového jazyka sú programovateľné. Toto riešenie ale vyžaduje určitú dávku kreativity a množstvo úprav počas doby používania, aby vytvorený systém fungoval efektívne.

Prezentácia dát v tabuľkových procesoroch je fádna. Výstupom je väčšinou text (v tabuľke) obohatený o grafy, farby. Vizualizácia entít v inej forme je takmer nemožná, prípadne vyžaduje pokročilú znalosť používateľského prostredia.

Okrem tabuľkových procesorov existujú aplikácie vytvorené na mieru pre konkrétne skupiny zberateľov. Príkladom môže byť internetová aplikácia *goodreads*[1], ktorá okrem iného ponúka katalogizáciu spolu s vyhľadávaním v celosvetovej databáze kníh. Aplikácia má mnoho funkcií, ktoré sú prispôsobené potrebám čitateľov. Zaujímavá je možnosť každoročného vytvorenia cieľa v počte prečítaných kníh, ktoré je možné zdieľať medzi priateľmi. Samozrejme, človek by si mohol vytvoriť hárok, v ktorom by si sám určil, aké atribúty by mohla mať entita *Kniha*, zapisovať si dané dáta a nad nimi vlastnoručne nastavovať grafy a štatistiky. Aplikácia *goodreads* však ponúka všetko to, čo tabuľkový procesor, a ešte k tomu mnoho funkcií naviac.

Navyše, každá kniha obsahuje titulnú stránku v podobe obrázku, čo by sa do tabuľky integrovalo veľmi nepohodlne. Zobrazenie entít v aplikácii ponúka omnoho viac možností

[1] https://goodreads.com

a každý pohľad na danú entitu a množiny entít môže mať vlastnú vizualizáciu. Detail knihy má inú štruktúru ako prehľad uložených kníh. Reprezentácia daných dát je lepšie pochopiteľná, využíva kontext a používateľ sa v aplikácii lepšie orientuje. Problémom je, že aplikácia je zameraná len na knihy.

Samozrejme, existujú ďalšie aplikácie zamerané na vinylové platne, izbové rastliny, topánky alebo komiksy. Problémom je, že existujú ľudia čo zbierajú veľmi špecifické predmety, a nemôžu pre svoje potreby využiť existujúce aplikácie. Zberatelia takýchto kúskov sú potom odkázaní na tabuľkový procesor, prípadne si dané dáta zapisujú na papier. Riešením nie je vytvoriť pre každý zberateľský predmet samostatnú aplikáciu, ale snažiť sa vytvoriť univerzálne prostredie, ktoré by mohli využiť všetci.

V rámci prieskumu som sa zamerala na aplikácie, ktoré nie sú obmedzené na jednu cieľovú skupinu. V nasledujúcich podkapitolách sú popísané tie najpoužívanejšie z nich.

2.1.1 Mobilná aplikácia *MyCollections*

Po zapnutí aplikácie sa zobrazí stránka, ktorú je možné vidieť na obrázku 2.1a. Aplikácii chýba vizuálne rozlíšenie jednotlivých sekcií a akcií, čo majú vyššiu váhu od tých menej dôležitých. Dizajn užívateľského zážitku (angl. *User Experience Design*, príp. *UX Design*) je v modernej dobe kľúčovým faktorom prilákania nových používateľov – v tejto aplikácii je implementovaný nesprávne. Aplikácia je neprehľadná, použitie ikon nevypovedá o ich úlohe. Testovaná bola iba mobilná aplikácia vyvinutá pre operačný systém *Android*, no *MyCollections* ponúka aplikáciu aj na operačný systém *Windows* a *iOS*.

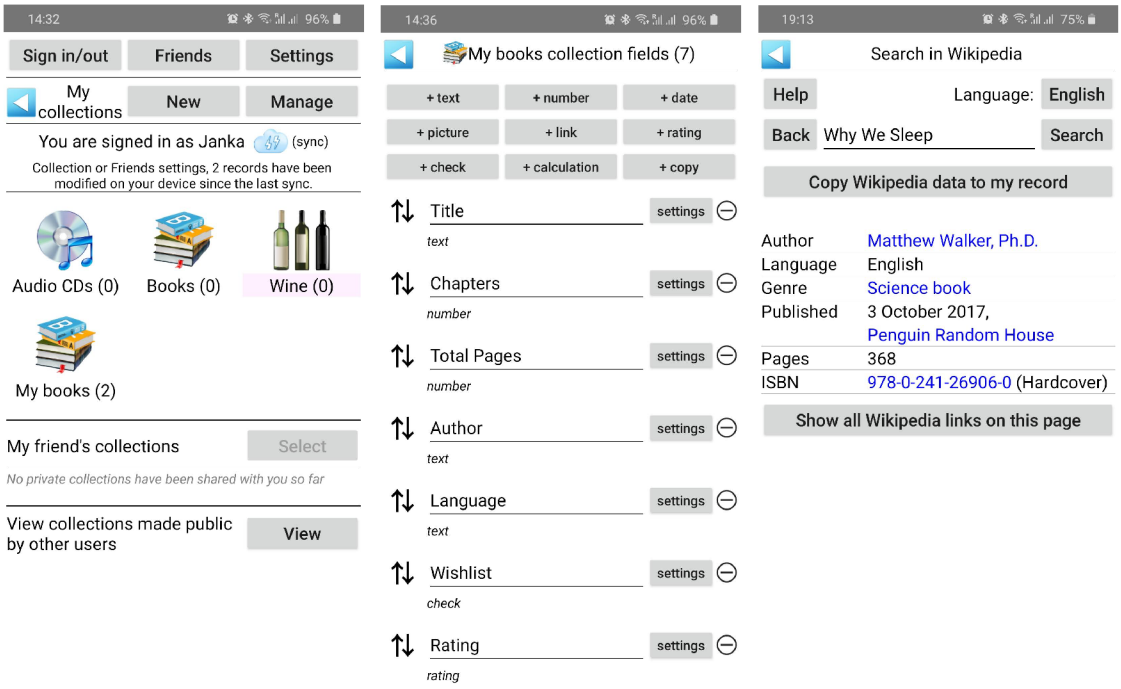
Odhliadnuc od zlého dizajnu má táto aplikácia zaujímavé funkcie. Aplikácia ponúka zálohovanie dát online, pričom má vlastné prihlasovanie (nie pomocou tretích strán). Je možné si v nej prechádzať kolekcie všetkých používateľov, ktorí ich zverejnili. Taktiež je možné si pridať priateľov podľa e-mailovej adresy a zdieľať s nimi priamo svoje súkromné zbierky.

Pri vytváraní kolekcie aplikácia ponúka množstvo šablón, ktoré majú prednastavenú ikonu a atribúty kolekcie. Na druhú stranu je možné špecifikovať úplne novú kolekciu a vytvoriť si vlastné atribúty rôznych typov (viz. obrázok 2.1b). Jednotlivé atribúty môžu byť povinné alebo voliteľné, ich hodnoty unikátne, je možné určiť ich skratku, ktorá sa zobrazí v hlavičke tabuľky. Atribút môže mať takisto predvolené hodnoty, ktoré si môže používateľ zvoliť pri vytváraní položky.

Chýba možnosť vytvorenia viacerých položiek naraz. Vytvorenie položiek je možné len jednotlivo po kliknutí na tlačidlo *Add*. Zobrazí sa stránka, kde je možné vyplniť hodnoty jednotlivých atribútov. Aplikácia taktiež predpokladá, že jednotlivé položky by používateľ mohol chcieť predať alebo požičať, a tak implementuje možnosť nastaviť predmet ako „požičaný“ alebo „na predaj“ spolu s kúpnou cenou.

Zaujímavá je funkcia vyhľadávania informácií o daných položkách pomocou stránok *Wikipedia*. Ak napríklad do tohto vyhľadávania zadáte názov knihy, zobrazia sa vám jej relevantné informácie, akými sú autor, jazyk, žáner a ISBN (viz. obrázok 2.1c). Nájdené hodnoty môžete priamo uložiť do vytváranej položky. Zároveň vám aplikácia ponúkne možnosť zobrazenia hypertextových odkazov nachádzajúcich sa na nájdenej stránke.

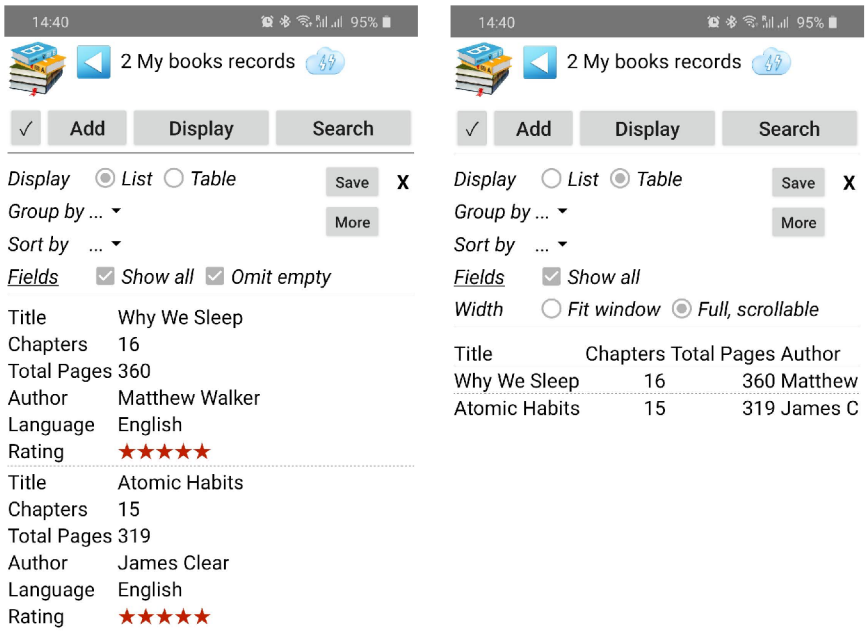
Vyhľadávaniu však chýba ukazovateľ postupu načítania, a tak je používateľ po dobu pár sekúnd zmätený, či sa niečo na pozadí deje alebo nie. Ak sa hľadaný výraz na stránke *Wikipedia* nenachádza, aplikácia to používateľovi taktiež neoznámi a zostane v stave, akoby dáta stále načítavala.



(a) Úvodná obrazovka (b) Nastavenie atribútov (c) *Wikipedia*

Obr. 2.1: Náhľad do aplikácie *MyCollections*

Vytvorené položky je možné zobraziť v zozname alebo tabuľke (viz. obrázky 2.2). Textu chýba formátovanie, hodnotám v tabuľkách chýba odsadenie, a tak sa text s číslicami miešajú. Aplikácia ponúka možnosť zoskupenia hodnôt podľa neobmedzeného množstva atribútov. Položky sa dajú triediť podľa viacerých faktorov naraz. Vyhľadávanie medzi položkami je rozdelené na „jednoduché“ a „pokročilé“, pričom druhá možnosť využíva *XPath* vyhľadávací jazyk.



(a) Položky v zozname (b) Položky v tabuľke

Obr. 2.2: Možnosti zobrazenia vytvorených položiek v aplikácii *MyCollections*

2.1.2 Mobilná aplikácia *My Collections*

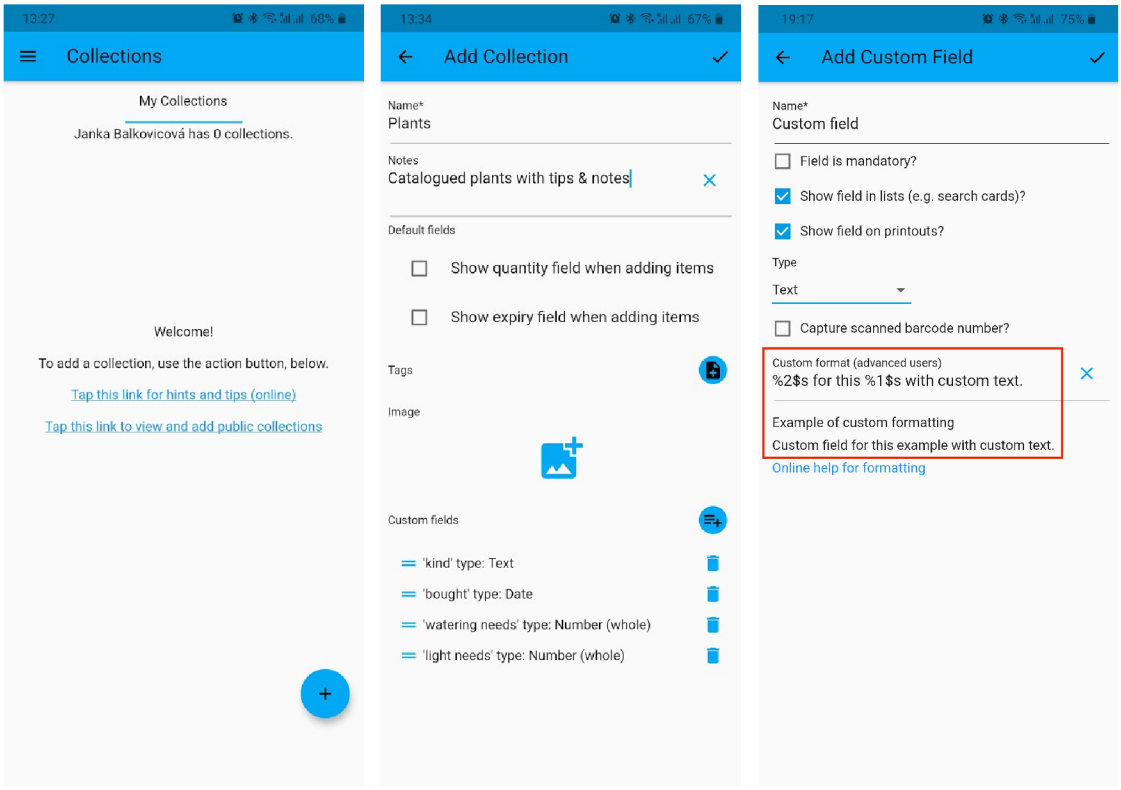
Táto aplikácia má názov podobný názvu aplikácie z predchádzajúcej kapitoly. Kvôli tomu si používatelia tieto aplikácie môžu mýliť.

Po spustení mobilnej aplikácie sa zobrazí stránka vyzývajúca na prihlásenie. Ak sa používateľ neprihlási, nemôže vytvárať alebo upravovať položky a kolekcie. Aplikácia využíva vlastné prihlasovanie, no existuje aj možnosť prihlásiť sa pomocou *Google* alebo *Apple* účtu.

Po prihlásení (viz. obrázok 2.3a) si používateľ môže vytvoriť kolekcie, prípadne prezerať a pridať verejne dostupné zbierky. Vytvorenie kolekcie (viz. obrázok 2.3b) ponúka možnosť pridania vlastných atribútov, zaujímavé sú voliteľné predvolené atribúty „množstvo“ a „dátum spotreby“. Tie naznačujú, že používatelia aplikáciu môžu využiť aj ako inventarizačný systém, napríklad skladu internetového obchodu.

Jednotlivé typy atribútov sú konfigurovateľné ako v aplikácii *MyCollections*, no škála úprav je menšia. Odlišuje sa aj možnosťou zaznamenania čiarového kódu ako samostatného atribútu. Nastaviť je možné aj vlastný formát hodnoty – predvolený formát je *názov atribútu: hodnota atribútu*, pričom používateľ si to môže upraviť do vlastného tvaru, napríklad pridať jednotky alebo text, ktorý bude zobrazený pri každej hodnote (viz. obrázok 2.3c).

Pridanie položky nie je ničím obohatené, zaujímavé je ale vytvorenie tzv. „kontajneru“, čo si môžeme predstaviť ako adresár vo vnútri kolekcie. Daný kontajner má rovnaké atribúty ako položky kolekcie a môže byť použitý na zoskupenie položiek, ktoré majú rovnaké vlastnosti. Rôzne zobrazenie vytvorených kontajnerov aj položiek je možné vidieť na obrázkoch 2.4a, 2.4b a 2.4c.



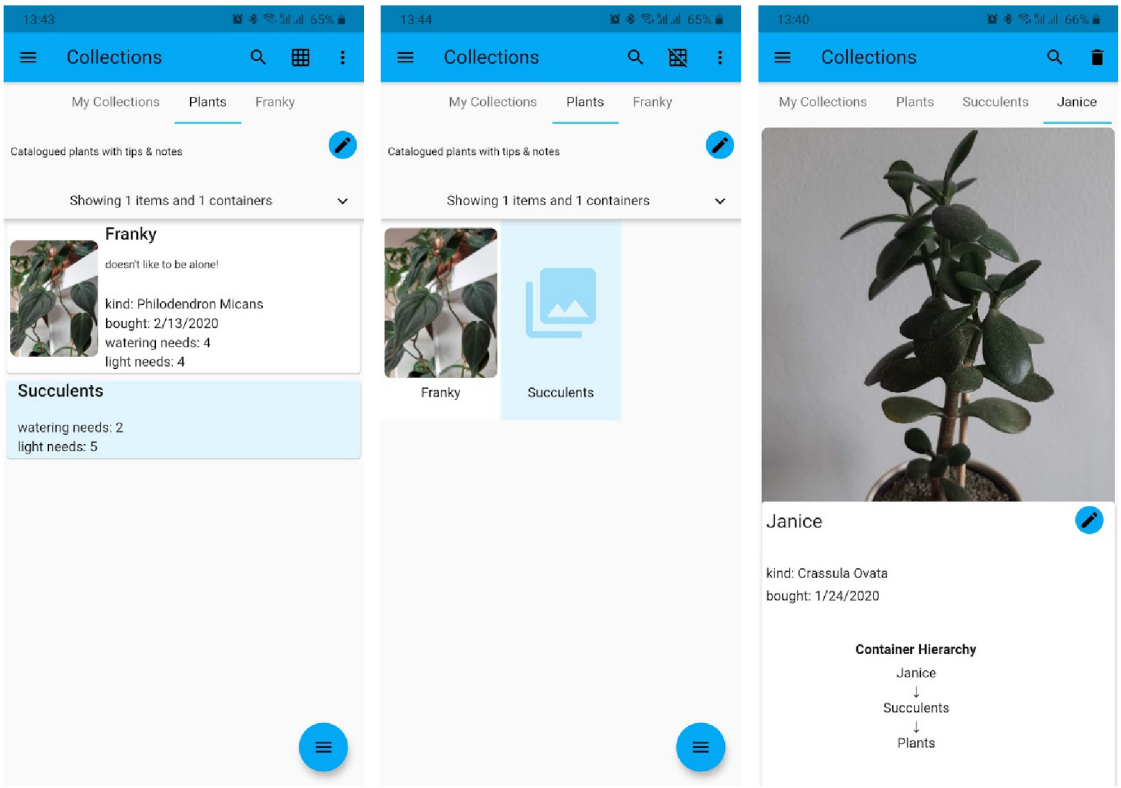
(a) Úvodná obrazovka (b) Vytvorenie kolekcie (c) Pridanie vlastného atribútu

Obr. 2.3: Náhľad do aplikácie *My Collections*

V detaile položky dominuje obrázok, pričom hodnoty atribútov nie sú nijak špeciálne formátované. Hierarchia kontajnerov je zbytočne zobrazená dvakrát v hornej a spodnej časti detailu (viz. obrázok 2.4c).

Vyhľadávanie je implementované ako filtrovanie položiek kolekcie na základe obsahu vyhľadávaného textu, pričom nie je možné špecifikovať žiadne ďalšie argumenty. Zobrazenie nájdených položiek sa aktualizuje v reálnom čase.

V tejto aplikácii sa kolekcie takisto rozdeľujú na súkromné a verejné. Prezeranie verejných kolekcií nie je intuitívne. Po kliknutí sa daná kolekcia odoberie zo zoznamu verejných kolekcií a presunie medzi používateľove zbierky. Používateľ následne musí opustiť zoznam verejných zbierok, a verejnú kolekciu si otvoriť z úvodnej obrazovky.



(a) Zobrazenie v zozname (b) Zobrazenie v galérii (c) Zobrazenie detailu

Obr. 2.4: Zobrazenie položiek a kontajnerov v aplikácii *My Collections*

2.1.3 Multiplatformová aplikácia *Sortly*

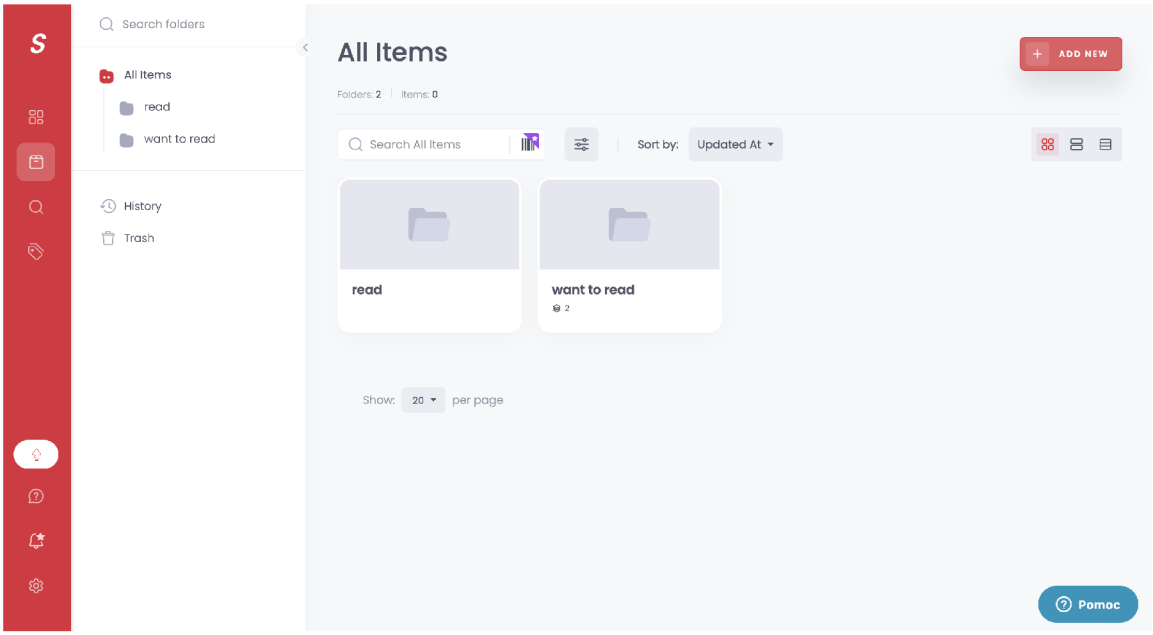
Aplikácia *Sortly* je dostupná na viacerých platformách – systémoch *Android*, *iOS* a internetovom prehliadači. Testovaná bola mobilná a webová aplikácia. Táto podkapitola sa bude venovať hlavne webovej aplikácii, keďže ponúka oveľa viac funkcií.

Už z prvého pohľadu je zrejmé, že aplikácia je prepracovaná, s moderným dizajnom a zaujímavými funkciami. Na úvodnej stránke webovej aplikácie[2] je zobrazené množstvo firiem, ktoré ju používajú. No z prezentácie danej aplikácie sa dá usúdiť, že *Sortly* je určená skôr na inventarizáciu, a nie na zberateľstvo.

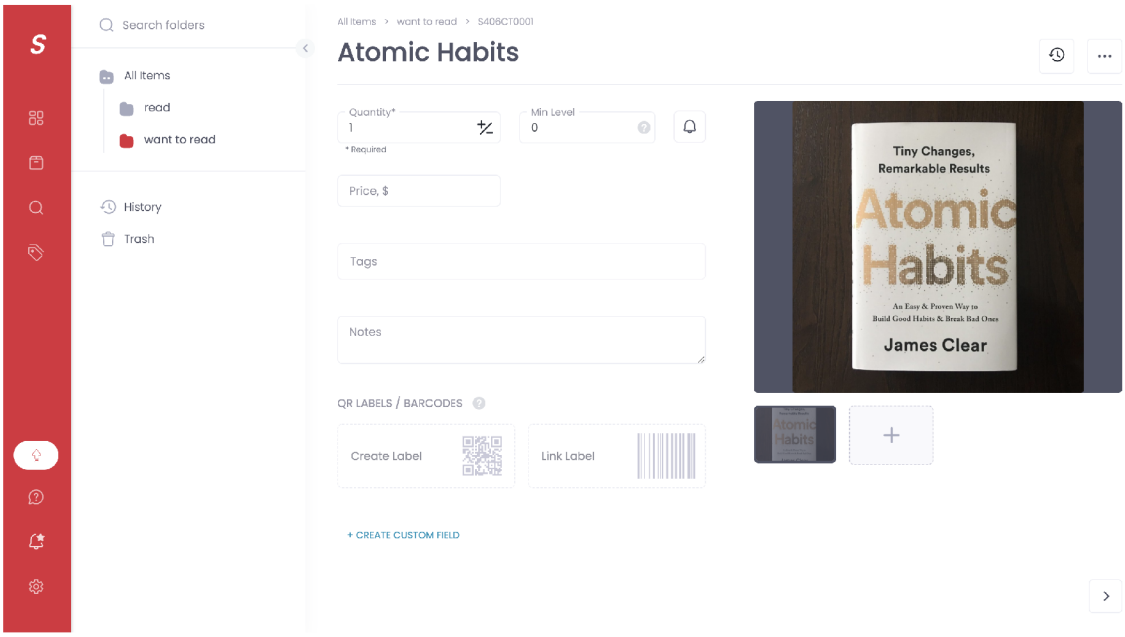
Po prihlásení je možné vidieť menu v dvoch úrovniach a prehľad adresárov a položiek (viz. obrázok 2.5). Adresáre aj položky sú zobrazené ako dlaždice, no zobrazenie sa dá zmeniť na zoznam alebo kompaktnú tabuľku.

Detail položky má formát rovnaký ako formulár vytvorenia novej položky. Ich priestor je vyvážený rovnomerne – polovicu stránky zaberá fotografia položky, druhú polovicu formulár na priamu úpravu danej položky (viz. obrázok 2.6). Každá položka má predvolené atribúty, ktorými sú „počet“, minimálny počet, ktorý indikuje že zásoby danej položky sú nízke, „cena“ a „poznámky“. Vytvorená položka má vygenerované jedinečné identifikačné číslo v rámci aplikácie *Sortly*, a je viditeľné v prehľade nad jej názvom. Zaujímavosťou je aj uloženie čiarového kódu alebo vytvorenie nového QR kódu, pomocou ktorých sa dajú položky jednoduchšie vyhľadávať.

[2] https://sortly.com/



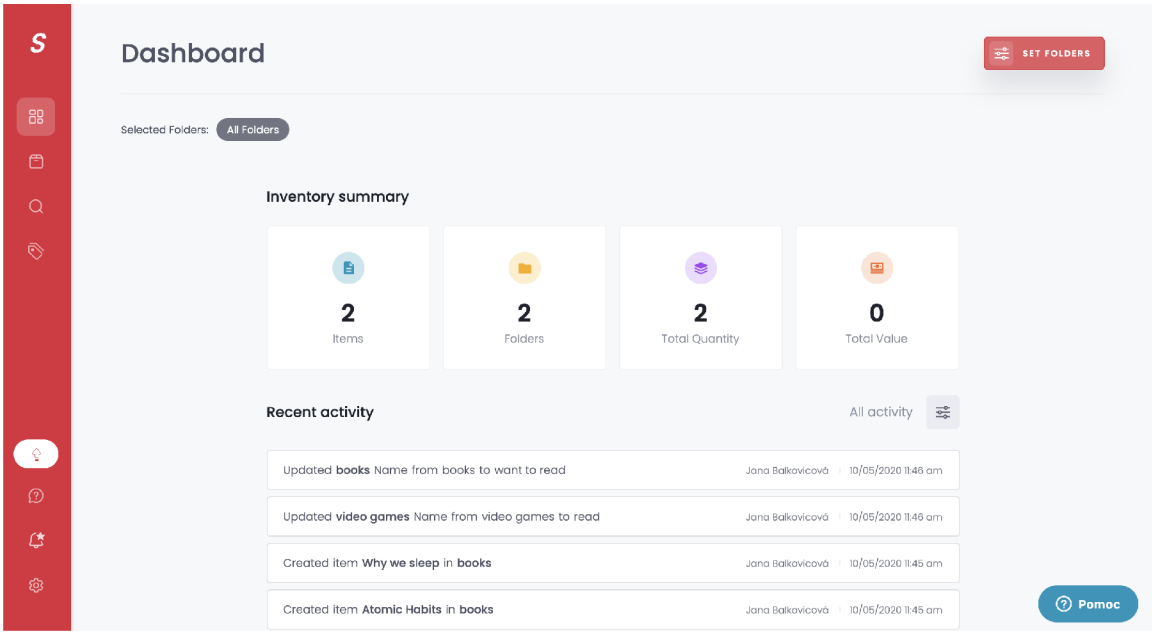
Obr. 2.5: Webová aplikácia *Sortly* po prihlásení



Obr. 2.6: Detail položky v aplikácii *Sortly*

Aplikácia poskytuje možnosť vytvorenia vlastných atribútov z predvolených typov. Škála typov je veľká, takisto ako výber z pár predvolených atribútov ako napríklad „dátum exspirácie“ alebo „sériové číslo“.

K dispozícii je pre používateľov aj „Nástenka“ (v orig. *Dashboard*, viz. obrázok 2.7), ktorá poskytuje prehľad používateľových položiek. Je možné vidieť ich počet a celkovú hodnotu, nedávnu aktivitu, nedávne otvorené položky a úrovne zásob. Tieto štatistiky je možné zobraziť aj pre ľubovoľnú podmnožinu adresárov.



Obr. 2.7: Nástenka v aplikácii *Sortly*

Vyhľadávanie vo webovej aplikácii je prepracované, položky je možné filtrovať pomocou kombinácie všetkých dostupných atribútov. Samozrejmosťou je aj možnosť zoradenia položiek.

Sortly je aplikácia pripravená na používanie v podniku. Používateľské prostredie je možné prispôsobiť dizajnu danej firmy, či už škálou farieb, alebo nahradenia viditeľného loga v menu. Mnoho komplexných funkcií má aplikácia dostupných len pre predplatiteľov, ako napríklad rôzne integrácie, prístup k API alebo spravovanie ľudí v tíme.

Zaujímavá je možnosť hromadného nahrania položiek zo súborov vo formáte *.csv* (*comma-separated values*, hodnoty oddelené čiarkami) alebo *.xlsx*, čo je formát programu Excel. Táto funkcionalita chýba v predošlých aplikáciách a patrí medzi tie, čo nevyžadujú predplatné.

2.2 Zhodnotenie

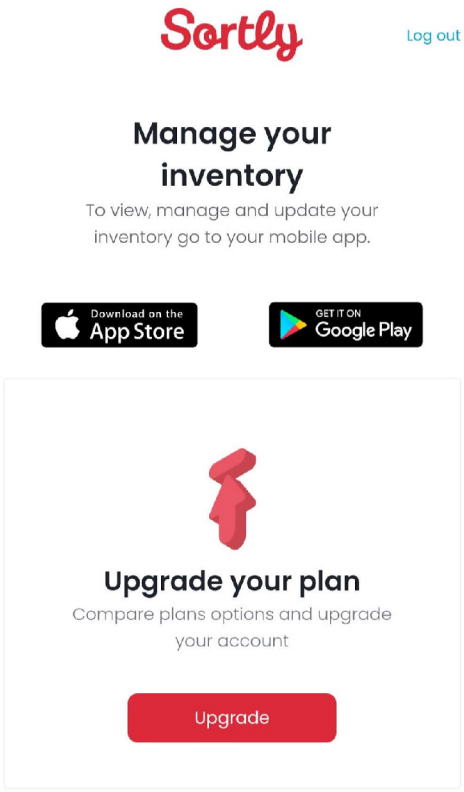
Pri hľadaní vhodných aplikácií pre zberateľov ma prekvapilo, že väčšina dostupných riešení je implementovaná pre operačný systém *Android*. Mobilné telefóny neponúkajú veľké zobrazovacie plochy. Aplikácie, ktoré zle naložia s týmto malým priestorom pri zobrazovaní prehľadu všetkých položiek, môžu mať problém, že sa dáta budú zlievať (viz. obr. 2.2b). Keďže dáta v kolekciách alebo zbierkach môžu mať veľa atribútov, je potrebné na to pri vývoji myslieť a vhodne navrhnúť celkový dizajn. Nedostatok a zlá organizácia priestoru ovplyvňuje celkový dojem z aplikácie. Preto je určite jednoduchšie začínať s väčším pries-

torom, ktorý ponúka napríklad počítačový monitor. Na ňom sa dá pohodlne vykresliť oveľa väčšia časť tabuliek a jednotlivých položiek bez dodatočných úprav.

Sú dva spôsoby, akými je možné vytvoriť aplikáciu, aby sa mohla spustiť na počítači. Buď bude implementovaná ako natívna aplikácia alebo ako webová aplikácia. Natívne aplikácie sú aplikácie, ktoré sú vytvorené na určitý operačný systém. Ak sa vývojári rozhodnú použiť natívne riešenie a chcú pokryť používateľov s operačným systémom *Linux*, *Windows* a *MacOS*, musia implementovať tri samostatné aplikácie. Druhý spôsob je z hľadiska vývoja a údržby softvéru efektívnejší. Aplikácia je dostupná prakticky na akomkoľvek zariadení, ktoré ponúka internetový prehliadač.

Čoraz viac webových aplikácií ponúka responzívny dizajn[3], vďaka ktorému je prehliadanie stránky na mobile alebo tablete oveľa príjemnejšie. Nie je teda potrebné vytvárať úplne novú aplikáciu na všetky platformy, stačí len navrhnúť takú webovú aplikáciu, aby bola dizajnom vhodná na menšie obrazovky.

Pomocou adaptívneho dizajnu je možné úplne vylúčiť niektoré funkcie alebo komponenty pre určité veľkosti obrazoviek. Ďalšou možnosťou ako vyriešiť problém malej obrazovky vo webovej aplikácii je odkázať sa priamo na existujúce riešenie v podobe mobilnej aplikácie. Príkladom je *Sortly*, viz. obrázok 2.8.



Obr. 2.8: Webová aplikácia *Sortly* po prihlásení na internetovom prehliadači v mobile

[3] *Responsive Web Design*, optimalizácia stránky pre rôzne druhy zariadení

Aplikácie pre zberateľov môžu obsahovať citlivé informácie, ktorých únik by mohol viesť k problémom. Ak má používateľ vo svojej zbierke predmety, ktoré majú vysokú hodnotu, malo by byť jeho rozhodnutím, či ich chce zverejniť alebo nie. Preto by vstup do aplikácie mal byť vždy zabezpečený prihlásením.

Čoraz viac aplikácií ponúka okrem vlastného prihlásenia možnosť použiť účet tretích strán (*Google*, *Facebook*, atď.). Ak si používateľ vytvorí nový účet v danej aplikácii, aplikácia je zodpovedná za bezpečné uloženie jeho prihlasovacích údajov. Naopak, keď používateľ využije už existujúci účet tretej strany, aplikácia nedostane jeho prihlasovacie údaje. Príjme len informáciu o tom, že používateľ je ten, za koho sa vydáva, pričom môže použiť dostupné dáta z daného účtu.

Veľké firmy ako *Google* a *Facebook* čelia veľkým bezpečnostným rizikám, keďže ich služby využívajú miliardy ľudí denne po celom svete[7, 9]. Malé firmy nemajú dostatok zdrojov na to, aby implementovali sofistikované riešenie na zabezpečenie ich systému. Preto je lepšie v tejto oblasti dôverovať internetovým gigantom. [17]

Navyše, *Google* aj *Facebook* ponúkajú možnosť nastavenia dvojstupňového overenia (angl. *two-factor authentication*, 2FA). Vďaka tomu sa používateľ nemusí spoliehať len na svoje prihlasovacie údaje – prvý stupeň overenia. V druhom stupni overovacieho procesu je využité väčšinou mobilné zariadenie, pomocou ktorého používateľ potvrdí, že sa naozaj pokúša prihlásiť pomocou daného účtu.

S prihlásením do aplikácie prichádza synchronizácia dát z online úložiska na úložisko daného zariadenia a opačne. Vďaka tomu je možné jednoducho rozdeliť kolekcie na verejné a súkromné, prípadne zdieľať kolekcie medzi priateľmi. Takýto prístup dáva možnosť vytvoreniu zberateľskej komunity, čo je dobré pre popularitu danej aplikácie a prináša prirodzený marketing.

Pri vytváraní kolekcií a zbierok je veľmi užitočné, keď v aplikácii už existuje aspoň pár vytvorených šablón s predvolenými atribútmi. Používateľovi to pomôže, aby sa rýchlejšie zorientoval v aplikácii. Aplikácia *MyCollections* má veľké množstvo rozmanitých šablón kolekcií, všetky sú veľmi detailne popísané atribútmi. Takisto má aplikácia najväčšiu rozmanitosť vo funkciách pri vytváraní vlastných atribútov. Veľká komplexnosť alebo voľnosť v takomto prípade môže priniesť zlú používateľskú skúsenosť (angl. *User Experience*, UX) v podobe nerozhodnosti a zmätku.

Všetky aplikácie ponúkajú štandardné typy atribútov akými sú text, celé číslo, desatinné číslo a dátum. Niektoré aplikácie obsahujú aj zaškrtávacie políčko alebo možnosť nastaviť množinu predvolených hodnôt. Nechýba ani možnosť k číslu pridať znak, menu či jednotku. Dôležitou funkcionalitou je znovu použiteľnosť atribútov nezávisle na kolekcii, v ktorej boli vytvorené. Niektorým aplikáciám nechýba zobrazovanie rôznych štatistík na základe zvolených atribútov, akou je napríklad *Sortly*.

Okrem kolekcií a položiek je niekedy možné nájsť aj adresáre, čo sú akési podmnožiny kolekcií. Niekedy je možné pridať jednotlivým adresárom aj atribúty. Aplikácia *My Collections* túto možnosť implementuje, avšak vytvoreniu položku v rámci takého adresára absentuje automatické vyplnenie daných hodnôt.

Kapitola 3

Informačný systém ako služba

Informačné systémy sú dnes súčasťou nášho každodenného života. Môžu byť komplexné (evidencia pacientov u lekára) alebo jednoduché (diskusné fórum). V minulosti sa informačné systémy implementovali „na mieru“ pre zákazníka, väčšinou ako počítačová aplikácia.[13] Dnes sa už čoraz viac stáva populárnejšia webová implementácia, ktorá pokrýva širšie publikum a dá sa použiť na každom zariadení, ktoré má internetový prehliadač.

V tejto kapitole si definujeme informačný systém, jeho históriu, vývoj webu a webových stránok, informačné systémy dostupné cez webové stránky, priblížime si ich rozhranie, popíšeme architektúru a uvedieme pár vecí, ktoré sa týkajú bezpečnosti na internete.

3.1 Informačné systémy a ich vývoj

Informačný systém má v modernom jazyku mnoho významov. Zamieňa sa za databázy, počítače, či komerčný Excel. Hoci tieto pojmy spĺňajú podmienky informačného systému, nevytvárajú kompletný a pravý obraz, čím všetkým informačný systém naozaj je.

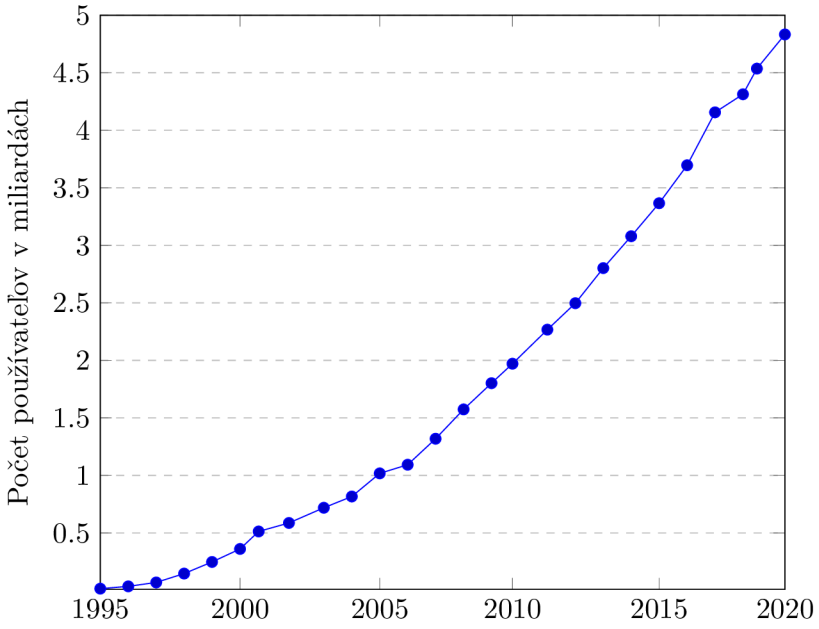
Viacero zdrojov[14, 16] hovorí o informačnom systéme ako o kombinácii hardvéru a softvéru, ktorí používajú ľudia na zber, vytváranie, distribúciu, analýzu a vizualizáciu užitočných dát, väčšinou v nejakej organizácii alebo za špecifickým účelom. Tieto definície sa zameriavajú na viacero spôsobov ako popísať informačný systém – vo všetkých sú ale dôležité komponenty, ktoré ho vytvárajú, takisto ako úloha, ktorú tieto systémy vykonávajú.

Ak sa pozrieme do minulosti na začiatok informačných systémov a internetu, uvidíme, ako veľmi rýchlo sa toto odvetvie vyvinulo v priebehu len pár desaťročí, a ako exponenciálne tento vývoj pokračuje. Na grafe 3.1 môžeme vidieť krivku používania internetu takmer od jeho začiatku.

Nasledujúce odseky ponúkajú prehľad informácií čerpaných z knihy *Information Systems Today*[14] a článku *Evolution of the World Wide Web*[15].

Začiatky sa odohrávajú v 50. rokoch minulého storočia, kedy počítače zaberali polovicu miestnosti. Počítač bol mechanizmus, ktorý vedelo ovládať len určité množstvo ľudí a naprogramovať ho bolo oveľa zložitejšie ako programovanie v súčasnosti. Pomocou neho človek vedel len vyriešiť konkrétny výpočet alebo rovnicu.

V priebehu ďalšieho desaťročia sa počítače stali dostupnejšími. Firma IBM bola dominantnou na trhu a poskytovala svoje počítače väčšine firiem. S operačným systémom sa stal počítač jednoduchším na ovládanie a začal sa vo veľkom používať na vytváranie zápisov,



Obr. 3.1: Používanie internetu v rokoch 1995–2020, dáta prevzaté z [12]

tabuliek a databáz. Tieto počítače ešte neboli pripojené na akúkoľvek sieť. Človek mal na počítači možnosť uložiť, vyhľadávať, upravovať a kategorizovať dáta.

Veľké firmy chceli využiť svoje dáta naplno, no ich zdieľanie bolo komplikované. Ak by firma aj mala viac počítačov, ich úložiská medzi sebou ešte nevedeli komunikovať, čo nebolo efektívne. Za týmto účelom vznikla architektúra klient-server. Počítače sa zapojili do lokálnej siete, pričom počítač – klient mal prístup k súborom a zdrojom na danej sieti aj napriek tomu, že ich nemal uložené na svojom pevnom disku. Jednoducho sa pripojil na počítač – server, ktorý toto zdieľané úložisko ponúkal. K dátam mala prístup celá organizácia v sieti.

V marci roku 1989 Tim Berners-Lee zverejnil *Information Management: A Proposal*[2], kde navrhol a popísal distribuovaný hypertextový informačný systém, ktorý mal spravovať veľké množstvo informácií v zložitých, vyvíjajúcich sa systémoch. Tento návrh je základným kameňom internetu ako ho poznáme dnes. Prvou fázou je preto hypertextový web (angl. *The Hypertext Web*).

Hoci mal *World Wide Web* slúžiť ako efektívna forma komunikácie medzi vedcami v CERNe a verejnosťou, nápad sa uchytil a generalizoval. Ako prvé boli na internete dostupné len jednoduché webové stránky, ktoré mali komunikovať novinky v internetovom svete[1]. Keďže mal tento koncept otvorenú licenciu, hneď ho začali využívať firmy na propagáciu svojich služieb a produktov. Tieto stránky boli pasívne, slúžili na doručovanie obsahu. Používateľ vedel vyhľadávať a čítať informácie, no nie dynamicky interagovať so stránkou. Stránky spravoval výlučne autor a iba on vedel upravovať ich text a podobu. Na spravovanie takýchto stránok museli mať autori vedomosti o tom, ako nastaviť server na sieti *World Wide Web*, vyriešiť problémy, ak sa nejaké vyskytli, a ovládať značkovací jazyk HTML (angl. *HyperText Markup Language*), v ktorom boli stránky napísané (a sú dodnes).

Po období statických stránok a zdieľaní informácií je centrom záujmu človek a vzájomná komunikácia. Po roku 2000 vznikli prvé webové stránky ako *Wikipedia* a *Blogger*, ktoré ponúkali používateľské prostredie bez potreby mať akékoľvek zručnosti s písaním HTML

[1] Jeden z prvých novinových článkov na internete bol vydaný v Januári roku 1992[3]

a spravovaním webových stránok a serverov. Ktokoľvek mohol pridávať a upravovať obsah, čo podporovalo vytváranie internetových komunít. Preto má táto fáza názov sociálny web (angl. *The Social Web*).

Internetové stránky začali mať flexibilný dizajn. Internet bol novou platformou so softvérom, ktorý sa dal spustiť na viacerých zariadeniach. Stále však existovali obmedzenia, na základe ktorých bolo šírenie informácií do určitej miery obmedzené.

Súčasnosť sa označuje pojmom „Post-PC“, keďže dôraz sa kladie na všetky ostatné technologické zariadenia. Mobilné telefóny, tablety, počítače v domácich zariadeniach a autách – to všetko v dnešnom svete môže obsahovať informačný systém a dokonca byť pripojený na internet. Dáta už nie sú vlastnené, ale zdieľané.

Nachádzame sa vo fáze sémantického webu (angl. *The Semantic Web*), ktorý je zameraný na kontext. Vyhľadávacie nástroje vedia čítať[2] internetové stránky a poskytovať používateľovi výsledky na základe polohy a iných, priebežne zbieraných dát. Internetovému trhu vládne reklama zameraná na personalizáciu a s ňou spojená umelá inteligencia.

Biznisu naďalej dominuje počítač (či už vo forme notebooku alebo klasického stolového počítača), no už nie je primárnym spôsobom pre komunikáciu medzi ľuďmi. Naopak, výpočtové zdroje počítača sa z daného počítača presúvajú na internet, takzvaný *cloud computing*.

3.2 Architektúra webových aplikácií

Webové aplikácie sú aplikácie založené na architektúre klient-server. Využívajú kombináciu skriptov na strane klienta a skriptov na strane servera, aby cez Internet prezentovali informácie používateľovi. Klient je internetový prehliadač, ktorý interpretuje zdrojový kód danej aplikácie a komunikuje so serverom pomocou URL a HTTP protokolu. Server môže byť pripojený na databázu, ktorá poskytuje dynamické úložisko dát [10].

Časť aplikácie, ktorú používateľ vidí a priamo ovplyvňuje, sa nazýva frontend. Je vytvorený zo zdrojového kódu, ktorý pozostáva z HTML, CSS a skriptovacieho jazyku JavaScript.

HTML (angl. *HyperText Markup Language*) je značkovací jazyk, ktorý slúži na vytvorenie kostry webovej stránky, pričom kladie dôraz na významovo správne prezentovanie informácií. Štandard aktuálnej verzie HTML5 oddeľuje štruktúru a obsah (nadpisy, odseky, citáty, tabuľky, atď.) od jeho výzoru (farba, písmo, atď.), ktorý sa presunul do kaskádových štýlov (angl. *Cascading Style Sheets*, CSS). Taktiež ponúka množstvo sémantických značiek, pomocou ktorých je možné ešte detailnejšie popísať obsah webovej stránky[18, 19].

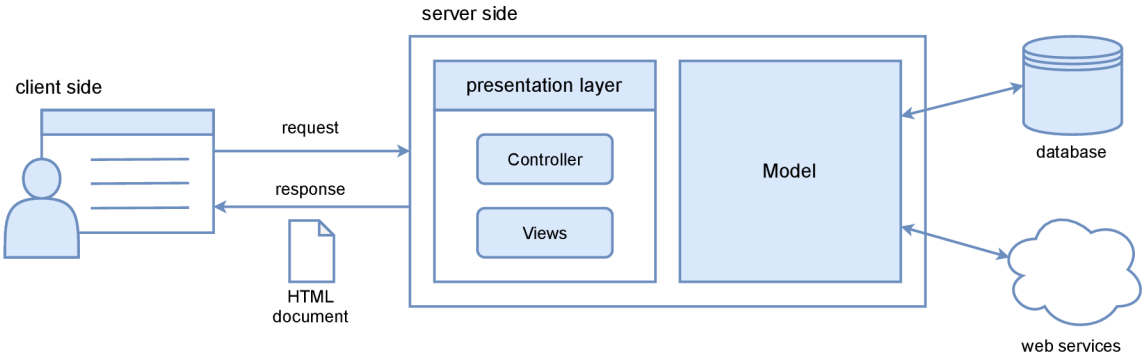
Kaskádovými štýlmi sa popisuje výzor elementov implementovaných v jazyku HTML. Tieto štýly je možné extrahovať do samostatných súborov, vďaka čomu sa zdrojový kód webovej stránky stáva prehľadnejším.

JavaScript je skriptovací jazyk, ktorý zlepšuje používateľské rozhranie a interakciu s dynamickými elementami webovej stránky. Keďže je interpretovaný priamo v prehliadači, nevyžaduje dodatočnú inštaláciu doplnkov alebo *runtime* prostredia[3]. Pred jazykom JavaScript bolo jedinou možnosťou vykresliť HTML stránky na strane servera – tzv. *server-side rendering*[15].

Pri implementovaní zložitej aplikácie môže byť zdrojový kód veľmi zle udržateľný, ak sa na jednom mieste mieša viacero jazykov a chýbajú mu konvencie. Za týmto účelom vzniklo viacero architektonických vzorov, ktoré ponúkajú návod, ako by mohli byť jednotlivé časti

[2] *web scraping*, získavanie dát z webových stránok
[3] prostredie, v ktorom je možné vykonávať kód

kódu oddelené podľa funkčnosti a výstupu. Jedným z nich je *Model-View-Controller* (skrátene MVC), ktorý rozdeľuje aplikáciu na tri funkčné bloky. Model má na starosti komunikáciu s databázou a inými zdrojmi, validáciu a obsahuje biznis logiku. View predstavuje HTML šablónu jedného pohľadu stránky, ktorá môže obsahovať minimálne množstvo logiky, na základe čoho sa v nej vykreslia alebo nevykreslia určité hodnoty alebo bloky. Controller prijíma požiadavky od používateľa, vyžaduje dáta z modelu a následne ich poskytuje *View* na vykreslenie, slúži ako prostredník. Pri generovaní výslednej stránky na strane servera sa pri každej požiadavke používateľa musí výsledná stránka prekresliť. Všetky funkčné časti MVC vzoru sú na strane servera (viz. obrázok 3.2)[5].



Obr. 3.2: *Server-side rendering* s použitím architektonického vzoru MVC

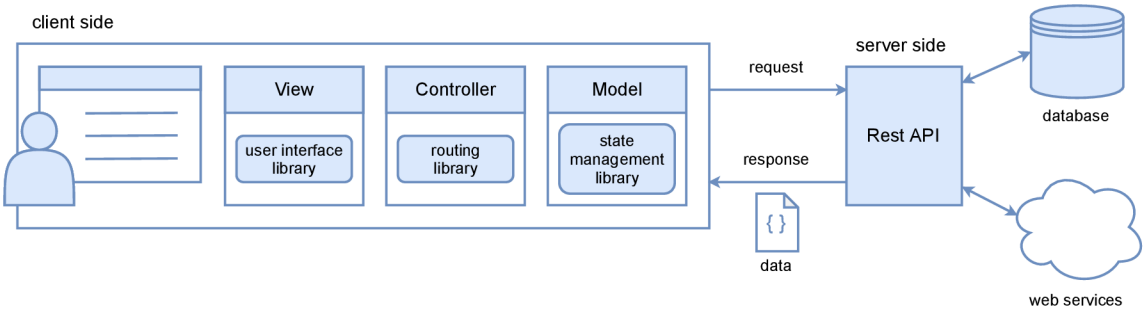
S pomocou jazyka JavaScript je možné časť tejto logiky presunúť na stranu klienta. Asynchrónna komunikácia medzi aplikáciou a serverom s použitím AJAX (angl. *Asynchronous JavaScript and XML*) a dynamická úprava objektového modelu dokumentu (angl. *Document Object Model*, DOM) v reálnom čase ponúkajú lepšiu používateľskú skúsenosť.

Tieto predpoklady sú taktiež nevyhnutné pre koncept *Single Page Application* (skrátene SPA), ktorý je založený na myšlienke, že celá aplikácia pozostáva z jednej webovej stránky, ktorá sa dizajnom a používateľským prostredím chce čo najviac priblížiť počítačovej aplikácii[15]. V SPA daná aplikácia načíta celú HTML stránku a jej zdroje pri prvej návšteve. Aplikácia je potom zodpovedná za vykresľovanie a zmeny používateľského prostredia, pričom môže prevziať časť validácie a iných procesov zo strany servera. Server už nie je potrebný v rozhodovaní, ako zobraziť dané dáta. Väčšinou poskytuje len surové dáta vo formáte JSON (angl. *JavaScript Object Notation*) alebo XML (angl. *Extensible Markup Language*) [25].

ReactJS[4] je jednou z mnohých knižníc jazyka JavaScript, ktoré boli vyvinuté na to, aby pomohli s implementáciou zložitých používateľských rozhraní. Táto knižnica predstavuje komponenty ako najmenšie funkčné časti aplikácie, ktoré môžu mať svoj stav. Namiesto definovania jedného modelu pre rozhrania, tieto rozhrania sú rozpadnuté do malých, znovu použiteľných komponent, čím sa zložitosť daného rozhrania znižuje. React ale nie je kompletný aplikačný rámec (angl. *application framework*), keďže neposkytuje dátový model a riadiacu logiku aplikácie. Slúži len ako prezentačná vrstva, teda *View* z architektúry MVC[26]. Pre dodržanie MVC vzoru je preto potrebné pridať riadiacu jednotku, ktorou

[4] https://reactjs.org/

môže byť *React Router*[5] a *Redux*[6] ako dátový model. Výsledná SPA aplikácia môže mať architektúru podobnú tej na obrázku 3.3.



Obr. 3.3: *Client-side rendering* s použitím architektonického vzoru MVC

Backend je časť aplikácie, ktorú používateľ nevidí a nevie s ňou priamo komunikovať. Jeho zdrojový kód už nemusí byť napísaný v jazyku JavaScript – je to samostatná jednotka, ktorá prijíma požiadavky a odosiela odpovede. Implementuje aplikačné rozhranie (angl. *application programming interface*, API), ktoré je súborom definícií a protokolov, ktoré určujú spôsob komunikácie medzi ním a aplikáciami.

Existuje viac typov API protokolov, no preferovaný štandard je dnes REST (angl. *Representational State Transfer*). Webová služba, ktorá spĺňa nasledujúce konvencie sa označuje ako *RESTful*.

Komunikácia prebieha cez internetový protokol HTTP (angl. *Hypertext Transfer Protocol*)[8] alebo jeho zabezpečené rozšírenie HTTPS (angl. *Hypertext Transfer Protocol Secure*), ktoré slúži pre prenos hypertextových dokumentov a ďalších informácií.

Požiadavka obsahuje HTTP metódu, URL (angl. *Uniform Resource Locator*) daného zdroja na internete a hlavičku s informáciami napríklad o prehliadači alebo v akom formáte sa majú vrátiť dáta v odpovedi. Na základe konkrétnej metódy potom môže obsahovať telo s dátami. [6]

Vlastnosti HTTP metód sú implementované na strane servera pomocou CRUD operácií (angl. *Create-Read-Update-Delete*). Odpoveď zo servera obsahuje stavový kód s jednoduchým hlásením, ktoré opisuje význam kódu a hlavičky s informáciami napríklad o serveri alebo dátume spracovania požiadavky. Takisto môže obsahovať telo s dátami alebo ďalšími informáciami. Stavové kódy sú rozdelené do piatich hlavných kategórií: informačné, úspešné, presmerovanie, chyba klienta a chyba servera.

Aj napriek tomu, že v SPA aplikácii sa väčšina logiky presúva na klienta, server stále zohráva dôležitú úlohu a mal by implementovať druhú vrstvu validácie. Medzi najčastejšie zodpovednosti servera je overenie prijatých údajov, autorizácia a autentifikácia klienta.

[5] https://reactrouter.com/
[6] https://redux.js.org/

3.3 Databázové systémy

Nasledujúca podkapitola popisuje databázy a databázové systémy, pričom rozlišuje dva základné typy – relačné a *NoSQL* databázové systémy.

V knihe *Database Systems Design, Implementation and Management*[4] sa píše: „Fakty sú surové dáta, ktoré nenesú žiadnu informáciu, zhodnotenie alebo odpovede na otázky. Sú to hodnoty, ktoré musia byť zaradené do určitého kontextu, aby nadobudli význam a stali sa informáciou."

Databáza je súbor dát, ktoré spolu navzájom súvisia. Jej štruktúra nemôže byť náhodná, ale pevne daná a musí mať prirodzený význam. Databáza je navrhnutá, vytvorená a naplnená s dátami ktoré majú vopred daný účel. Reprezentuje aspekty nejakej časti reálneho sveta, pričom všetky zmeny v tejto časti sa odzrkadlia v databáze. Spolu s konkrétnymi dátami obsahuje aj dodatočné informácie o týchto dátach a definíciu svojej štruktúry, čo sa spoločne nazýva metadáta. Databáza je uložená v súboroch na disku, no pristúpiť k jej údajom je možné jedine cez systém riadenia báze dát.

Systém riadenia báze dát, skrátene SRBD (angl. *Database Management System*, DBMS) je súbor programov, ktoré umožňujú používateľom vytvoriť, udržiavať a pristupovať k súborom databázy. Pomocou neho je možné databázu vytvoriť na špecifikovanom úložisku, ktoré je pod správou SRBD, vyhľadávať v databáze, modifikovať jej dáta a vytvárať správy o týchto dátach. Takisto implementuje transakčné spracovanie operácií, ktoré musí mať vlastnosti, ktoré majú skratku ACID (angl. *Atomicity, Consistency Preservation, Isolation, Durability*). Transakcia je najmenšia jednotka spracovania – je vykonaná v celku, alebo vôbec. Spracovanie transakcie nemôže dostať databázu do nekonzistentného stavu a transakcia je navonok izolovaná, takže do jej spracovania nemôže zasiahnuť iná transakcia. |meny vykonané potvrdenou transakciou sú trvalé a uložené v databáze

Každý SRBD musí mať systémovú ochranu proti poruche technického vybavenia alebo softvéru, a ochranu proti neoprávneným vstupom alebo vírusom. Navyše musí mať schopnosť prispôsobiť sa prípadným novým požiadavkám, ktoré sa postupom času vyvíjajú.

Databáza a SRBD softvér spolu tvoria databázový systém. [24]

Relačné databázové systémy majú stále veľkú komunitu zástancov. Od svojho vzniku v 70. rokoch sa relačný model zväčša nezmenil, čo hovorí o dobre premyslenom a fungujúcom systéme.

Relácia v databáze predstavuje tabuľku, kde každý riadok tvorí záznam logicky príbuzných hodnôt. Jeden riadok tabuľky reprezentuje fakt, ktorý odpovedá skutočnej realite alebo vzťahu. Názvy jednotlivých stĺpcov a celej tabuľky sú popisy, ktoré majú správne interpretovať hodnoty uvedené v riadkoch.

Aby bolo možné vytvárať vzťahy medzi riadkami jednotlivých tabuliek, napríklad definovať vzťah študenta k svojmu projektu medzi tabuľkami `študent` a `projekt`, musí existovať možnosť jednoznačne identifikovať jednotlivé riadky. V relačnom databázovom modeli sa tento problém rieši pomocou kľúčov. Primárny kľúč slúži ako identifikátor záznamu v rámci celej tabuľky. Jeho hodnota sa potom uvedie v druhej tabuľke ako cudzí kľúč. Ten predstavuje odkaz na daný záznam a zároveň vytvára vzťah medzi danými tabuľkami.

To, že primárny kľúč záznamu musí mať unikátnu hodnotu, je jedným z integritných obmedzení tabuliek. Tieto obmedzenia majú za účelom zabrániť narušeniu štruktúry a integrity databázy. Všeobecné integritné obmedzenia definujú pravidlá ako napríklad nutnosť

existencie primárneho kľúča[7] pre každý záznam. Patrí sem aj referenčná integrita, ktorá kontroluje, že cudzí kľúč odkazuje na existujúci záznam, prípadne má hodnotu `NULL`. Špecifické integritné obmedzenia sú súčasťou definície databázy a slúžia ako detailný popis štruktúry alebo obsahu vkladaných dát. Majú zamedziť situáciám, kedy by používateľ do databázy vložil niečo iné ako platný údaj. Neplatným údajom v stĺpci `vek` v tabuľke `osoba` by bolo napríklad záporné číslo, prípadne hodnota s písmenami.

Integritné obmedzenia je možno kontrolovať aj pomocou procedúr s názvom `trigger`. Ide o komplexnejší spôsob kontroly, ktorá sa bude vykonávať vždy, keď sa používateľ rozhodne zapisovať dáta do databázy.

Ak dáta spĺňajú všetky integritné obmedzenia, sú konzistentné. [24]

Moderné webové technológie a priemysel generujú obrovské množstvo dát, ktoré sú priveľké na to, aby boli spracované tradičnými nástrojmi na spracovanie údajov.

Pojmom veľké dáta (angl. *Big Data*) sa označuje masívna a komplexná množina údajov, ktorá môže, ale nemusí byť štruktúrovaná, a používa sa na analýzu dát na vyššej úrovni.

Relačné databázové systémy boli navrhnuté v ére, ktorá mala oveľa menšie nároky na úložisko, hardvér a výkon. Dnes preto výkonom a rozšíriteľnosťou málokedy stačia na dáta obrovských rozmerov. [11]

Výpočtové zdroje ako napríklad siete, servery, úložisko, aplikácie a webové služby vytvárajú jeden prepojený zdroj, ktorý je dostupný ako služba na požiadanie (angl. *on-demand service*). Táto výpočtová paradigma sa nazýva *cloud computing*. Používateľ pri tejto službe platí iba za využitie konkrétneho zdroja. Takto sa používateľom znižuje cena za prevádzkovanie svojej služby a počiatočné investície do výpočtových zdrojov, keďže všetky zdroje sú uložené na úložisku poskytovateľa, sú ľahko rozšíriteľné a nevyžadujú žiadnu údržbu. Mnoho aplikácií sa kvôli tomu presunulo na cloud prostredie, čím prispievajú k zväčšovaniu veľkých dát. Cloud prostredia a hlavne systémy na správu údajov nachádzajúce sa na cloud prostredí prichádzajú s novými požiadavkami:

- vysoký výkon a rozšíriteľnosť (angl. *scalability*), keďže aplikácie generujú čím ďalej, tým viac dát, ktoré je potrebné uložiť, spracovať a poskytnúť,
- elasticita, pretože aplikácie v cloud prostredí sú vystavené veľkým výkyvom v používateľnosti,
- schopnosť bežať na akomkoľvek úložisku, keďže väčšina cloud prostredí je založená na množine druhovo rôznorodých serverov, ktoré sa využívajú paralelne,
- odolnosť voči chybám, kvôli druhovej rôznorodosti serverov, ktoré sú náchylnejšie na poruchu ako „high-end“ servery,
- funkcie zabezpečenia a ochrany súkromia, keďže dáta budú uložené na úložiskách tretej strany, ktoré môžu byť zdieľané medzi používateľmi cloud prostredí,
- dostupnosť, pretože všetky kritické časti aplikácie sú taktiež presunuté do cloud prostredia a nemôžu si dovoliť žiadne výpadky.

Existujúce riešenia v podobe relačných databázových systémov čelia mnohým problémom, aby splnili vyššie spomenuté požiadavky pri nasadení na cloud prostredia. Preto

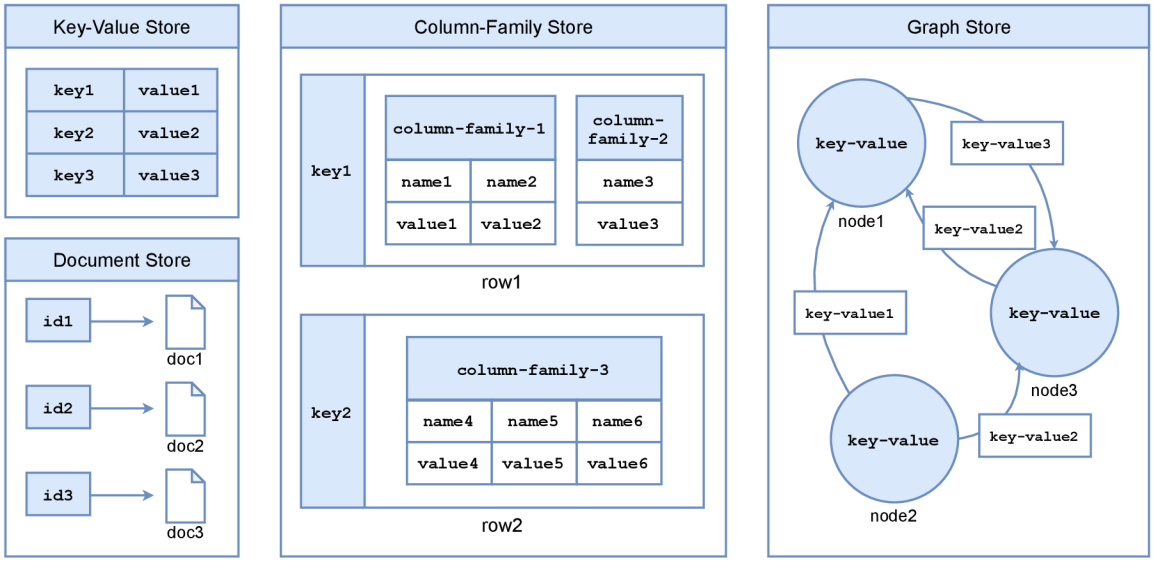
[7] nesmie mu chýbať hodnota

vznikli nové riešenia, ktoré tieto problémy adresujú. Nazývajú sa *NoSQL* a *NewSQL* dátové úložiská, pričom sa prezentujú ako alternatívy k tradičným nástrojom na spracovanie údajov, ktoré ale dokážu zvládnuť obrovské množstvo dát a sú jednoducho rozšíriteľné. [11]

NoSQL databázový systém je *open-source* softvér[8], ktorý je distribuovaný a nie je založený na relačnom vzťahu ako relačná databáza. Pojem NoSQL znamená „Not only SQL“, čo zdôrazňuje, že SQL nemusí byť jediným dopytovacím jazykom databázových systémov.

NoSQL databázy sú rôznorodé, niektoré vytvorené pre špecifické aplikácie. Od tradičných databáz sa líšia tým, že väčšinou nepodporujú ACID vlastnosti. Sú označované ako systémy, ktoré majú vlastnosti BASE (angl. *Basically Available, Soft-state, Eventually consistent*).

NoSQL databázy sa môžu rozdeliť do štyroch hlavných skupín podľa svojho dátového modelu na typy kľúč-hodnota, stĺpcové, dokumentové a grafové (viz. obrázok 3.4). [11]



Obr. 3.4: Znázornenie typov NoSQL databáz, prevzaté z [11]

Dátový model kľúč-hodnota (angl. *key-value store*) je jednoduchý, založený na pároch tvorených z kľúča a hodnoty, ktoré pripomínajú asociatívnu[9] mapu alebo slovník. Kľúč jednoznačne identifikuje príslušnú hodnotu a slúži na získanie hodnoty z úložiska. Keďže tento model nepotrebuje žiadnu schému, hodnoty môžu byť akéhokoľvek typu. Pri vyhľadávaní sa ale jednotlivé hodnoty prvkov použiť nedajú, na dopytovanie musí byť použitý jedine kľúč prvku. Tento dátový model neposkytuje žiadne vzťahy a štruktúry medzi prvkami, preto ak sú potrebné, musia byť implementované v aplikácii, ktorá pristupuje k tomuto úložisku.

Stĺpcové databázy (angl. *column-family store*) majú dáta uložené v skupinách stĺpcov namiesto riadkov. Množina dát pozostáva z niekoľkých riadkov, pričom každý má priradený jedinečný kľúč nazvaný primárny kľúč, tak ako relačné databázy. Každý riadok následne obsahuje niekoľko skupín stĺpcov, ktoré sa môžu líšiť v jednotlivých riadkoch. Tieto skupiny sú taktiež jednoznačne označené kľúčom a obsahujú jeden alebo viacero stĺpcov, kde každý stĺpec pozostáva z párov názov-hodnota. Stĺpcové úložiská sú účinnejšie v indexovaní a dopytovaní, keďže implementujú identifikátory aj v riadkoch aj v stĺpcoch, teda horizontálne

[8] softvér, ktorého zdrojový kód je verejne dostupný
[9] vlastnosť, kedy nezáleží na poradí prvkov

aj vertikálne. Podobajú sa na model typu kľúč-hodnota, keďže nepoznajú vzťahy medzi prvkami.

Dátový model dokumentového úložiska (angl. *document store*) je podobný úložisku kľúč-hodnota. Takisto používa kľúč na vyhľadanie prvkov, ktoré sa v tomto úložisku nazývajú dokumenty. Dokumenty sú väčšinou reprezentované formátom JSON alebo jeho odvodeniny ako *Binary JSON* (skrátene BSON). Jednotlivé dokumenty môžu obsahovať dáta komplexnej štruktúry, keďže podporuje vnorené entity a nemusia mať rovnakú schému. Tieto entity sa môžu zoskupovať do kolekcií, pričom v jednej kolekcii musia byť kľúče dokumentov unikátne. Aj napriek tomu, že vnútri kolekcie môžu mať jednotlivé dokumenty odlišnú štruktúru, úložisko umožňuje vyhľadávanie a indexovanie na základe hodnôt v dokumente.

Grafové úložisko (angl. *graph store*) je vytvorené na základe teórie grafov a používa graf ako svoj dátový model. Graf je matematická štruktúra definovaná množinou vrcholov (uzlov) a hrán, ktoré spájajú tieto vrcholy. Grafové databázy používajú úplne odlišný model ako predchádzajúce úložiská, kvôli čomu sú veľmi efektívne v ukladaní vzťahov medzi jednotlivými dátovými uzlami. Sú špecializované na zaobchádzanie s komplexne prepojenými dátami, tým pádom sú veľmi efektívne v prechádzaní a vyhľadávaní vo vzťahoch medzi jednotlivými entitami. Môžu byť využité na rozpoznávanie vzorov, analýzu závislostí alebo odporúčacie systémy[10]. Grafové databázy ale nemusia byť veľmi dobre rozšíriteľné, ak sú súvisiace uzly uložené na odlišných serveroch. Prechádzanie viacerých serverov totiž nie je veľmi výkonne efektívne. [11]

NewSQL databázy sa snažia vyriešiť problémy horizontálnej rozšíriteľnosti a odolnosti proti chybám, čo relačným databázovým systémom chýba. Všetky tieto databázy podporujú relačný databázový model a používajú SQL ako dopytovací jazyk, hoci sú založené na inej architektúre ako relačné databázy. Ich transakcie môžu upravovať viac ako jeden objekt, pričom sú vysoko konzistentné. Sú vhodné ako úložisko pre dáta, ktoré majú vopred definovanú štruktúru, ktorá sa pravdepodobne nebude meniť. NewSQL úložiská nájdu využitie napríklad vo finančnej sfére, kde operácie ako prevod peňazí musia automaticky aktualizovať oba účty naraz a všetky aplikácie musia mať rovnaký pohľad na databázu. [11]

3.4 Problémy komunikácie na internete

Webové aplikácie sa snažia poskytnúť používateľom čo najjednoduchší prístup k dátam kedykoľvek a z akéhokoľvek zariadenia čo poskytuje webový prehliadač. Tieto aplikácie môžu byť rôzneho charakteru a obsahovať menej alebo viac dôverné informácie. Všetky by ale mali zabrániť neoprávneným vstupom do aplikácie, zabezpečiť komunikáciu a ochrániť prihlasovacie údaje osôb spolu s ich osobnými dátami.

Prístup do webovej aplikácie je najčastejšie zabezpečený autentifikáciou, čo je proces, počas ktorého je verifikovaná identita daného používateľa. Používateľ sa preukazuje vopred dohodnutým dôkazom o svojej identite, napríklad prihlasovacím menom a heslom, biometrickými údajmi alebo rôznymi hardvérovými zariadeniami ako sú čipy a karty.

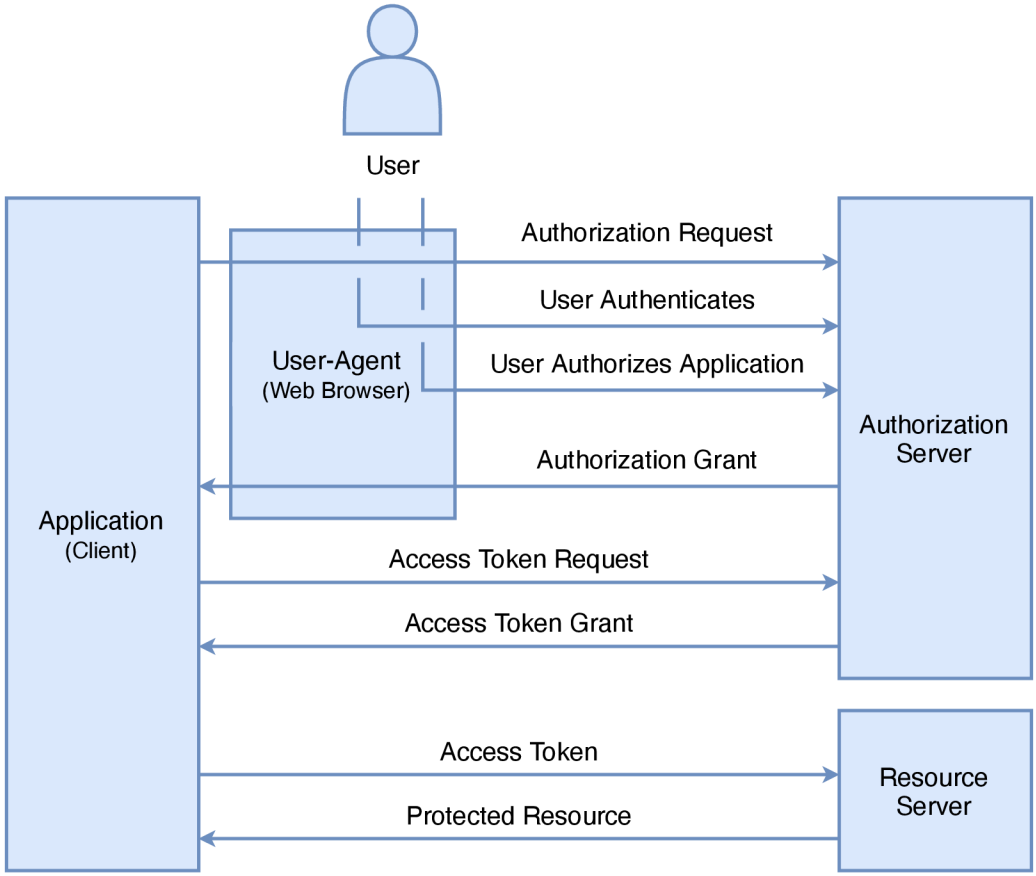
Pri autentifikácii *service-to-service* sa považuje za najbezpečnejšie, ak webová aplikácia nedostane prístup k používateľovmu heslu. Používateľ sa reálne prihlasuje pomocou účtu v inej službe, pričom aplikácia dostane len potvrdenie, či bol daný používateľ autentifikovaný. Poskytovatelia týchto autentifikačných služieb je mnoho, preto je dobré overiť si dôveryhodnosť daného poskytovateľa. Taktiež je vhodné zvoliť poskytovateľa, ktorý ponúka

[10]Systémy, ktoré využívajú vzory chovania užívateľov, aby im mohli ponúknuť to, čo skutočne preferujú.

viacstupňové overenie, kedy k úspešnej autentifikácii nestačí len použiť správne prihlasovacie údaje. Používateľ je následne vyzvaný k dodatočnej akcii napríklad na mobilnom zariadení, kde potvrdí svoje prihlásenie v aplikácii poskytovateľa, alebo opíše kód z SMS.

Proces autentifikácie je súčasťou autorizácie, ktorá overuje oprávnenia používateľov na prácu s dátami. Najznámejšie protokoly, ktoré sa používajú pri autorizácii a považujú sa za bezpečné, sú *OAuth 2.0* a *JSON Web Token* (skrátene JWT)[11]. Pri použití OAuth je nutné najprv danú aplikáciu zaregistrovať na stránkach poskytovateľa, keďže identita aplikácie je taktiež overovaná počas autorizácie.

OAuth autentifikácia prebieha v niekoľkých krokoch. Aplikácia najprv požiada používateľa o autorizáciu, ktorú vyžaduje na prístup ku chráneným zdrojom. Používateľ sa autentifikuje. Ak používateľ požiadavku aplikácie potvrdí, aplikácia bude autorizovaná. Aplikácia následne odošle požiadavku, ktorá obsahuje potvrdenie od používateľa a autentifikačné údaje aplikácie, na autorizačný server. Ak je identita aplikácie potvrdená a autorizácia od používateľa stále platná, autorizácia je dokončená a aplikácia dostáva prístupový token. Aplikácia potom požiada server o chránené zdroje, požiadavka už obsahuje prístupový token. Ak je prístupový token platný, server pošle dané zdroje aplikácii. Proces je prehľadnejšie znázornený na obrázku č. 3.5.



Obr. 3.5: Autorizácia pomocou rozhrania OAuth 2.0

Prístupový token obsahuje zabezpečené údaje o relácii (angl. *session*), identifikuje používateľa, jeho rolu a s ňou spojené práva. Posiela sa v hlavičke požiadavky a môže mať

[11] https://jwt.io/

rôznu formu. V protokole OAuth sa využíva prístupový token vo formáte JWT, ktorý je jeden z najbezpečnejších v súčasnosti. [1]

Pri zabezpečení aplikácií voči bezpečnostným rizikám je dobré sa riadiť overenými postupmi, ktoré sú účinné. Útočníci môžu použiť rôzne cesty aby sa dostali k citlivým informáciám alebo napáchali škodu. Tieto cesty môžu byť očividné alebo naopak skryté veľmi hlboko v aplikácii.

Open Web Application Security Project (skrátene OWASP)[12] je nezisková organizácia, ktorá sa zaoberá bezpečnosťou aplikácií a svoje zastúpenie má aj v Českej republike. V roku 2017 vytvorila dokument povedomia o najväčších rizikách dnešných webových aplikácií [22]. V tejto podkapitole sú popísané dve najvážnejšie z nich, ktoré z daného dokumentu vychádzajú.

Pojem *injection* predstavuje situáciu, pri ktorej používateľ aplikácie pri zadaní špecifického zdrojového kódu do nejakého vstupného poľa v aplikácii dokáže poškodiť alebo ohroziť daný systém.

Najzraniteľnejšie časti aplikácie sú často miesta, kde sa spracovávajú požiadavky do SQL alebo NoSQL databáz, prípadne vykonávajú skripty, ktoré závisia na používateľovom vstupe. Vstupné polia v aplikácii by preto mali byť ošetrené validáciou. Získané dáta by nikdy nemali byť použité priamo, ale mali by byť vhodne spracované.

Nasledujúci úryvok z kódu je považovaný za zraniteľný:

```
const query = "SELECT *
              FROM accounts
              WHERE custID='" + request.getParameter("id") + "'";
```

Ak používateľ upraví `id` danej požiadavky do určitého tvaru, výsledná SQL požiadavka bude vyzerať následovne:

```
http://example.com/app/accountView?id=' or '1'='1

SELECT *
FROM accounts
WHERE custID='' or '1'='1'
```

Namiesto vrátenia jedného výsledku požiadavka vráti všetky záznamy v danej `accounts` tabuľke, čím poskytuje útočníkovi prístup k citlivým informáciám. Komplexnejšie útoky môžu upraviť alebo zmazať dáta, prípadne vyvolať uložené SQL procedúry.

Autorizácia je ďalším procesom, ktorý môže viesť k ukradnutiu identity alebo zverejneniu vysoko citlivých informácií, ak je implementovaná nesprávne.

Útočníci väčšinou používajú automatizované procesy, s ktorými sa snažia dostať do systému pomocou ukradnutej identity.

Z registračného formulára alebo z formulára na obnovenie hesla môžu útočníci vydolovať zoznam používateľských mien len tým, že budú zadávať rôzne mená a systém ich upozorní, ak je nejaké používateľské meno obsadené. Nápovedou je aj čas, ktorý daná požiadavka spotrebuje. Pri úspešnej požiadavke je čas väčšinou až dvojnásobne kratší ako pri neúspešnej. Tento útok sa nazýva *account enumeration attack*. [23]

Zoznamy ukradnutých prihlasovacích údajov sú následkom iného útoku – tzv. *credential cracking*, počas ktorého majú útočníci prihlasovacie mená účtov, no nevedia heslá. Na nájdenie odpovedajúceho hesla použijú zoznamy najčastejšie používaných hesiel alebo generátory náhodných údajov a metódou pokus-omyl skúšajú všetky možné kombinácie. Pri

[12] https://owasp.org/

tomto automatizovanom kontrolovaní rôznych kombinácií prihlasovacieho mena a hesla je veľkou nápovedou pre útočníka, ak server odosiela odlišné odpovede pre situácie, kedy je nesprávne práve používateľské meno alebo naopak heslo. [20]

Útočníci používajú aj tzv. *credential stuffing*, pri ktorom už majú zoznam ukradnutých prihlasovacích údajov. Automatizovaným skriptom skúšajú dané prihlasovacie údaje v lukratívnych službách, za predpokladu že používateľ nepoužíva odlišné kombinácie prihlasovacieho mena a hesla v službách, ktoré používa. Týmto spôsobom sa útočníci v krátkom čase dostanú do veľkého množstva účtov, v ktorých môžu ukradnúť citlivé informácie alebo vykonávať akcie predstierajúc, že sú daným používateľom. [21]

Existujú však postupy, ktoré tieto útoky zastavia alebo minimalizujú ich následky. Zabezpečená aplikácia by mala zabrániť vytvoreniu slabého hesla. Na detekciu slabého hesla existujú v súčastnosti rôzne knižnice a postupy, ktoré by vývojár pri implementácii registračných formulárov mohol použiť. Prihlasovacie údaje by mali byť dostatočne zašifrované, aby v prípade ukradnutia z databáze ich dešifrovanie nebolo triviálne. Formuláre pre prihlasovanie, registráciu a obnovenie zabudnutého hesla by nemali explicitne potvrdiť existenciu používateľského mena, prípadne vyzradiť ktorá časť prihlasovacích údajov je nesprávna. Implementácia viacstupňového overenia alebo časové obmedzenie neúspešných prihlásení môže takisto zabrániť alebo spomaliť automatizované útoky.

Kapitola 4

Návrh aplikácie

Po štúdii existujúcich aplikácií v kapitole 2 by nová aplikácia mala poskytovať riešenie navrhnuté na mieru pre zberateľov, ktoré má zrozumiteľné používateľské prostredie a neodradí používateľov počas používania. V tejto kapitole sú popísané jej funkcie a technické špecifikácie spolu s technológiami, ktoré sú vhodné na jej implementáciu.

4.1 Funkcie aplikácie

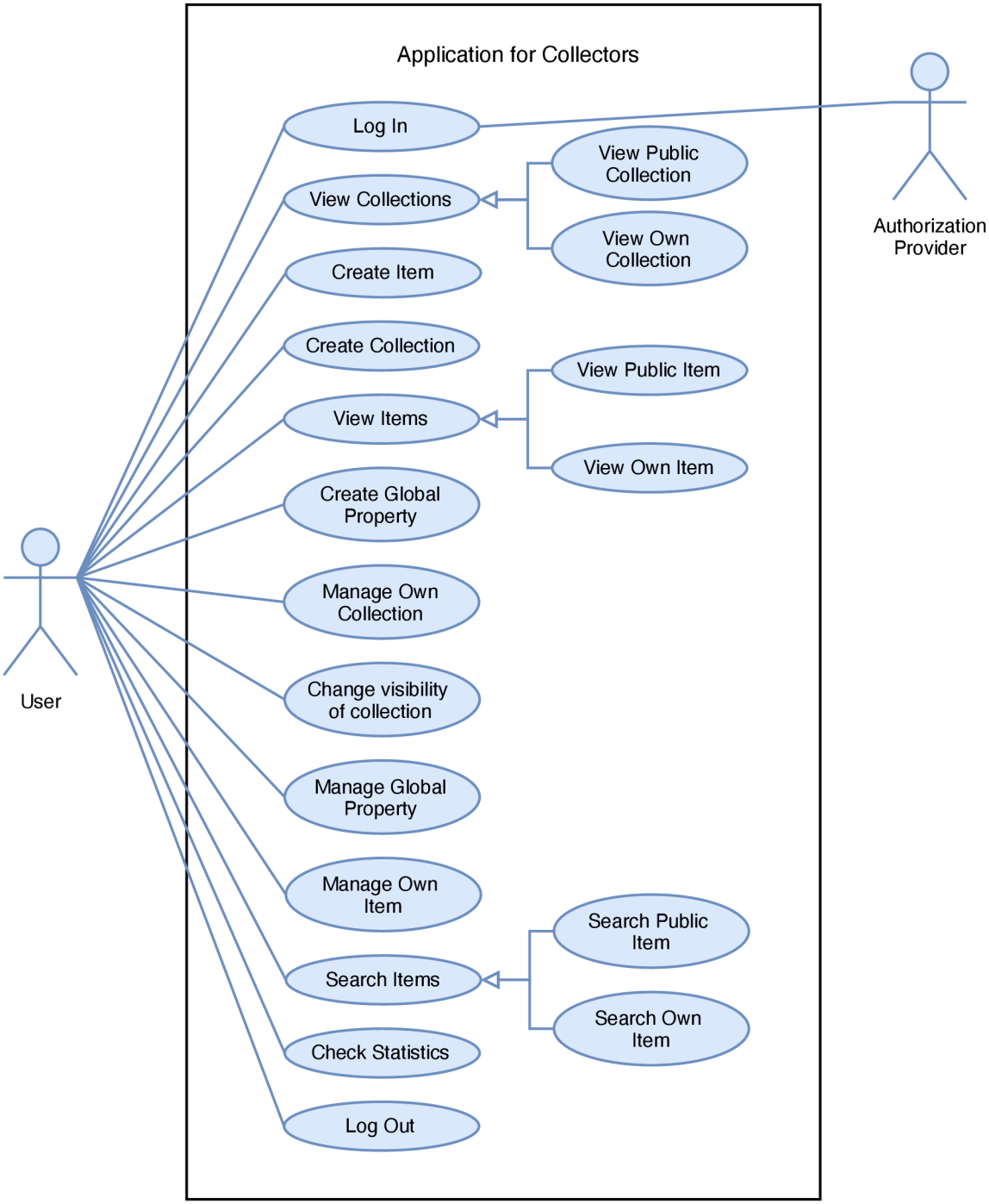
Aplikácia pre zberateľov by mala implementovať pár kľúčových funkcií. Prvou je zabezpečený prístup do aplikácie, a teda použitie overeného poskytovateľa autorizačných služieb. Aplikácia sa bude spoliehať výhradne na autorizáciu pomocou poskytovateľa a nebude implementovať vlastný autorizačný systém – nebude ukladať používateľove meno a heslo. Spomedzi všetkých poskytovateľov je vhodný napríklad *Google*, keďže ponúka viacstupňové overenie, je známy a rozšírený na trhu[17, 9]. Neautorizovaný používateľ nebude môcť vstúpiť do aplikácie.

Po prihlásení aplikácia používateľovi ponúka niekoľko možností. Používateľ môže zobraziť prehľad vlastných kolekcií. V kolekciách vidí prehľad položiek, pričom je možné daný prehľad filtrovať. Rovnako môže zobraziť prehľad všetkých verejných kolekcií, spolu s detailom ich položiek. Detail vlastných položiek a detail položiek nachádzajúcich sa vo verejných kolekciách sa mierne líši. Používateľ dokáže vytvoriť položky, kolekcie a následne ich spravovať, tzn. vie ich upraviť a odstrániť. Z dát, ktoré používateľ vloží do danej aplikácie, sa vytvárajú štatistiky, ktoré si daný používateľ môže pozrieť a zistiť, ako sa jeho zbierky vyvíjajú v čase. Z akejkoľvek sekcie aplikácie sa používateľ dokáže odhlásiť.

Tieto funkcie patria medzi základné a vo výslednej aplikácii pre zberateľov nesmú chýbať. Sú prehľadne znázornené v diagrame prípadov použitia (angl. *use case diagram*) na obrázku 4.1.

Existujú však ďalšie funkcie, ktoré majú nižšiu prioritu, no mohli by pozitívne ovplyvniť existujúcich používateľov a prilákať nových. Medzi ne patrí napríklad možnosť vytvorenia kolekcie použitím šablóny alebo komentovanie verejných položiek spolu s možnosťou pridať si jednotlivých používateľov medzi svoje kontakty, čím by sa mohla vytvoriť používateľská komunita. Takisto možnosť importu dát z externého súboru, čím by používatelia mohli jednoducho prejsť na danú aplikáciu z existujúcich riešení. Tieto funkcie ale nie sú súčasťou tzv. *minimum viable product* (skrátene MVP)[1], ale boli by súčasťou vývoja v ďalších iteráciách.

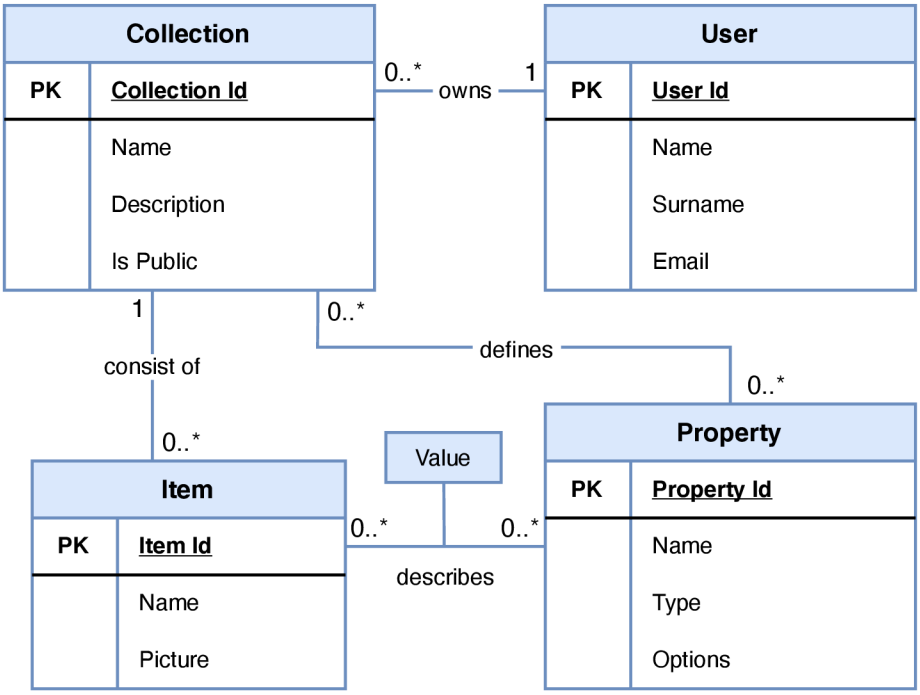
[1] produkt s najmenšou možnou funkcionalitou



Obr. 4.1: Diagram prípadov použitia aplikácie pre zberateľov

4.2 Dátový model

Aplikačný dátový model sa dá abstraktne znázorniť pomocou *entity-relationship diagram* (skrátene ERD). ERD sa používa pri návrhu informačných systémov, ktoré majú informácie uložené v databáze. V aplikácii pre zberateľov je možné identifikovať entity: `collection`, `item`, `property` a `user`. Tieto entity a ich vzťahy sú zakreslené na obrázku 4.2.



Obr. 4.2: ERD dátového modelu aplikácie pre zberateľov

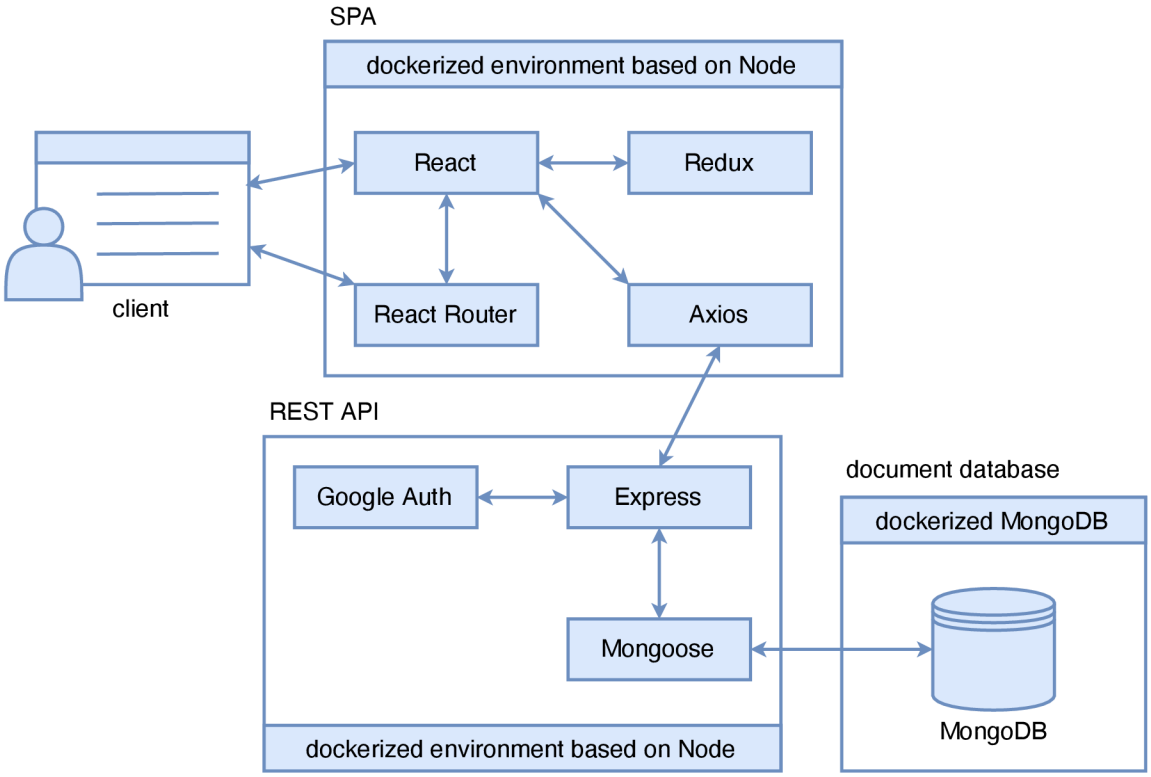
Kolekcia má vzťah ku všetkým ostatným entitám – používateľ ju vlastní, atribúty definujú jej položky a jednotlivé položky ju tvoria. Jednotlivé inštancie kolekcie musia mať názov, môžu mať popis a navyše obsahujú príznak o tom, či je daná kolekcia súkromná alebo verejná. Na základe príznaku sa kolekcie a položky kolekcie budú zobrazovať vo verejných zbierkach a budú mať inú množinu akcií. Položky kolekcie musia mať názov, môžu mať obrázok a obsahujú hodnoty jednotlivých atribútov, ktoré sú priradené k danej kolekcii. Položka môže byť súčasťou len jednej kolekcie, a naopak kolekcia môže (ale nemusí) mať viacero položiek. Atribút sa skladá zo svojho názvu a typu. Atribút môže mať dodatočné vlastnosti, ktoré ho odlíšia v rámci jedného druhu. Keďže atribúty nebudú vytvárané pre konkrétne kolekcie, ale budú sa môcť použiť globálne, atribút môže byť priradený k viacerým kolekciám a kolekcia môže mať viacero atribútov. Atribúty popisujú jednotlivé položky a hodnota daného atribútu je závislá na ich vzťahu. Atribút môže popisovať viacero položiek a položka môže mať viacero atribútov. O používateľovi je potrebné vedieť jeho meno, priezvisko a e-mailovú adresu, ktorá slúži ako kontakt. Kolekcia musí mať priradeného len jedného používateľa – vlastníka, no používateľ môže vlastniť niekoľko kolekcií.

Zaujímavou úlohou bolo vyriešiť modelovanie entity `property` a vzťahy tejto entity voči entitám `item` a `collection`. Ak má byť atribút globálny pre všetky kolekcie daného používateľa, nie je možné ho zahrnúť do entity `collection`. Preto je modelovaný ako samostatná entita. Hodnoty atribútov by bolo možné modelovať v rámci entity `property`, no získanie danej hodnoty by bolo podmienené identifikátormi oboch entít – `collection` a `item`. Preto sú hodnoty atribútov mapované priamo v entite `item`, keďže v tomto prípade je hodnota podmienená len jedným identifikátorom entity `property`. Pri zvolení dokumentovej databázy sa tieto hodnoty môžu implementovať vo forme hašovacej tabuľky[2], takže získanie daných hodnôt bude mať časovú zložitosť O(1) nezávisle od počtu položiek v tabuľke.

[2]štruktúra, ktorá asociuje kľúče s hodnotami

4.3 Architektúra a zvolené technológie

Klient aj server bude implementovaný v rovnakom programovacom jazyku – JavaScript. Takýto vzor sa nazýva *JS end-to-end*. Architektonické riešenie je možné vidieť na obrázku 4.3.



Obr. 4.3: Architektúra navrhnutej aplikácie pre zberateľov

Klientom bude SPA, ktorá bude asynchrónne komunikovať so serverom, čím sa docieli lepšia UX pre používateľa. Knižnica React v spolupráci s knižnicou React Router zabezpečia podmienečné vykresľovanie jednotlivých komponent spolu s navigáciou v aplikácii. React bude takisto komunikovať s knižnicou Redux, ktorá zabezpečí správu stavu aplikácie, do ktorého sa budú ukladať dáta načítané z databázy. Asynchrónna komunikácia medzi klientom a serverom bude implementovaná pomocou knižnice Axios[3].

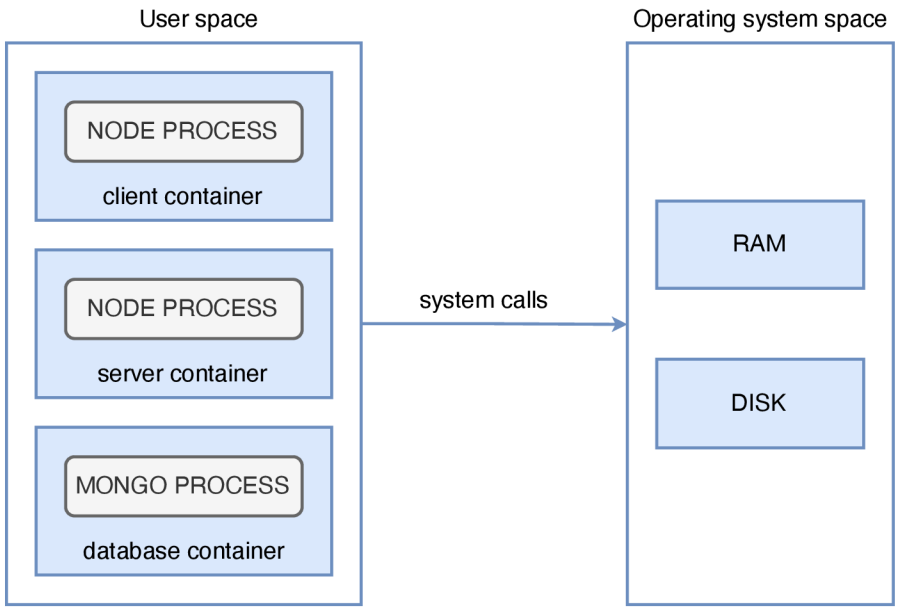
Serverová časť bude implementovať štandard REST rozhrania, na čo je vhodné minimalistické aplikačné rozhranie Express[4] založené na prostredí Node.js. Ako poskytovateľa autorizácie je využitý Google, ktorý pre takéto účely poskytuje knižnicu Google Auth[5]. Komunikácia s databázou bude prebiehať cez knižnicu Mongoose[6], ktorá poskytuje objektové modelovanie entít založené na schéme, pričom má zabudované overovanie dátových typov, funkcie na dopytovanie databázy a jednoduchú manipuláciu s dátami.

Dáta budú uložené v dokumentovej databáze MongoDB[7] vo formáte JSON.

[3] https://www.axios.com/
[4] http://expressjs.com/
[5] https://github.com/googleapis/google-auth-library-nodejs
[6] https://mongoosejs.com/
[7] https://www.mongodb.com/

Všetky komponenty aplikácie budú bežať v tzv. dockerizovanom prostredí (angl. *dockerized environment*). Dockerizované prostredie je prostredie, ktoré je vytvorené pomocou softvéru Docker. Docker[8] je open-source projekt, ktorý vytvára virtuálne prostredia – kontajnery nad operačným systémom. Takýmto spôsobom je možné vytvoriť prostredie nezávislé na operačnom systéme, na ktorom sa daná aplikácia vyvíja alebo beží, čím je jednoduchší presun aplikácie medzi operačnými systémami, či nasadenie na produkčný server. Jediné, na čom záleží, je možnosť spustenia softvéru Docker na danom systéme. Kontajnery sú vlastne skupiny procesov, ktoré simulujú izolované operačné systémy a využívajú RAM a úložisko systému, na ktorom bežia. Neobsahujú však celé operačné systémy, len závislosti a požiadavky, ktoré sú nevyhnutné pre procesy v danom kontajneri. Základom je tzv. *Dockerfile*, ktorý predstavuje konfiguráciu požadovaného prostredia. Prostredie je následne zostavené v obraze (angl. *image*), ktorý beží v kontajneri. Mnoho obrazov je verejne dostupných na oficiálnych stránkach Docker Hub[9] a je možné ich použiť priamo, tie viac špecifické je naopak možné nakonfigurovať individuálne cez Dockerfile.

Každá logická časť aplikácie by mala byť oddelená v samostatnom kontajneri. Aj keď server a klient podľa návrhu vyžadujú rovnaké Node runtime prostredie, nie je vhodné ich spájať do jedného kontajneru, lebo prípadná rozšíriteľnosť a správa len jednej časti v rámci kontajneru nie je možná. Preto je správne aplikáciu rozdeliť do troch kontajnerov, viz. obrázok 4.4. Dva kontajnery obsahujú obraz založený na Node runtime prostredí, v ktorých beží klient a server, a ďalší kontajner s Mongo obrazom obsahuje databázu.



Obr. 4.4: Docker kontajnery pre server, klienta a databázu aplikácie

[8] https://www.docker.com/
[9] https://hub.docker.com/

Kapitola 5

Implementácia

Je prospešné, ak implementácia aplikácie prebieha v iteráciách. Tým sa docieli, aby v čo najkratšom čase existoval produkt, ktorý je funkčný a používateľný. V ďalších fázach sa postupne pridávajú funkcie a vlastnosti, ktoré zvyšujú kvalitu produktu, no nie sú až také nevyhnutné.

V nasledujúcich podkapitolách je popísaná implementácia jednotlivých komponentov aplikácie, ktorá odpovedá architektonickému návrhu z obrázku 4.3.

5.1 Serverová časť

Server beží na Node runtime prostredí a využíva rozhranie Express. Základný skript vytvára inštanciu Express servera, ktorá má nastavené API rozhranie a port, na ktorom má server počúvať. V tomto skripte sa taktiež vytvára pripojenie do Mongo databázy. Na konci skriptu sa nad inštanciou zavolá metóda `listen`, ktorá po spustení skriptu načúva požiadavkám až kým sa skript manuálne nezastaví. Ďalej je server rozdelený do troch adresárov: `/routes`, `/models` a `/controllers`.

Adresár `/routes` obsahuje súbory, ktoré implementujú a testujú REST API. Koncové body (angl. *endpoints*) zohľadňujú relácie medzi jednotlivými zdrojmi vo formáte:

```
/<source>/<id>/<relation>/<relation_id>/...
```

Jednotlivé koncové body sú zoskupené do súborov podľa hlavného zdroja v URI koncového bodu, kde napríklad súbor `collection-routes.js` obsahuje funkcie implementujúce koncové body:

```
GET http://localhost:8081/api/collections
GET http://localhost:8081/api/collections/<collectionId>
GET http://localhost:8081/api/collections/<collectionId>/items
GET http://localhost:8081/api/collections/<collectionId>/items/<itemId>
```

Tieto koncové body vrátia všetky alebo konkrétne používateľove kolekcie, takisto ako všetky alebo jednotlivé položky konkrétnej kolekcie.

Pripojenie na databázu je vyriešené objektovo-dokumentovým modelovaním založeným na schémach cez knižnicu Mongoose. Táto knižnica poskytuje rozhranie pre správu databázových pripojení spolu so správou dát v databáze. Súbory v adresári `/controllers` slúžia na komunikáciu s databázou. Sú v nich implementované funkcie, ktoré pomocou knižnice

Mongoose vytvoria, upravia alebo zmažú dokumenty z databázy. Každá funkcia vykonáva jednu úlohu a väčšinou patrí k jednému koncovému bodu. Tento princíp uľahčuje testovanie a kvalitu kódu. Výnimkou môžu byť úlohy, ktoré vyžadujú úpravu závislostí, ktoré by po akcii zostali v neúplnom stave. Napríklad, ak by sa po vymazaní používateľovej kolekcie nevymazali taktiež položky danej kolekcie, používateľ by k nim stratil prístup. Preto sa v koncovom bode pre zmazanie kolekcie volajú dve funkcie, a to na zmazanie kolekcie a zmazanie všetkých položiek danej kolekcie. Každý prístup do databázy je podmienený kontrolou, či má používateľ práva vidieť alebo spravovať dané záznamy.

Schémy jednotlivých databázových entít sa nachádzajú v adresári `/models`. Schéma je Mongoose objekt, ktorý popisuje kolekciu a dokumenty, ktoré daná kolekcia obsahuje. Podľa navrhnutého ER diagramu (viz. podkapitola 4.2) je možné namodelovať všetky entity pomocou Mongoose schém. Tieto schémy sú dynamické, je možné ich upraviť počas behu aplikácie. Na rozdiel od relačných databáz, dokumentové úložiská nemajú prísnu štruktúru a dokumenty sa v rovnakej kolekcii môžu od seba líšiť (viz. podkapitolu 3.3).

Vzťahy sa v dokumentových databázach modelujú dvojakým spôsobom. Dokumenty povoľujú vnorené štruktúry, takže odkazovaný dokument je možné priamo vložiť do daného dokumentu. Tento vstavaný dátový model (angl. *Embedded Data Model*) je vhodné využiť pri vzťahoch typu *one-to-one* alebo *one-to-many*, kedy entita vzťahu *many* je zobrazovaná výhradne v kontexte *one* entity. Vstavané dátové modely poskytujú lepší výkon pri čítaní a zápise do databázy, keďže dané operácie je možné vykonať pomocou jedného atomistického príkazu. Dokument je taktiež možné odkazovať referenciou, čo sa podobá na odkazovanie pomocou cudzích kľúčov pri relačných databázových systémoch. Odkazovaný dokument sa nachádza v inej kolekcii ako samostatná štruktúra a odkazuje sa naň pomocou identifikátora. Tento model sa nazýva normalizovaný dátový model (angl. *Normalized Data Model*). Používa sa v prípadoch, kedy by vstavaný dátový model vytvoril duplicitné dáta, ale efektivita dopytovania týchto dát by neposkytla dostatočné výhody, ktoré by prevážili dôsledky duplikácie. Pri modelovaní entít z navrhnutého ERD sa využíva odkazovanie identifikátorom, pričom je použitá agregačná metóda `populate`, ktorú poskytuje knižnica Mongoose. Táto metóda pri čítaní dokumentov automaticky vymení odkazovaný identifikátor za odkazovaný dokument alebo dokumenty, pričom je možné špecifikovať len konkrétne atribúty, ktoré bude odkazovaný dokument obsahovať. V dokumentoch je takto uložený len identifikátor, takže databáza neobsahuje žiadne duplicitné dáta.

V schéme je možné vytvoriť pravidlá a kontroly, ktoré sa budú vykonávať pri vkladaní alebo získavaní dát z databázy. Je možné určiť, ktoré atribúty sú povinné a ktoré majú byť unikátne. Z ERD je možné určiť, že kolekcia `collection` bude mať ako povinný atribút `userId`, prípadne kolekcia `item` bude mať ako povinný atribút `collectionId`. Do schémy je takisto možné pridať virtuálne atribúty, ktoré sú tvorené pomocou iných atribútov. Tieto atribúty budú súčasťou každého dokumentu, ktorý sa načíta pomocou knižnice Mongoose, nie je potrebné ich ukladať alebo dopytovať zvlášť. Príkladom je spojenie dvoch atribútov dokumentu do určitého formátu ako je meno a priezvisko používateľa. Takto je možné priamo implementovať virtuálny atribút `fullName` a vrátiť `‘${this.name} ${this.surname}‘`. Každý vytvorený dokument implicitne obsahuje identifikátor, nie je potrebné ho udávať do schémy entity. Identifikátor je uložený v atribúte `_id`. Je možné ho upraviť do tvaru bez podčiarkovníka implementovaním virtuálneho atribútu `id`, do ktorého sa uloží hodnota atribútu `_id`.

Ak pri vytváraní dokumentu neposkytneme položke identifikátor, Mongo automaticky vygeneruje identifikátor vo forme tzv. *ObjectId*. Ten sa skladá z časovej značky vytvorenia

danej položky, náhodných čísel a počítadla, ktoré je inicializované na náhodnú hodnotu. Identifikátory vo forme celočíselnej rady, kedy sa pri vytvorení nového záznamu jeho identifikátor zvýši o jeden v porovnaní s posledným záznamom v tabuľke, umožňujú *enumeration attack* (viz. podkapitola 3.4). Pri identifikátoroch vo formáte ObjectId je databáza viac zabezpečená.

Každá požiadavka by mala byť ošetrená kontrolou práv alebo správnosti požiadavky. Používateľove práva sa kontrolujú v tzv. *middleware*, ktorý predstavuje medzivrstvu servera, cez ktorú prechádzajú všetky požiadavky na serveri. Keďže tieto práva je potrebné kontrolovať pri každej požiadavke, middleware je vhodnou implementáciou. Middleware kontroluje platnosť používateľovho tokenu pomocou knižnice Google Auth a na základe neho získa informácie a identifikátor daného používateľa. Všetky ďalšie požiadavky do databázy sa dejú na základe používateľovho identifikátora zisteného v middleware, takže nie je možné, aby používateľ získal dáta ktoré nevlastní. Správnosť požiadavky je potrebné kontrolovať kvôli integrite servera alebo napríklad implementácii návratových stavových kódov v odpovedi. Používateľ pri dopytovaní servera môže do požiadavky vložiť čokoľvek. Úlohou vývojára je predvídať nebezpečné vstupy a náležite ich ošetriť. Všetky funkcie servera by si mali poradiť s takýmito vstupmi a nemali by vyústiť v stav, kedy server neočakávane zlyhá (angl. *unexpected server crash*). Samotný zápis do databázy pomocou knižnice Mongoose môže vyhodiť chybové hlásenie, napríklad pri pokuse o vloženie nového dokumentu, ktorý neprejde validáciou alebo má rovnaký identifikátor ako už existujúci dokument v kolekcii. Chybové hlásenie podobného charakteru by mohlo vyústiť k zlyhaniu servera, preto sa tieto situácie musia adekvátne obslúžiť. Pri tejto obsluhe sú implementované stavové kódy:

- `400: Bad request` – chyba vo validácii pri pokuse o vloženie danej položky do databázy,
- `401: Unauthorized` – požiadavka nemá platný token,
- `404: Not found` – používateľ dopytuje neexistujúci koncový bod, požadovaný zdroj nebol nájdený alebo používateľ nemá práva na zobrazenie daného zdroja,
- `500: Internal server error` – neočakávaná chyba na strane servera.

5.2 Klientská časť

Frontend aplikácie je postavený na knižnici React. Všetky časti webu sú rozdelené na stránky (angl. *pages*) na vyššej úrovni a komponenty na nižšej úrovni. Stránka je typ komponentu, ktorý zoskupuje menšie komponenty pod jednu webovú adresu – cestu. Prvú stránku, ktorú nový používateľ uvidí, je prihlásenie.

Autorizácia je implementovaná pomocou knižnice React Google Login[1], čo je knižnica, ktorá poskytuje React komponenty a funkcie, ktoré komunikujú s autorizačným serverom Google. Pred prvým použitím danej knižnice je potrebné aplikáciu zaregistrovať na stránkach Google Cloud Platform[2]. V tejto registrácii sa definujú domény, ktoré budú autorizované počas dopytovania na Google server. Google týmto spôsobom poskytuje vrstvu ochrany pred útokmi, ktoré by prišli z iných domén – autorizácia takého požiadavku sa nepodarí. Každá aplikácia získa identifikátor, na základe ktorého sa vyvoláva autorizácia

[1] https://www.npmjs.com/package/react-google-login
[2] https://console.cloud.google.com/apis/credentials

klienta a používateľa. Tento identifikátor je vhodné nezverejňovať, preto je uložený v súbore `.env`, ktorý sa neukladá do verzovacieho systému, ani nikde nezverejňuje. Po kliknutí na tlačidlo „Sign in with Google“ sa používateľ autorizuje svojím Google účtom, povolí aplikácii na základe jej identifikátora zdieľanie svojich údajov a autorizačný server aplikácii vráti potvrdenie a prístupový token. Aplikácia si následne token uloží do cookies na pevný disk používateľa. Použitím cookies je možné pri opätovnej návšteve stránky preskočiť autorizačný proces. Prístupový token je súčasťou každej požiadavky na server, až pokým platnosť tokenu nevyprší – použitá dĺžka platnosti tokenu je 55 minút. Ak používateľ aktívne používa stránku viac ako 55 minút, token sa obnoví na pozadí s použitím *refresh* tokenu. Inak sa používateľ musí autorizovať znova.

Presmerovanie a načúvanie na konkrétne URL adresy poskytuje knižnica React Router. `<Route />` je komponenta, ktorej sa zadajú dva atribúty – cesta a komponent, ktorý sa má vykresliť po navštívení danej cesty. Na najvyššej úrovni sú definované dva `<Route />` komponenty. Prvý má cestu `/login`, na ktorú je používateľ presmerovaný zo všetkých ciest aplikácie ak nie je prihlásený, druhá cesta je `/`, ktorá predstavuje celú aplikáciu viditeľnú po prihlásení.

Vizuálna štruktúra aplikácie je rozdelená na viacero častí. Aplikácia má vrchnú lištu, navigačný panel vľavo a zvyšok obrazovky je telo stránky, ktoré sa dynamicky prekresľuje podľa zvolenej cesty.

Po vstupe do aplikácie sa používateľovi automaticky načítajú všetky dáta, ktoré má právo vidieť. Tieto dáta sú uložené v stave aplikácie, tzv. *store*, ktorého poskytovateľom je knižnica Redux.

Všetky stránky sú napojené na tento store pomocou knižnice React Redux a dokážu čítať a vyvolávať akcie, ktoré ho upravia. Dáta sú potom do komponent, z ktorých sú zložené stránky na najvyššej úrovni, propagované cez atribúty. Ak stránka vyvolá akciu, na základe ktorej sa zmení časť store, všetky komponenty, ktoré používajú tieto zmenené dáta sa prekreslia s novými dátami. Porovnanie starej a novej hodnoty jednoduchých dátových typov akými sú `string`, `number` alebo `boolean` je spoľahlivé. Dáta sú ale v store zoskupené do polí `items`, `collections`, `publicCollections` a `properties`. Pole je dátového typu `object`, a Redux pri tomto type porovnáva nový a starý stav len na základe jeho referencie[3]. Ak sa zmení čokoľvek vo vnútri poľa, jeho referencia sa tým nezmení, takže vo výsledku v aplikácii nenastanú žiadne vizuálne zmeny, lebo Redux tieto zmeny nezachytí. Táto situácia sa dá vyriešiť spôsobom, v ktorom namiesto úpravy položiek poľa vytvoríme nové pole, ktoré má novú referenciu, pričom sa všetky pôvodné hodnoty skopírujú a tie požadované upravia. Tento spôsob má ale vedľajší účinok – pri úprave jedného prvku poľa sa prekreslia všetky prvky daného poľa. To môže mať za následok problémy s výkonom aplikácie alebo neočakávané vizuálne zmeny spojené s prekresleniami závislými na poradí prvkov. Spoľahlivé je použitie knižnice Reselect, ktoré po implementácii vykonáva tzv. *hlboké* porovnanie[4], a teda vie zistiť zmenu len jedného prvku z poľa, na základe čoho sa vyvolá prekreslenie len daného prvku. Samotné React komponenty môžu mať takisto stav, ktorý sa nastavuje vo vnútri danej komponenty. Ak sa v projekte používa knižnica Redux, je spoľahlivejšie React stavy využiť len pre jednoduché dátové typy, ktorých zmeny je možno zistiť.

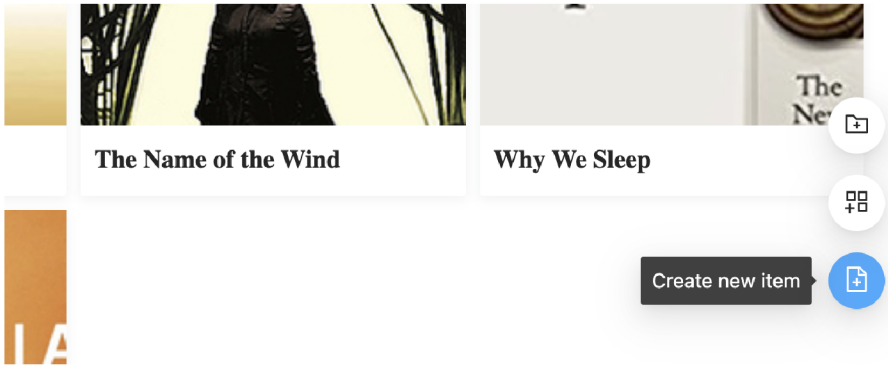
Po prihlásení sa používateľovi zobrazí domovská stránka. Na nej je možné vidieť štatistiky agregované z dát v store aplikácie – počet položiek v jednotlivých kolekciách, hodnota kolekcií a vývoj ich hodnoty v čase. V spojnicovom grafe vývoja hodnôt jednotlivých kolekcií si používateľ môže zvoliť časový rozsah spomedzi možností posledný týždeň, posledný

[3] miesto v pamäti operačného systému, v ktorom je hodnota daného objektu uložená

[4] rekurzívne porovnanie všetkých prvkov

mesiac, posledný kvartál a posledný rok. Výpočet hodnoty zbierok sa viaže na konkrétny atribút „Price“, ktorý je ako jediný verejný a môžu ho použiť všetci používatelia. Grafické zobrazenie štatistík je implementované pomocou knižnice Recharts[5], ktorá ponúka široké spektrum možností vykreslenia grafov a diagramov. Na stránke sa taktiež nachádza zoznam nedávno vytvorených položiek, pričom po kliknutí na jednotlivé položky je používateľ presmerovaný na ich detail.

Stránky, ktoré zobrazujú jednotlivé entity, obsahujú v pravom dolnom rohu *floating action buttons* (skrátene FAB). Je to tzv. plávajúce tlačidlo, ktoré sa dynamicky vykresľuje podľa kontextu aktuálnej stránky. Na stránke s prehľadom používateľových atribútov sa zobrazí FAB pre pridanie atribútu. Pri prehľade kolekcií sa zobrazí FAB pre pridanie kolekcie. V detaile kolekcie sa zobrazia FAB pre pridanie položky, pridanie atribútu a pridanie kolekcie (viz. obrázok 5.1).



Obr. 5.1: *Floating action buttons* na stránke *Collections*

Na stránke „Collections“ používateľ vidí prehľad svojich kolekcií. Z tejto stránky je možné dané kolekcie upraviť, odstrániť alebo pridať nové. V detaile kolekcie sa nachádza zoznam položiek kolekcie vo forme galérie. Každú položku v tomto zobrazení predstavuje obrázok s jej názvom, pričom ak obrázok nie je špecifikovaný, vykreslí sa na jeho mieste zástupný obrázok určitej veľkosti (angl. *placeholder*). Po kliknutí na položku sa zobrazí detail položky. Ľavej časti zobrazenia dominuje obrázok alebo zástupný obrázok, pravá časť je určená atribútom danej položky. Jednotlivé hodnoty atribútov sa vykresľujú rôzne na základe typu atribútu – napríklad hodnoty atribútov typu „Price“ majú za číslom skratku meny a hodnoty atribútov typu „Web address“ obsahujú odkaz na danú webovú stránku. Pri vytvorení položiek sa komponenty formulára takisto líšia na základe typu atribútu. „One-line text“ a názov predstavuje jednoduché vstupné pole, „Multiline text“ a description atribúty obsahujú viacriadkové vstupné pole a atribút typu „Date“ obsahuje komponentu pre výber dátumu. Atribúty názov a obrázok sú predvolené atribúty všetkých položiek. Používateľ dokáže vložiť obrázok iba vo formáte URL, čím sa znižujú požiadavky na kapacitu úložiska produkčného serveru.

Zoznam položiek v kolekcii obsahuje vyhľadávacie vstupné pole (angl. *search bar*). Na základe zadaného textu sa zoznam položiek, ktoré odpovedajú hľadanému výrazu, aktualizuje v reálnom čase. Položky sa filtrujú na klientovi. Hoci databáza Mongo implementuje pokročilé funkcie vyhľadávania, toto vyhľadávanie je možné optimalizovať jedine vytvorením indexov nad atribútmi, ktoré sa týmto vyhľadávaním budú dopytovať. Atribúty položiek sa

[5] https://recharts.org/en-US/

ale vytvárajú dynamicky, preto nie je možné efektívne vopred definovať indexy nam týmito atribútmi.

Stránka „Properties“ zobrazuje zoznam používateľových atribútov a poskytuje možnosť na ich úpravu alebo odstránenie. Pri kliknutí na tlačidlo odstránenia akejkoľvek entity v aplikácii je používateľ upozornený, že daná akcia sa nedá vrátiť späť a svoje rozhodnutie musí dodatočne potvrdiť v modálnom okne.

Na stránke „Public Collections“ je zoznam kolekcií, ktoré ich autori označili ako verejné. Tieto kolekcie nie je možné z daného pohľadu upraviť alebo odstrániť, na vykonanie týchto akcií musí ísť používateľ na stránku „Collections“. Verejné kolekcie obsahujú názov kolekcie a meno autora spolu s jeho profilovým obrázkom. Informácie o autorovi kolekcie sa zo servera vracajú priamo v entite verejnej kolekcie, kvôli čomu nie je nutné dodatočne dopytovať server.

Komunikácia so serverom je implementovaná v súboroch nachádzajúcich sa v adresári `/services`. Tie poskytujú funkcie, ktoré sa dopytujú servera pomocou knižnice Axios. V klientskej časti aplikácie je v komunikácii medzi klientom a serverom taktiež použitý middleware, ktorý zabezpečuje, aby každá požiadavka obsahovala vo svojej hlavičke autorizačný token. Ak zo servera príde odpoveď so stavovým kódom `401 - Unauthorized`, používateľovi sa automaticky vymažú cookies spojené s aplikáciou a následne je presmerovaný na prihlasovaciu obrazovku. Odhlásenie používateľa prebieha rovnako, tým pádom sa používateľ pri opätovnej návšteve stránky musí znova autorizovať.

Návrh dizajnu stránky prebiehal v iteráciách a úzko súvisel s možnosťami použitej UI knižnice Ant[6]. Táto knižnica ponúka množstvo React komponentov, určitú farebnú škálu a ikony, ktoré vytvárajú jednoducho použiteľný dizajnový systém. Aj vďaka tomu bolo možné vytvoriť minimalistické používateľské prostredie za relatívne krátky čas, ktoré odpovedá moderným aplikáciám.

[6] https://ant.design/

Kapitola 6

Testovanie

Verziu MVP aplikácie testovali používatelia rôznych vekových kategórií a povolaní. Aplikácia bola kvôli týmto účelom nahraná na produkčný server a zverejnená pod vlastnou doménou, aby k nej používatelia mali prístup cez internet. Používateľské testovanie potvrdilo fakt, že autorovi aplikácie, ktorý je do hĺbky zoznámený s aplikáciou, môžu jednoducho uniknúť inak očividné nedostatky. Každému používateľovi bol poskytnutý krátky videoklip v ktorom bolo predvedené, ako sa aplikácia používa, a následne obdržal testovací protokol. Použitý testovací protokol je súčasťou príloh (viz. Príloha B).

Pri vyhodnocovaní testovania boli používatelia rozdelení do niekoľkých skupín:

- používatelia nad 40 rokov,
- používatelia do 40 rokov nepracujúci v obore informačných technológií (skrátene IT),
- používatelia do 40 rokov pracujúci v obore IT.

Hranica 40 rokov predstavuje oddelenie generácie, ktorá vyrastala s modernými technológiami od tej, ktorá tieto technológie začala využívať v neskoršom období svojho života a nie je s nimi tak spätá. Používatelia, ktorí pracujú v IT priemysle mali od ostatných používateľov navyše poznámky a rady, ktoré úzko súviseli s ich nadobudnutými vedomosťami a skúsenosťami ohľadne vývoja a fungovania webových aplikácií.

6.1 Spätná väzba

Používatelia nad 40 rokov mali najväčší problém zorientovať sa na stránke a pochopiť, ktoré akcie majú aký účel aj napriek zhliadnutiu informačného videoklipu. Z tohto dôvodu bolo testovanie konané vo forme diskusie cez komunikačný kanál podporujúci zdieľanie obrazovky za prítomnosti autora, čím boli títo používatelia mierne ovplyvnení jeho znalosťami. Títo používatelia potrebovali vysvetliť navigáciu na stránke viac do hĺbky, nevedeli nájsť jednoduché akcie ako pridanie položky alebo úprava kolekcie. Taktiež koncept vytvárania atribútov a následne vypĺňanie týchto atribútov pri vytváraní položiek im bol cudzí a neboli schopní tieto akcie vykonať plynule, bez pomoci autora. V aplikácii je použitý minimalistický dizajn a väčšina operácií je znázornená ikonou bez textu, pričom po umiestnení kurzoru nad danú ikonu sa objaví popis danej akcie. Aplikácia sa chcela zamerať na čo najväčšie spektrum používateľov, preto sú všetky texty písané v anglickom jazyku. Používatelia uviedli, že navigácia by bola zrozumiteľnejšia, ak by aplikácia obsahovala viac textovej

interpretácie jednotlivých akcií a ponúkala možnosť prepnutia jazyka na český jazyk alebo slovenčinu. Väčšina si vôbec nevšimla FBA na jednotlivých stránkach a navrhla, aby tieto tlačidlá boli buď väčšie, alebo umiestnené v inej, viac viditeľnej časti obrazovky.

Používatelia do 40 rokov boli schopní testovať aplikáciu samostatne. Odpovede všetkých sa zhodujú v pár bodoch, ktoré by boli v aplikácii upravené podľa návrhov v ďalšej iterácii vývoja, keďže daná funkcionalita kriticky ovplyvňuje celkové vnímanie aplikácie. Najväčší problém používatelia videli v priraďovaní a vytváraní atribútov vytvorenej kolekcie. Súčasná implementácia neumožňuje vytváranie atribútov počas vytvárania kolekcie. Povinné atribúty, akými sú „Name“ a „Picture“, nie sú uvedené nikde v aplikácii okrem formulára na vytvorenie položky. Atribút „Price“, ktorý je spoločný pre všetkých používateľov, je možné priradiť ku kolekcii, no nie je zrejmé že iba na základe neho sa počítajú štatistiky. Nenachádza sa ani v zobrazení všetkých používateľových atribútov na stránke „Properties“. Tieto fakty vyústili v situáciu, kedy si každý z používateľov vytvoril vlastné atribúty „Name“, „Picture“ a „Price“, pričom pri vytváraní položky zistil, že atribúty s rovnakým názvom už existujú, takže vlastné atribúty následne odstránil. Používatelia nepracujúci v IT oblasti mali navyše problém s výberom typov dodatočných atribútov „Year“ a „Place“. Pre atribút „Year“ sa snažili použiť typ atribútu „Date“, ktorý ale umožňuje výber jedine celého dátumu. Komponenta výberu dátumu navyše nepodporuje rýchlu navigáciu medzi rokmi, a tak používatelia strávili zbytočne veľa času na to, aby v nej zvolili dátum 1.1.1937. Títo používatelia taktiež očakávali, že sa pri vytvorení atribútu „Place“ bude vo výbere typov nachádzať typ atribútu „Place“. Používatelia sa následne zhodli na tom, že by pri typoch atribútov boli vhodné príklady použitia daných atribútov a aké hodnoty dané atribúty môžu nadobudnúť. Používatelia pracujúci v IT oblasti na tento problém nenarazili a vedeli si vhodne dátové typy hodnôt jednotlivých atribútov premietnuť do dostupných možností.

Pri vytváraní a upravovaní položky používatelia taktiež narazili na pár spoločných nedostatkov. Pri vytvorení vlastného atribútu typu „Price“ nie je možné špecifikovať menu. Momentálne je mena tohto typu atribútu nastavená na korunu českú (skrátene Kč), no daná skratka meny sa vo formulári na vyplnenie hodnoty atribútu nenachádza. Tým pádom nie je jasné, že výsledná hodnota bude obsahovať danú menu a nie je ju potrebné uvádzať explicitne. Pri testovaní používatelia takisto upozornili na fakt, že nie je jasné v akom formáte má byť vložená hodnota obrázku (URL). Viacerí používatelia by ocenili, ak by vkladanie obrázku podporovalo nahratie súboru. Používatelia pracujúci v IT oblasti si navyše všimli, že formuláre pre úpravu a pridanie entít nefungujú jednotne. Formulár modálneho okna na úpravu položky si pamätá posledný stav, no po aktualizovaní údajov sa formulár nepredvyplní. Očakávané chovanie by bolo opačné – pri zatváraní modálneho okna zmeny neuložiť, a po upravení dát a znovu otvorení formulár vyplniť s aktualizovanými hodnotami. S týmto riešením by bolo vhodné používateľa upozorniť, že vyplnené údaje po zatvorení modálneho okna zmiznú. Aplikácia nie je prispôsobená na rôzne veľkosti obrazoviek.

Medzi špecifické poznámky, ktoré sa už neopakovali, patrilo napríklad pridanie tlačidla na vytvorenie prvej položky do prázdneho stavu kolekcie, možnosť nastavenia vlastného poradia atribútov v rámci kolekcie alebo pridanie tmavého režimu. V poznámkach bol uvedený taktiež nápad na funkciu, kedy by verejné kolekcie boli prístupné používateľom aj bez prihlásenia, teda by bolo možné jednoducho zdieľať svoje zbierky len pomocou webovej adresy.

Aj napriek faktu, že aplikácia bola výhradne vyvíjaná v prehliadači Chrome[1], testovanie prebiehalo na viacerých internetových prehliadačoch a používatelia nezaznamenali žiadne odchýlky od očakávaného správania, ktoré by mohli byť zapríčinené nepodporovanou verziou použitého prehliadača.

Celkovo boli používatelia s aplikáciou spokojní, jednoduchý dizajn im vyhovoval. Niektorí používatelia ocenili použitie zabezpečeného protokolu HTTPS, ktorý je nevyhnutný na implementáciu Google autorizácie.

6.2 Vyhodnotenie

Motiváciou testovania bolo zistiť ako vníma navrhnutú aplikáciu rôznorodé spektrum používateľov. Testovací protokol bol navrhnutý spôsobom, aby používateľov nenavigoval pomocou atomických krokov, ale aby používatelia museli použiť dávku fantázie a vlastnými krokmi sa dostali k výsledku. Len takto mohlo testovanie aplikácie odhaliť množstvo nedostatkov, ktorých druh skutočne závisel na veku a povolaní používateľa.

Mladší používatelia, ktorí používajú rozmanité webové aplikácie dennodenne a sú viac flexibilní ohľadom vnímania používateľského prostredia, z dodaného protokolu rýchlo porozumeli konceptu a navigáciu na stránke považovali za prijateľnú v porovnaní s bežne dostupnými aplikáciami. Starší používatelia, ktorí prichádzajú do kontaktu s menším počtom aplikácií a sú navyknutí na určité používateľské rozhranie, mali problém už s porozumením konceptu a minimalizmus použitý v dizajne im naopak prekážal. Aplikácia chce byť zameraná na používateľov všetkých vekových kategórií, preto by mala reflektovať zmeny na základe spätnej väzby používateľov zapojených do prvého testovania. Používateľovi by pri prvej návšteve malo byť jasné aké tlačidlo má stlačiť alebo do akej sekcie vstúpiť, aby vložil dáta svojej kolekcie. Postup vytvárania kolekcie, priraďovanie atribútov a následné vytvorenie položiek v kolekcii by nemal vyvolávať otázky, ale naopak, mal by byť intuitívny a rýchly.

Dizajnové zmeny sa preto budú týkať hlavne hlbšieho popisu akcií a príklady použitia, ktoré by zrozumiteľne vyjadrovali ich význam. Tieto zmeny zahrnú pridanie výrazného tlačidla na vytvorenie prvého záznamu do prázdnych stavov entít, ktoré pomôže používateľovi rýchlo sa zorientovať a nehľadať túto možnosť po celej obrazovke. Tlačidlá na úpravu a vymazanie budú obsahovať názov, keďže význam použitých ikon nie je dostatočne intuitívny. Proces vytvorenia kolekcie bude urýchlený pridaním možnosti vytvoriť atribút priamo v danom formulári. Kroky vytvorenia atribútu sa zredukujú do jedného, pričom do formulára pribudnú príklady použitia daných typov a informácia o tom, čo vlastne atribúty v aplikácii predstavujú. Všetky texty budú preložené do slovenčiny a stránka bude obsahovať prepínač medzi anglickým a slovenským jazykom.

Zmeny funkcionality budú obsahovať správu a použitie atribútov, aby používateľom bolo jasné, že už po prihlásení v aplikácii existujú atribúty, ktoré môžu použiť a nedajú sa upraviť. Do zoznamu atribútov budú pridané atribúty s názvom „Name“, „Picture“ a „Price“, pričom tieto atribúty budú vizuálne odlíšené od atribútov vytvorených používateľom. Novým atribútom typu `price` bude používateľ môcť špecifikovať menu z množiny predvolených možností. Štatistiky na domovskej stránke budú zahŕňať dáta zo všetkých atribútov typu `price`, pričom bude možné zvoliť z atribútov ktorej meny majú výpočty vychádzať.

[1] https://www.google.com/chrome/

Kapitola 7

Záver

Výsledkom bakalárskej práce je MVP verzia aplikácie pre zberateľov. Na vytvorenie návrhu som skúmala existujúce riešenia dostupné na rôznych platformách, a urobila som komplexný rozbor, v ktorom som zdôvodnila, prečo dané riešenia nevyhovujú mojej predstave o aplikácii určenej zberateľom. Po rozhodnutí, že výsledná aplikácia bude webová, som preskúmala dostupné informácie ohľadne vývoja a architektúry webových aplikácií, popísala som ich historický vývoj a základné prvky a koncepty technológií, ktoré som chcela použiť. Jednu podkapitolu som venovala bezpečnostným problémom, na ktoré je potrebné si dávať pri vývoji webových aplikácií pozor. Na základe týchto teoretických poznatkov som si bola schopná predstaviť, ako daný vývoj prebieha, a prejsť ku konkrétnemu návrhu.

V návrhu aplikácie bolo dôležité začať jednoducho – stanoviť si ciele, ktoré sú reálne a splniteľné v rámci bakalárskej práce a vyústia v produkt, ktorý je funkčný a vizuálne odpovedá moderným aplikáciám. Popis základných funkcií aplikácie som vyjadrila v diagrame prípadov použitia. Na základe definovaných akcií som vedela identifikovať jednotlivé entity, ktorých vzťahy a atribúty som popísala v E-R diagrame. Na implementáciu všetkých častí aplikácie som zvolila architektúru JS end-to-end, takže klient aj server je implementovaný pomocou jazyka JavaScript. V klientovi využívam React a dizajnový systém Ant na vykreslenie dynamickej SPA, Redux na správu stavu aplikácie a React Router na navigáciu medzi sekciami aplikácie. Server je založený na Node prostredí a je implementovaný pomocou knižnice Express. Ako úložisko dát som zvolila dokumentovú databázu MongoDB, takže dynamické vytváranie atribútov kolekcií nevyústi v redundanciu prázdnych hodnôt v databázových štruktúrach, ako by to bolo v prípade relačných databázových systémov. Na komunikáciu servera s databázou som sa rozhodla použiť objektovo-dokumentové modelovanie, čo prináša dodatočnú kontrolu schémy dokumentov, ktoré implicitne žiadnu definovanú schému nemajú. Autorizácia používateľov je implementovaná pomocou služby Google, pričom je to jediný spôsob, ako sa prihlásiť do aplikácie. Jednotlivé aplikačné časti bežia vo vnútri oddelených Docker kontajnerov, čím sa rozšíriteľnosť a nasadenie na akýkoľvek produkčný server stáva jednoduchším. Túto skutočnosť som využila a úspešne som aplikáciu zverejnila na doméne dostupnej cez internet pre účely testovania.

Cieľom bakalárskej práce bolo vytvoriť zrozumiteľnú aplikáciu pre zberateľov, k čomu výrazne pomohlo používateľské testovanie po implementácii MVP verzie aplikácie. Testovanie prebiehalo pomocou testovacieho protokolu, pričom kroky protokolu som schválne nedefinovala algoritmicky, aby používatelia k ich splneniu museli využiť fantáziu a test mohol skutočne preveriť zrozumiteľnosť implementovaného dizajnu. Chyby návrhu a implementácie, ktoré testovanie odhalilo, som spísala v spätnej väzbe a následne som vyhodnotila zmeny, na základe ktorých by sa aplikácia v ďalšej iterácii stala zrozumiteľnejšou a vše-

obecne lepšou. Po vykonaní týchto nevyhnutných zmien by som opätovne nechala túto aplikáciu otestovať, no už nie na základe predpísaného protokolu, ale voľne používateľmi, ktorí sú zberateľmi pre nich významných predmetov.

Vypracovaním tejto bakalárskej práce som mala možnosť spoznať moderné technológie do hĺbky, implementovať aplikáciu od základov a následne ju ako svoju prvú kompletnú aplikáciu zverejniť na internete. Naučila som sa pracovať systematicky a v iteráciách, čo ma doviedlo k úspešnej implementácii webovej aplikácie pre zberateľov.

Literatúra

[1] Anicas, M.: An Introduction to OAuth 2. online, 2014.
URL https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

[2] Berners-Lee, T.: Information Management: A Proposal. online, 1990.
URL https://www.w3.org/History/1989/proposal.html

[3] Berners-Lee, T.: What's new in '92. online, 1992.
URL https://www.w3.org/News/9201.html

[4] Coronel, C.; Morris, S.; Rob, P.: *Data Systems Design Implementation and Management*. Cengage Learning, 9 vydanie, 2009, ISBN 978-0538748841.

[5] Dalling, T.: Model View Controller Explained. online, 2009.
URL https://www.tomdalling.com/blog/software-design/model-view-controller-explained/

[6] Davidse, J.: API fundamentals. online, 2020.
URL https://developer.ibm.com/articles/api-fundamentals/

[7] Facebook: Facebook Reports Fourth Quarter and Full Year 2020 Results. online, 2021.
URL https://investor.fb.com/investor-news/press-release-details/2021/Facebook-Reports-Fourth-Quarter-and-Full-Year-2020-Results/default.aspx

[8] Fielding, R.; Irvine, U.; Gettys, J.; aj.: Hypertext Transfer Protocol – HTTP/1.1. online, 1999.
URL https://tools.ietf.org/html/rfc2616

[9] Georgiev, D.: Google Services Statistics and Facts. online, 2021.
URL https://review42.com/resources/google-statistics-and-facts/

[10] Gibb, R.: What is a Web Application? online, 2016.
URL https://blog.stackpath.com/web-application/

[11] Grolinger, K.; Higashino, W. A.; Tiwari, A.: Data management in cloud environments:. *Journal of Cloud Computing*, ročník 22, 2013, ISSN 2192-113X-2-22, https://doi.org/10.1186/2192-113X-2-22.

[12] Group, M. M.: INTERNET GROWTH STATISTICS. online, 2020.
URL https://www.internetworldstats.com/emarketing.htm

[13] Gómez, H.; Serna, M.; Badanes, R.: EVOLUTION AND TRENDS OF INFORMATION SYSTEMS FOR BUSINESS MANAGEMENT. *Dyna*, ročník 77, 2009, ISSN 2346-2183.

[14] Jessup, L. M.; Valacich, J. S.; Wade, M.: *Information Systems Today*. Toronto: Pearson College, Štvrté vydanie, 2008, ISBN 978-0136078401.

[15] Khanzode, K. C. A.; Sarode, D. R. D.: EVOLUTION OF THE WORLD WIDE WEB: FROM WEB 1.0 TO 6.0. *International Journal of Digital Library Services*, ročník 6, č. 2, 2016, ISSN 2250-1142.

[16] Laudon, K. C.; Laudon, J. P.: *Management Information Systems*. New York: Prentice Hall, 12 vydanie, 2011, ISBN 978-0132142854.

[17] Lemonnier, J.; Latto, N.: Is It Safe to Sign in with Facebook or Google? online, 2020. URL https://www.avg.com/en/signal/is-it-safe-to-log-in-with-facebook-or-google

[18] MDN-contributors: HTML: HyperText Markup Language. online, 2021. URL https://developer.mozilla.org/en-US/docs/Web/HTML

[19] MDN-contributors: HTML5. online, 2021. URL https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[20] MyraSecurity: What is credential cracking? online. URL https://www.myrasecurity.com/en/what-is-credential-cracking/

[21] MyraSecurity: What is credential stuffing? online. URL https://www.myrasecurity.com/en/what-is-credential-stuffing/

[22] OWASP: OWASP Top Ten 2017. online. URL https://owasp.org/www-project-top-ten/2017/

[23] Pankov, N.: Enumeration attack dangers. online, 2020. URL https://www.kaspersky.com.au/blog/username-enumeration-attack/27193/

[24] Ramez, E.: *Fundamentals of database systems*. Boston: Pearson ; Addison-Wesley, 6 vydanie, 2010, ISBN 0-136-08620-9.

[25] Scott, E.: *SPA Design and Architecture*. Manning Publications, prvé vydanie, 2015, ISBN 978-1617292439.

[26] V. Roesler, E. B.; Willrich, R.: *Special Topics in Multimedia, IoT and Web Technologies*. Springer International Publishing, prvé vydanie, 2020, ISBN 9783030351021.

Príloha A

Obsah pamäťového média

- `app/` – zdrojové súbory implementovanej aplikácie s návodom na spustenie
- `doc/` – zdrojové súbory technickej správy
- `demo.mp4` – demonstračné video z používania aplikácie
- `xbalko01_color.pdf` – technická správa
- `xbalko01_print.pdf` – technická správa bez farebných odkazov

Príloha B

Testovací protokol

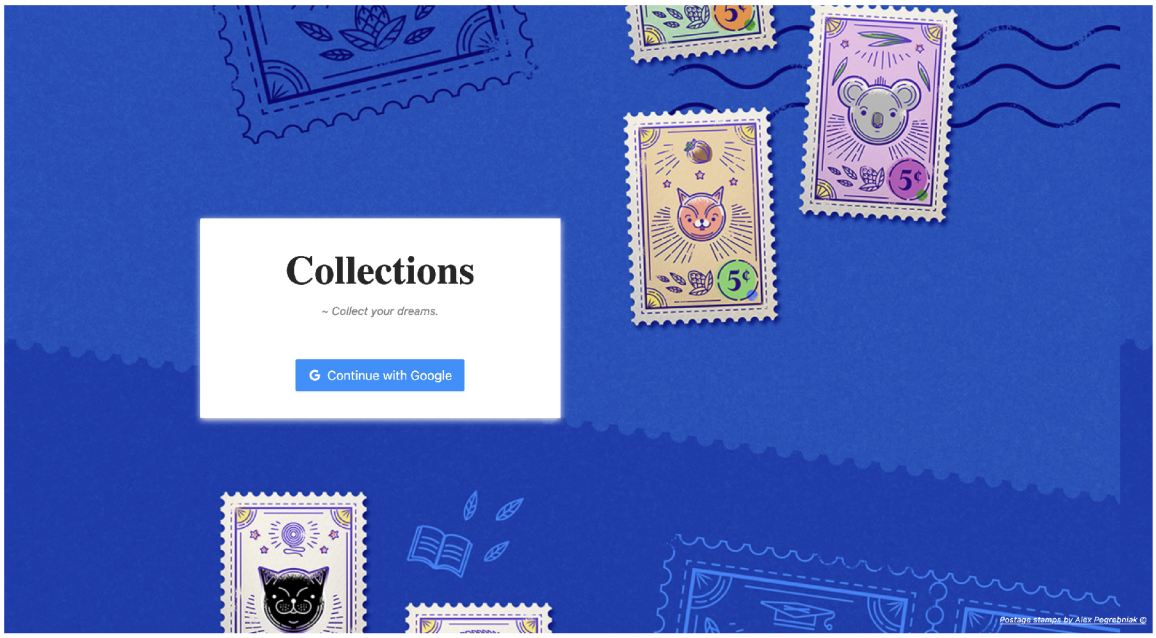
Táto testovacia sada by mala autorovi aplikácie poskytnúť informácie o funkčnosti, zrozumiteľnosti a kvalite dizajnu užívateľského prostredia aplikácie. Výsledkom je množina problémov, na ktoré používatelia počas testovania narazili, na základe ktorých následne autor aplikácie navrhne možnosti na zlepšenie.

Meno a priezvisko:
Použitý prehliadač:

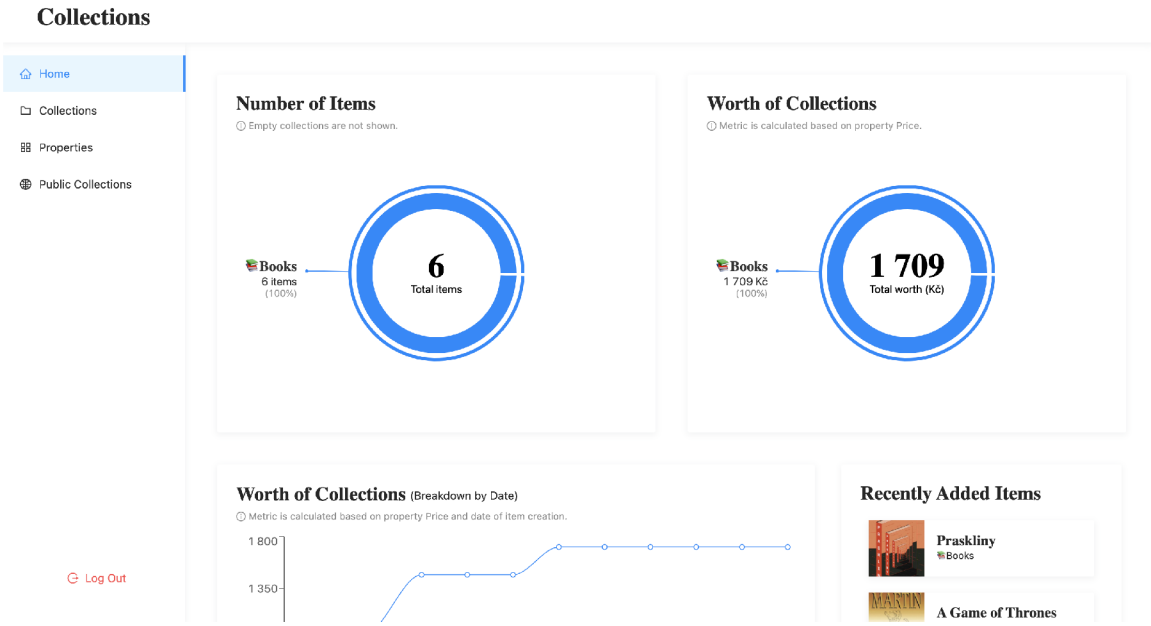
1. Navštívte stránku `https://collections.mikita.eu` a prihláste sa.
2. Vytvorte kolekciu s názvom **Postcards** a zaistite, aby jej položky mali atribúty **Name**, **Picture**, **Place**, **Price** a **Year**.
3. Pridajte do kolekcie **Postcards** novú položku:
   - Name: **Brno**
   - Picture: `https://collections.mikita.eu/postcard.jpeg`
   - Place: **Zelný trh**
   - Price: **25 Kč**
   - Year: **1937**
4. Upravte položke s názvom **Brno** cenu na **250 Kč**.
5. Pridajte kolekcii **Postcards** popis:
   - **Historical postcards of cities from Czech Republic.**
6. Zmeňte stav kolekcie **Postcards** zo súkromnej na verejnú.
7. Zobrazte si všetky verejné kolekcie.
8. Zobrazte si štatistiky.
9. Odhláste sa.

Príloha C

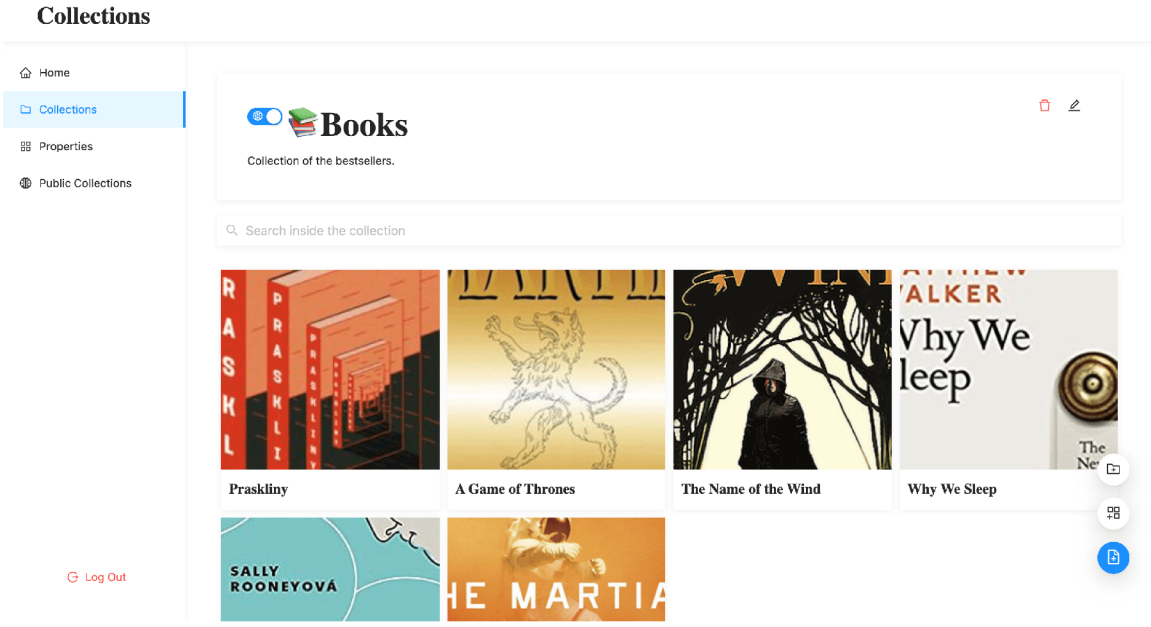
Snímky webovej aplikácie pre zberateľov



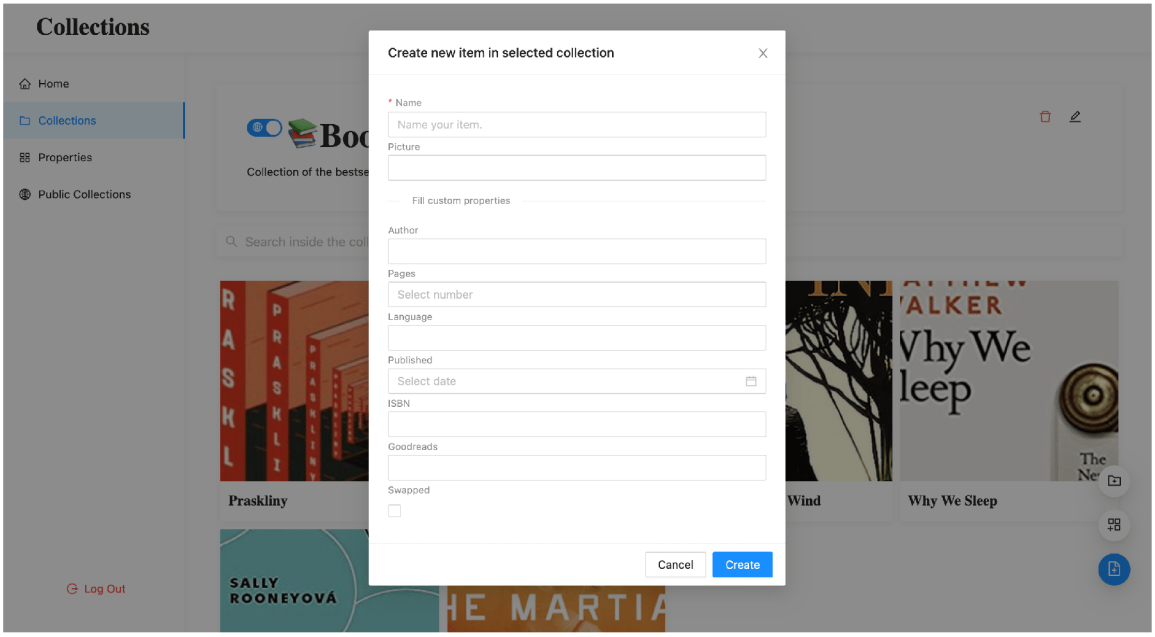
Obr. C.1: Prihlasovacia obrazovka



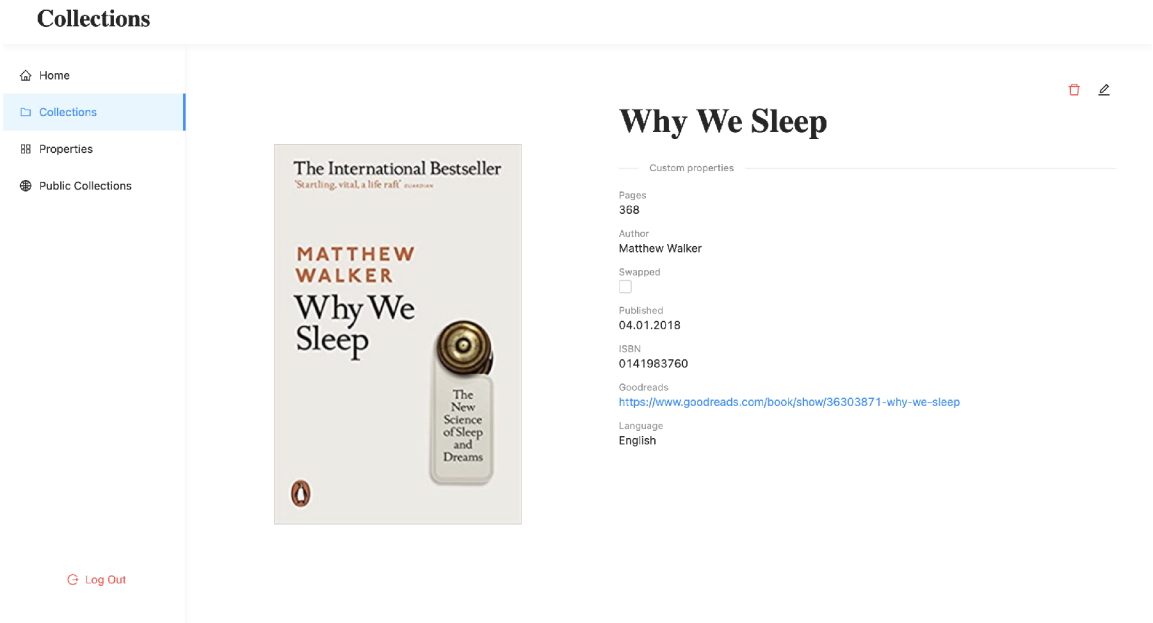
Obr. C.2: Domovská obrazovka



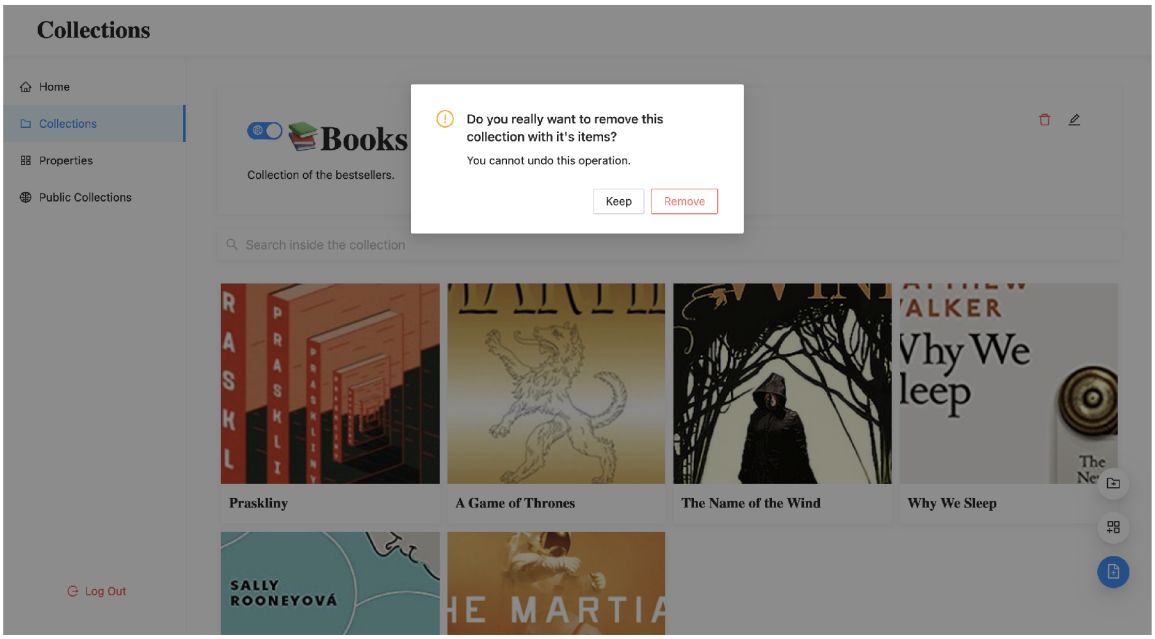
Obr. C.3: Položky kolekcie



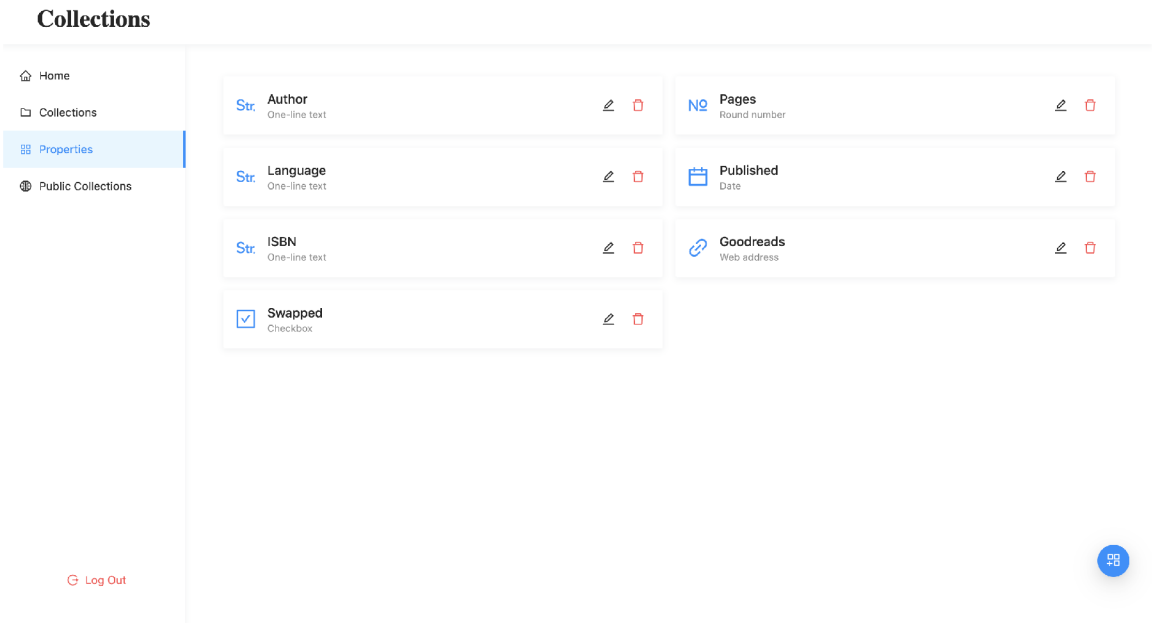
Obr. C.4: Formulár na vytvorenie položky



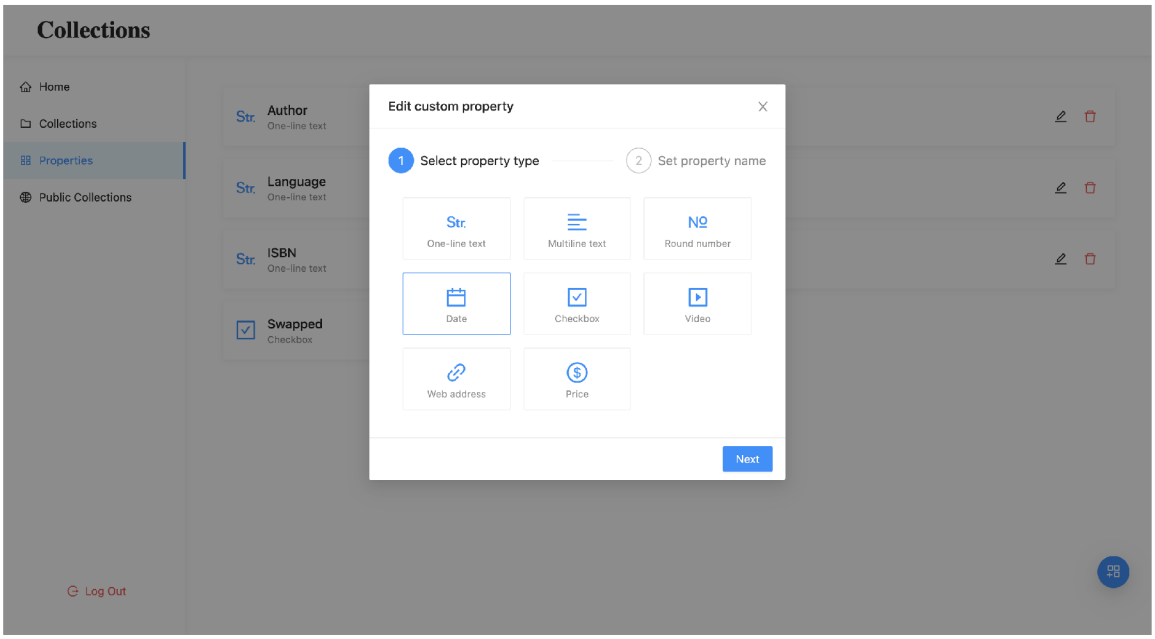
Obr. C.5: Detail položky



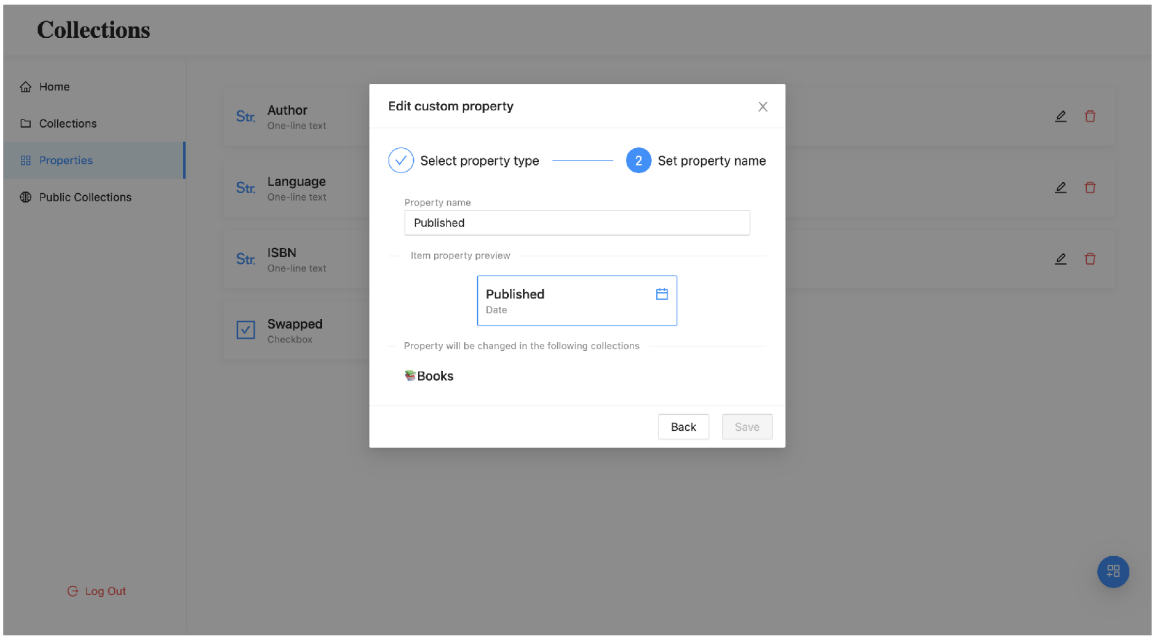
Obr. C.6: Potvrdenie vymazania kolekcie a jej položiek



Obr. C.7: Zoznam atribútov



Obr. C.8: Upravenie atribútu – krok 1



Obr. C.9: Upravenie atribútu – krok 2