VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PLATFORMA PRO SPRÁVU POZVÁNEK UŽIVATELŮ SOFTWARE

REFERRAL PROGRAM PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MAROŠ KOPEC**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20858/2017/xkopec44

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Kopec Maroš**
Obor: Informační technologie
Téma: **Platforma pro správu pozvánek uživatelů software**
Referral Program Platform
Kategorie: Web

Pokyny:

1. Seznamte se s principy tvorby webových aplikací a potřebnými technologiemi.
2. Navrhněte aplikaci, pomocí které uživatel bude posílat pozvánky potenciálním uživatelům softwaru, bude udržovat seznam čekajících, kteří se zajímají o software a zobrazovat statistiky o pozvánkách.
3. Po konzultaci s vedoucím navrženou aplikaci implementujte a proveďte testování.
4. Zhodnoťte dosažené výsledky a další možné pokračování tohoto projektu.

Literatura:

- Swicegood. T.: Programming Node.js. O´REILLY, 2012. ISBN 1934356891.
- Green, B.: AngularJS. O´REILLY, 2013. ISBN 1449344852.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Táto práca popisuje návrh a implementáciu platformy pre správu používateľských pozvánok softvéru realizovanú ako webovú aplikáciu. Aplikáciu umožňuje zakladať programy, v rámci ktorých je možné zasielať pozvánky, prijímať používateľov zo zoznamu čakateľov alebo spravovať stav pozvánok používateľov. V úvode sú špecifikované požiadavky na platformu. Následne je vytvorený návrh a implementácia aplikácie.

Abstract

This thesis describes the design and implementation of a referral program platform implemented as a web application. The application allows you to set up programs in which you can send invitations, accept users from the waiting list, or manage the status of users' invitations. The initial requirements for the platform are specified in the beginning of this thesis. Consequently, the design and the implementation of the application is discussed.

Kľúčové slová

správa pozvánok, webová aplikácia, platforma, web, REST API, javascript, fullstack, Vue.js, koa.js, Node.js, NoSQL, CouchDB

Keywords

referral, invitation, web application, platform, web, REST API, javascript, fullstack, Vue.js, koa.js, Node.js, NoSQL, CouchDB

Citácia

KOPEC, Maroš. *Platforma pro správu pozvánek uživatelů software*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Platforma pro správu pozvánek uživatelů software

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vladimíra Bartíka Ph.D.. Odborné informácie mi poskytol konzultant Ing. Martin Večeřa z firmy RedHat. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....
Maroš Kopec
16. mája 2018

Poďakovanie

Rád by som sa poďakoval za nie len odborné vedenie práce Martinovi Večeřovi. Poskytol mi dôležitú spätnú väzbu, ktorá je nevyhnutná pre kvalitnú prácu. No predovšetkým patrí moja vďaka mojej dlhoročnej partnerke Dominike. Len vďaka jej trpezlivosti a podpore som nakoniec dokázal dopísať túto prácu po roku a pol odkladania. Za to ti ešte raz ďakujem.

Obsah

1 Úvod **3**

2 Špecifikácia požiadaviek **4**
2.1 Definícia základných pojmov 4
2.2 Technické požiadavky na aplikáciu 4
2.3 Užívateľské požiadavky na aplikáciu 4
2.3.1 Model prípadov užitia 5
2.4 Existujúce riešenia 6

3 Návrh aplikácie **9**
3.1 Technologické požiadavky 9
3.2 Návrh databázy 10
3.2.1 Relačný databázový systém 10
3.2.2 NoSQL alternatíva 10
3.2.3 Návrh štrukúry dokumentov v databáze CouchDB 12
3.3 Návrh back-end časti 13
3.3.1 Využitie JavaSciptu za pomoci Node.js 13
3.3.2 Middleware framework Koa 14
3.3.3 Návrh štruktúry API 14
3.4 Návrh front-end časti 14
3.4.1 Reaktívny prístup 15
3.5 Zabezpečenie 16
3.6 Model návrhu platformy 18

4 Implementácia návrhu aplikácie **19**
4.1 Implementácia databázového systému 19
4.1.1 Úprava návrhu dokumentov databázy 19
4.1.2 Zobrazenia 21
4.2 Back-end 21
4.2.1 Koncové body rozhrania API 22
4.2.2 Implementovaná funkcionalita API 23
4.3 Front-end 23
4.3.1 Implementovaná funkcionalita 25
4.4 Testovanie 26
4.5 Návrh budúceho rozvoja 26

5 Záver **29**

Literatúra **30**

Prílohy **33**

A Návrh GUI **34**

B Screenshoty aplikácie **36**

C Obsah priloženého pamäťového média **42**

Kapitola 1

Úvod

Zasielanie pozvánok a prístup k webovej aplikácii len pre vyhradenú skupinu ľudí je bežná súčasť životného cyklu veľkej časti aplikácií. Najčastejšie využitie s ktorým sa používateľ môže stretnúť je uzavreté beta testovanie aplikácie. V tomto prípade má prístup k aplikácii len obmedzená skupina používateľov. Ďalšie časté využitie pozvánok vo svete softvérových aplikácií je kontrola rastu používateľskej základne. Momentálna situácia neposkytuje žiadne centralizované open source riešenie pre správu a zasielanie pozvánok.

Cieľom tejto práce je oboznámiť sa s princípmi tvorby webových aplikácií a potrebnými technológiami. Následne získané vedomosti z teoretickej časti aplikovať pri návrhu a implementácií webovej aplikácie. Vytvorenie prototypu open source webovej aplikácie, ktorú môže ďalej rozširovať komunita a analýza jeho funkcionality spolu s vytvorením návrhu budúceho rozvoja.

Text práce popisuje definíciu a špecifikáciu ako používateľských tak aj technologických požiadaviek na platformu. V tejto oblasti bol prevedený prieskum existujúcich riešení, pri ktorom nebola nájdená open source alternatíva spĺňujúca požadované požiadavky. Práca sa následne zameriava na návrh aplikácie a výber technológií vhodných pre tvorbu návrhu. Návrh aplikácie je rozdelený do štyroch hlavných častí. V prvej časti sa práca zameriava na rozhodovanie výberu databázového systému medzi štandardným relačným databázovým systémom a alternatívnym NoSQL systémom a následým návrhom databázového systému. Nasleduje časť venujúca sa výberu technológií a návrhu back-end časti aplikácie a REST API. Predposledná časť opisuje výber technológií a návrh front-end časti za pomoci ceruzkovej metódy. V poslednej časti návrhu aplikácie sa práca venuje zabezpečeniu aplikácie.

Implementácia aplikácie je taktiež rozdelená na štyri hlavné časti kopírujúce tri hlavné časti z návrhu. Ako prvé sa práca venuje implementácii databázového systému a úprave návrhu dokumentov. Práca pokračuje v implementácii back-end časti, kde bolo implementovane REST API rozhranie. Záver implementácie back-end časti obsahuje sumarizáciu implementovanej funkcionality. V neposlednom rade sa venuje pozornosť implementácii front-end časti, problémom s tým spojeným a sumarizácií implementovanej funkcionality. Kapitolu implementácie uzatvára časť venujúca sa testovaniu a návrhu budúceho rozvoja.

Kapitola 2

Špecifikácia požiadaviek

Táto kapitola podrobne pojednáva o tom, čo výsledný systém rieši. Na začiatok je vhodné definovať základné pojmy, s ktorými sa pracuje ako v celej práci, tak aj vo vytvorenej aplikácii.

2.1 Definícia základných pojmov

- Aplikácia – softvér, do ktorého môže užívateľ nadobudnúť prístupové práva po pozvaní
- Program – kampaň umožňujúca používateľom byť pozvanými do aplikácie
- Používateľ – potencionálny užívateľ aplikácie po pozvaní
- Systém – samotná webová platforma pre správu pozvánok používateľov
- Administrátor – používateľ systému a správca programu

2.2 Technické požiadavky na aplikáciu

Systém má byť schopný kontrolovane regulovať stav používateľov v aplikácií. Preto je nutné, aby bol čo najlepšie škálovateľný, ale zároveň prenosný.

Požiadavka na horizontálnu škálovateľnosť[1] je najdôležitejšou. Stojí na nej celý systém a je nutné, aby vedel zvládnuť exponenciálny nárast používateľskej základne každej zaregistrovanej aplikácie.

Prenosnosť systému rozumiem v zmysle, databázy a rozhrania. Databázu je možné migrovať zo servera na server alebo môže bežať na konfigurácii niekoľkých spolupracujúcich serverov. Užívateľské rozhranie bude priamo reagovať na stav používateľov, a teda v závislosti na požiadavkách môžu byť rozšírené servery, obsluhujúce užívateľské rozhranie a aplikačné programové rozhranie, nezávisle na sebe.

2.3 Užívateľské požiadavky na aplikáciu

Cieľovou skupinou používateľov sú ľudia spravujúci vlastnú webovú aplikáciu. Aplikácie môžu byť v zásade rôzneho druhu a preto nie je možné špecifikovať technickú zdatnosť potencionálnych používateľov. Je dôležité dbať na jednoduchosť a priamočiarosť funkcií.

[1] Rozšírenie pridaním viacerých serverov. Naproti tomu vertikálne škálovanie je upgrade serveru, na ktorom služba beží

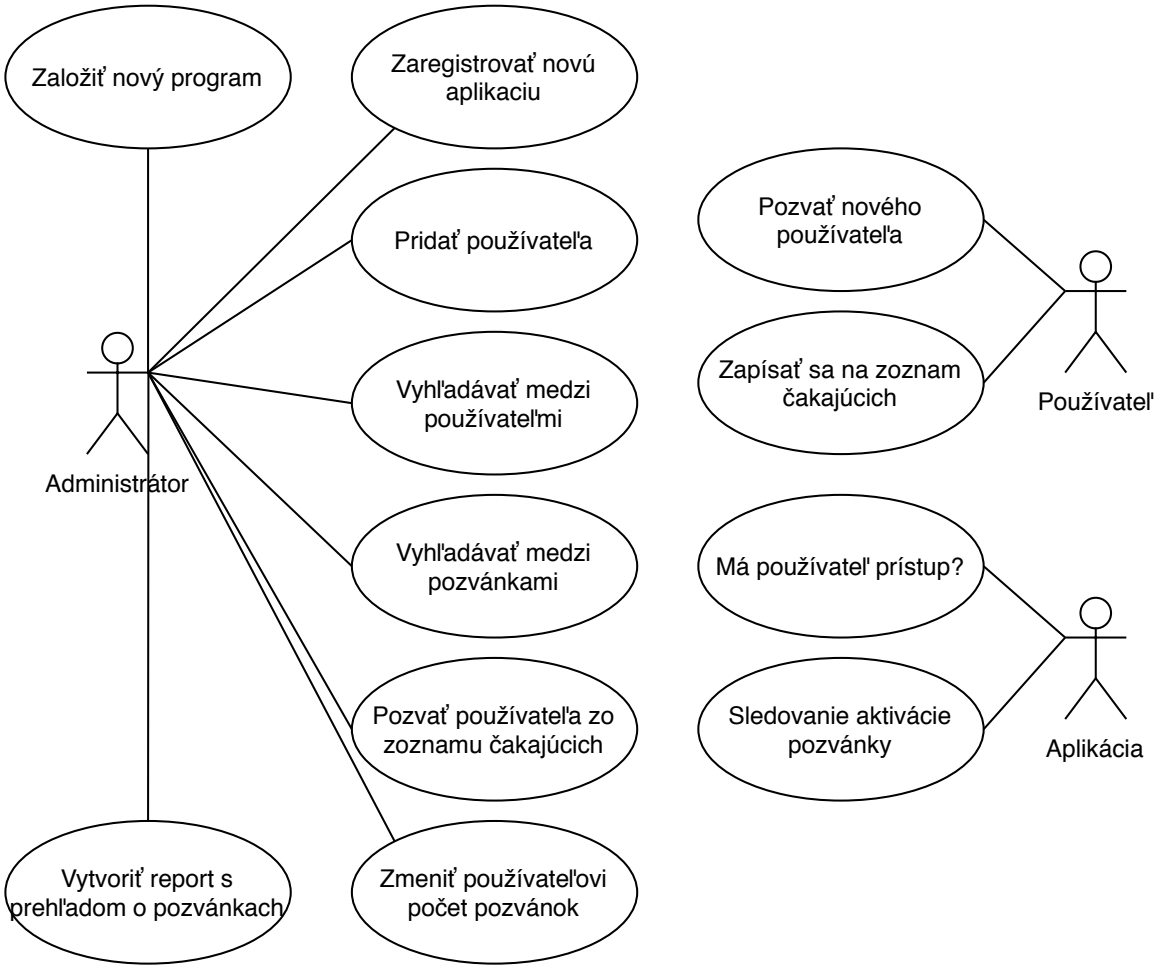
Vzhľadom na cieľovú skupinu by systém mal byť schopný posielať pozvánky, akceptovať pozvánky, v zmysle sprístupniť používateľovi prístup do aplikácie, zobrazovať administrátorovi štatistické údaje vo vhodnej forme a v neposlednom rade umožniť administrátorovi modifikáciu kvantity používateľových pozvánok.

Z vyššie uvedených faktov som špecifikoval tieto základné prípady užitia.

- Aplikácia sa môže spýtať systému, či je špecifikovaný používateľ už pozvaný do aplikácie
- Administrátor môže založiť nový program
- Administrátor môže zaregistrovať novú aplikáciu do už existujúceho programu
- Administrátor môže pridať akéhokoľvek používateľa do programu zadaním jeho emailovej adresy, čím sa ihneď vytvorí pozvánka
- Akýkoľvek používateľ s pozvánkou môže začať ihneď používať aplikáciu po kliknutí na pozývací odkaz
- Každý používateľ ma určený počet pozvánok, ktoré môže rozoslať, to znamená, koľko ďalších používateľov môže pozvať
- Žiaden používateľ nemôže byť pozvaný viac ako raz do jednej aplikácie, pri pokuse o opakované pozvanie je používateľ upozornený
- Pozvanie už pozvaného používateľa nezníži počet pozvánok
- Prvá aktivácia pozývacieho odkazu je sledovaná vrátané dátumu, času a geo-lokácie
- Administrátor môže vyhľadávať medzi používateľmi podľa dátumu ich aktivácie, dátumu zápisu na zoznam čakajúcich, počtu poslaných pozvánok, celkového počtu pozvánok, počtu pozvánok, ktoré ešte môžu odoslať
- Používateľ sa môže zapísať na zoznam čakajúcich
- Administrátor môže pozvať akýkoľvek počet používateľov zo zoznamu čakajúcich a zároveň špecifikovať počet povolených pozvánok, ktoré budú mať k dispozícii
- Administrátor môže zmeniť počet pozvánok, ktoré má používateľ alebo zvolená skupina používateľov ešte k dispozícii
- Administrátor môže vyhľadávať medzi pozvánkami a radiť, filtrovať ich podľa dátumu zaslania, podľa faktu, či boli prijaté, kde bolí prijaté a kedy boli prijaté
- Administrátor môže vytvoriť report s grafmi ukazujúcimi prehľad pozvánok podľa dátumu ich aktivácie, kedy boli zaslané, či boli aktivované, kedy boli aktivované a kde boli aktivované
- Používatelia sú rozoznávaní podľa ich emailových adries

2.3.1 Model prípadov užitia

Na základe užívateľských požiadaviek som vytvoril model prípadov užitia, ktorý je zobrazený na obrázku 2.1.



Obr. 2.1: Model prípadov užitia

2.4 Existujúce riešenia

V nasledujúcej časti sa zameriam na už existujúce riešenia, ktoré umožňujú nejakým spôsobom zasielanie pozvánok do softvéru a ich správu.

Existuje niekoľko riešení pre správu odporúčaní, no vo väčšine prípadov sa venujú problematike rozširovania aktívnych používateľov lákaním na odmeny. Napríklad platforma ReferralCandy[16] umožňuje jej zákazníkovi výber odmeny, ktorú získa po úspešnom odporúčaní nového zákazníka. Ponúka widgety pre webové stránky zákazníka, emailové šablóny, individuálne odkazy pre používateľov, ktoré môžu zdieľať, či špeciálne ponuky pre nových zákazníkov. ReferralCandy nedovoľuje self-referral, teda odporúčanie samého seba. Jedná sa o podvod za účelom obohatenia sa. Mesačný poplatok za túto službu začína na 49 dolároch.

Rovnaké služby ako ReferralCandy ponúka aj Referral SaaSquatch[15]. Platforma ponúka riešenia pre rôzne typy aplikácií, menovite On demand services (služby na požiadanie), Subscription services (predplatné služby), Subscription boxes (predplatné baliky) a Online marketplaces (onlnie obchody). Taktiež využíva ochranu pred nelegálnym zneužívaním od-

mien za pomoci automatizovaného odhaľovania podvodov. Cena za využívanie platformy Referral SaaSquatch začína na 499 dolárov mesačne.

Ďalšou podobnou platformou, ktorá láka používateľov na odmeny je Rocket Refferals[17]. Mimo spomínané služby ponúkané vyššie spomínanými platformami, Rocket Referrals umožňuje zasielať rukou písané ďakovné listy. Mesačný poplatok začína na 80 dolároch.

Všetky tri spomínané platformy ponúkajú prehľad štatistických údajov v prehľadnom formáte. Uvedené platformy však riešia úplne iný prípad užitia, a to „odporuč nového používateľa a získaj bonus“. Tento bonus závisí na aplikáciách využívajúcich služby týchto programov. Aplikácie si takýmto spôsobom začnú rozširovať svoju základňu používateľov, ak sú ochotné investovať do počiatočného rastu kapitál alebo naopak, už nedokážu zväčšiť počet aktívnych používateľov. Žiadna zo spomínaných nie je Open-source a dokonca nie je dostupná bez poplatkov.

V predošlom odseku som sa venoval platformám, ktoré sa zameriavali na rast používateľskej základne pomocou systému odmien. V tomto odseku sa viac zameriam na platformy ponúkajúce zasielanie pozvánok ako doplnkovú službu.

Najznámejším zástupcom je Firebase. Jedná sa o Platform as a Service[2] od spoločnosti Google LLC. Ide o kolekciu nástrojov, vďaka ktorým môže ich používateľ vytvoriť a spravovať webovú aplikáciu, mobilnú aplikáciu alebo aj aplikáciu pre klasický stolový počítač veľmi rýchlo a jednoducho. Platfmorma je dostupná aj zadarmo s obmedzeniami vo forme maximálneho stropu pre požiadavky na server a pod. Firebase ponúka širokú škálu produktov od hostingu, monitorovania výkonu či testovania, až po analýzy, dynamické odkazy, pozvánky a najnovšie dokáže aj predvídať správanie používateľov.[6]

Veľkou nevýhodou je, že platforma Firebase je proprietárnym softvérom a núti jej predplatiteľov do tzv. vendor lock-in[2], čo je stav, kedy je zákazník závislý na produkte alebo produktoch od jediného dodávateľa a nie je preňho možné prejsť k inému dodávateľovi bez navýšenia výdajov alebo vzniknutia nepríjemností. Tento fakt znemožňuje vytvoriť aplikáciu nezávislú na platforme. Za ďalší problém považujem, že paltforma nie je Open-source, teda nie je možné ju ďalej rozširovať, vylepšovať a zabezpečovať ju jej samotnými používateľmi.

Open source alternatív k platforme Firebase je hneď niekoľko. Medzi ne patrí aj Parse Server[14]. Open source platforma ponúkajúca Backend as a Service[3] bola pôvodne proprietárny softvér Parse.com patriaci pod spoločnosť Facebook. Parse Server ponúka jednoduchú tvorbu webového API, čo môže uľahčiť správu pozvánok softvéru. Neobsahuje však hotové funkcie pre zasielanie pozvánok.

Ďalším zástupcom spomedzi open source alternatív je Hoodie[8], ktorá taktiež predstavuje Backend as a Service službu špecifickú technológiou noBackend. Ide o technológiu, kedy je všetka serverová časť aplikácie extrahovaná mimo aplikáciu. Tento fakt považujem za problém, pretože používaním platformy Hoodie stráca jej používateľ kontrolu nad serverovou časťou. Usudzujem, že to určite nevyhovuje používateľom, ktorý majú implementovaný vlastný backend a hľadajú riešenie pre správu zasielania pozvánok.

[2]Kategória cloud computing services, ktorá ponúka platformu, ktorá dovoľuje zákazníkom vytvárať, spúšťať, a manažovať aplikácie bez komplexnosti budovania a správy infraštrukúry.

[3]Model, ponúkajúci vývojárom webových a mobilných aplikácií spôsob, ako prepojiť serverové úložisko na cloude a API. Zároveň ponúka funkcie ako správa používateľov, notifikácie a integrácia so sociálnymi sieťami

Vyššie uvedonou analýzou existujúcich riešení som dokázal, že momentálne neexistuje žiadne riešenie, ktoré by sa priamo venovalo správe pozvánok používateľov softvéru a zároveň bolo aj open source.

Požiadavka žiadajúca od platformy open source licenciu je pre mňa najdôležitejšia, nakoľko pod takouto licenciou je pre jej používateľov možné si platformu upravovať podľa svojich špecifických požiadaviek, na ktoré nie je možné brať ohľad pri proprietárnych riešeniach.

Okrem platformy Firebase, žiadna z nájdených platforiem neponúkala prostý manažment pozvánok, prostredníctvom ktorého je možné striktne a kontrolovane regulovať používateľský stav aplikácie.

Kapitola 3

Návrh aplikácie

V nasledujúcej kapitole sa budem venovať návrhu webovej aplikácie. Postupoval som metódou zdola nahor, čo znamená, že ako prvú som navrhol databázu a rozhranie, následne som vytvoril návrh užívateľského prostredia. Ako vôdzku pri tvorbe návrhu platformy som si zvolil model MVC - Model-View-Controller (Model-pohľad-kontrolér). Jedná sa o koncept zapuzdrenia niektorých dát spolu s ich spracúvaním a izolácie tejto časti od jej manipulácie a prezentácie. Model predstavuje objekt dát, zatiaľ čo view reprezentuje vizualizáciu modelu. Controller predstavuje prostriedky, pomocou ktorých je možné meniť stav modelu [3]. To mi umožnilo venovať sa, v rámci možností, každej z častí individuálne.

3.1 Technologické požiadavky

Vo svete softvérového inžinierstva je možné na základe návrhového princípu oddelenia zodpovednosti aplikáciu rozdeliť na dve základné časti, prezentačnú vrstvu a prístupovú vrstvu. Jedná sa o časti softvéru podľa modelu klient-server, kde front-end reprezentuje klientsku časť, a teda prezentačnú vrstvu aplikácie a back-end reprezentuje naopak serverovú časť, inak povedané prístupovú vrstvu. Pre jednoduchosť je možné generalizovať pojmy front-end a back-end v tomto poradí, ako užívateľské rozhranie a server [22].

Pre vytvorenie užívateľského rozhrania sa vo všeobecnosti používajú značkovacie jazyky HTML (Hypertext Markup Language) a CSS (Cascading Style Sheets). Vďaka spomínaním jazykom je jednoduché vytvoriť grafické užívateľské prostredie. To je potrebné ešte oživiť a sprístupniť tak používateľovi funkcionalitu aplikácie. V tejto časti prichádza na rad programovací jazyk JavaScript. JS, ako je často skracovaný, je vysoko-úrovňový, dynamický, slabo-typovaný a objektovo orientovaný interpretovaný jazyk.

Ako som poukázal v predošlom odseku, je pre mňa nevyhnutné používať JavaScript pre tvorbu front-end časti aplikácie. Preto som sa rozhodol, že využijem JS aj pri programovaní back-end časti. To mi umožní plne sa sústrediť na obsah a funkcionalitu, zatiaľ čo sa nebudem musieť zamýšľať nad sémantikou a syntaxou viacerých programovacích jazykov. Takýto prístup k tvorbe webovej aplikácie sa nazýva „Full-stack Javascript“ a v dnešnej dobe sa jedná o jeden z najpopulárnejších spôsobov vytvárania webových aplikácii [9]. Zvolenie si programovacieho jazyka vopred je pre mňa výhodou z hľadiska výberu technológií už pri návrhu platformy. Dokážem tým do istej miery prispôsobiť navrhované princípy, a tým sa vyhnúť prípadným problémom pri hľadaní vhodnej implementácie.

3.2 Návrh databázy

Platforma je priamo závislá na návrhu a implementácii databázy. Je preto veľmi dôležité rozmyslieť si štruktúru a model databázy dopredu. Vďaka tomu sa dokážem vyhnúť mnohým problémom pri implementácii aplikačného programového rozhrania, ako napríklad problému chýbajúcej entity. Návrh však nemôžem považovať za konečný a úplný, pretože nie je možné myslieť na všetky faktory dopredu. Preto je len logické načrtnúť čo najpresnejší hrubý návrh, ktorý sa bude upresňovať podľa budúcich potrieb.

V nasledujúcich podkapitolách sa venujem výberu databázy s ohľadom na horizontálnu škálovateľnosť, viď kapitola 2.2.

3.2.1 Relačný databázový systém

Relačný databázový systém je systém riadenia databázy založený na relačnom modeli [34]. Tento model je prístup ako riadiť dáta v databáze za pomoci štruktúr a jazyka založeného na predikátovej logike prvého rádu, kde sú všetky dáta reprezentované ako množiny spájané do relácií [28]. Dáta sú teda reprezentované vo forme tabuliek, kde jeden záznam predstavuje jeden riadok.

Vyššie spomínané databázy sa vyznačujú vlastnosťami značenými akronymom ACID – Atomicity, Consistency, Isolation, Durability. Atomicity alebo tiež atomickosť zaručuje, že prebehne každá alebo žiadna z akcií v transakcii. Pri zlyhaní jednej z akcií tvoriacej transakciu sa transakcia nevykoná vôbec a žiadne zmeny nebudú vykonané. Consistency, slovensky konzistentnosť nikdy nespôsobí nekonzistentné dáta. Isolation, v preklade izolovanosť, nazývame vlastnosť, keď transakcie nevedia o konkurentných transakciách. Durability, teda trvácnosť, záruka, že výsledok transakcií pretrvá všetky udalosti, aj nehody [27].

Jdnou z prvých volieb sú jednoznačne všetky databázy využívajúce jazyk SQL. SQL - Structured Query Language, teda štruktúrovaný dotazovací jazyk, je programovací jazyk využívaný relačne riadenými databázovými systémami.

Medzi jeho výhody patrí prístup k viacerým záznamom v rovnakom čase a eliminácia potreby špecifikácie cesty k záznamu. SQL je ale štruktúrovaný deklaratívny programovací jazyk, ktorému chýbajú funkcie procedurálneho programovacieho jazyka [33]. Tento nedostatok riešia rôzne rozšírenia, najznámejší zástupcovia sú IBM DB2, MySQL, MariaDB, Oracle, PostgreSQL.

3.2.2 NoSQL alternatíva

Od svojho vzniku vytvárali databázy využívajúce technológie NoSQL konkurenciu široko rozšíreným už tradičným SQL databázam. NoSQL, čo je akronym pre „Not Only SQL“ (Nie len SQL), nemá za úlohu nahradiť SQL, práve naopak. Spája nerelačné databázy so zaužívanými relačnými databázami. Tieto technológie sa snažia využiť to najlepšie z oboch strán, a tým pokryť rastúce požiadavky na výkon, škálovateľnosť a flexibilitu schém s ohľadom na integritu dát a ich konzistenciu [31].

Na rozdiel od relačne riadených databáz sa databázy NoSQL sa nevyznačujú vlastnosťami ACID, ale BASE - Optimistic behavior, Basically available, Soft state, Eventually consistent. Optimistic behavior, slovensky optimistické správanie, dovoľuje dočasné nezrovnalosti v databáze. Basically Available, alebo tiež prakticky prístupné, hovorí o prístupnosti za pomoci replikácie. Soft state, v preklade mäkký stav, určuje úkon udržiavania databázy konzistentnou jej používateľovi. Eventually consistent, teda nakoniec konzistentné, opisuje konzistentnosť ako slabú. Databáza bude konzistentná v dlhodobej prespektíve a „stale“

dáta[1] sú povolené. Niektoré NoSQL databázy už dovoľujú aj transakcie, ako napríklad MongoDB.

NoSQL technológie

NoSQL databázy riešia mnohé problémy, a preto je každá zameraná na špecifickú podmnožinu funkcií. Vo všeobecnosti sa dajú rozdeliť na sklad kľúč-hodnota, databázy dokumentov, sklady so širokými stĺpcami a grafové databázy [31].

Sklad kľúč-hodnota Tieto systémy ukladajú hodnoty a ich indexy, parameter podľa ktorého sa dajú nájsť, na základe programátorom definovaného kľúča [26].

Sklad so širokými stĺpcami Tieto systémy ukladajú rozšírené záznamy, ktoré môžu byť rozložené vertikálne aj horizontálne naprieč uzlami. V niektorých publikáciách sú tiež nazývané sklady rozšírených záznamov [26].

Grafová databáza Tieto systémy ukladajú záznamy ako uzly usporiadané v grafe a referencie medzi nimi. Sú k dispozícií dotazy nad stromami uzlov a efektívne distribuované úložisko [26].

Databáza dokumentov Tieto systémy ukladajú dokumenty, ktoré sú indexované. Vďaka jednoduchým dotazom je možné vyberať správne dáta [26].

Pre svoj návrh a následnú implementáciu som si vybral práve databázu typu databáza dokumentov. Forma dokumentov najlepšie reprezentuje entity zo špecifikácie definovanej v kapitole 2.3. Bezschémovosť databázy zabezpečuje, že každý dokument môže vyzerať úplne inak, vďaka čomu je možné ďalej dodefinovávať potrebné informácie vzhľadom na potreby aplikácie.

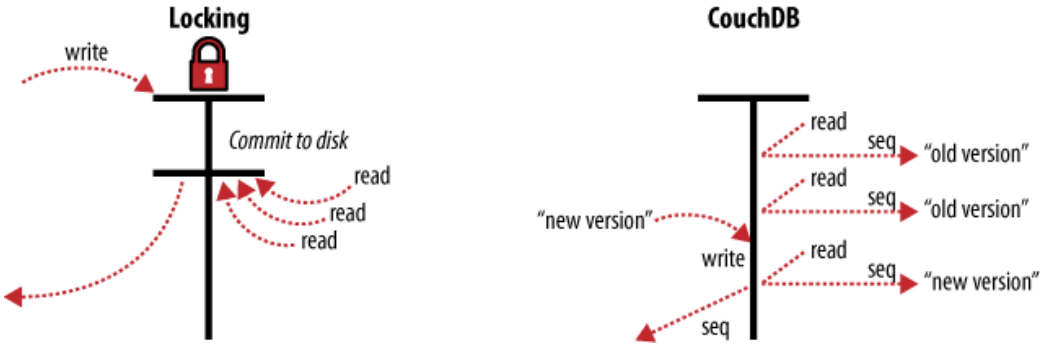
CouchDB

Spomedzi desať najpopulárnejších databáz orientovaných na dokumenty som si vybral databázu CouchDB[2]. Zastrešuje ju spoločnosť Apache Software Foundation pod open-source licenciou. Posledná verzia 2.1.1 je z novembra 2017, z čoho usudzujem, že sa jedná o stále živú a rozvíjajúcu sa technológiu [4]. Podporuje všetky známe operačné systémy a replikáciu master-master aj master-slave. Pre komunikáciu využíva protokol HTTP a všetky dáta sú uložené vo formáte JSON. JavaScript Object Notation je štandardizovaný formát dát založený na podmnožine programovacieho jazyka JavaScript [1]. Práve tieto vlastnosti spĺňajú požiadavky na škálovateľnosť a prenosnosť platformy opisované v kapitole 2.2.

Pri tradičných relačných databázach, je každá akcia súčasťou kontroly konzistencie databázy. Oproti tomu CouchDB obetuváva okamžitú konzistenciu za nárast výkonu. Pre zaistenie konzistencie pri kontrole súbežnosti nepoužíva tzv. lock, slovensky zámok, na práve aktualizované pristupované dáta, ktorý umožní prístup k dátam iba jednému klientovi. Namiesto toho, sú dokumenty v databáze verzované. V prípade požiadavky na zmenu dokumentu sa vytvorí úplne nová verzia, ktorou sa prepíše starý dokument, no len za predpokladu, že v požiadavke bol kód najnovšej verzie - nazývaný `_rev`. Po uvedenom úkone existujú v databáze dve verzie dokumentu, stará a upravená nová. Prístup databázy

[1] Artefakt cachovania, pri ktorom sa stáva, že dáta v cache nie sú najnovšou verziou potvrdených dát.
[2] Dostupné na http://couchdb.apache.org/

CouchDB je nazývaný MVCC (Multi-Version Concurrency Control, v preklade verzované konkurenčné riadenie), rozdiely medzi lock systémom a MVCC ilustruje obrázok 3.1. [24]



Obr. 3.1: Rozdiel medzi MVCC a klasickým lock systémom. Obrázok prevzatý z [24]

3.2.3 Návrh štrukúry dokumentov v databáze CouchDB

Podľa kritérií špecifikovaných v sekcii 2.3 som navrhol entity, „Aplikácia“ a „Pozvánka“, a tiež štruktúry ich dokumentov. Rozhodol som sa vytvoriť entity len pre aplikáciu a pozvánku, nakoľko som vyhodnotil, že všetky nutné informácie som schopný uchovávať len v týchto dvoch entitách.

Prístup jednotlivých používateľov budem uchovávať priamo v dokumente aplikácie, čím docielim, že nie je potrebné vytvárať dokument aj pre jednotlivých používateľov. Administrátor je rozpoznávaný na základe jeho emailovej adresy, tak ako aj všetci používatelia, to znamená, že administrátora, rovnako ako aj používateľov, môžem pridať rovno k dokumentu aplikácie.

Na základe môjho rozhodnutia, nevytvoriť dokumenty pre jednotlivých používateľov, bolo nutné extrahovať dáta o pozvánke do oddeleného dokumentu. V dokumente je okrem odosielateľa a príjemcu potrebné uložiť aj referenciu na aplikáciu, goe-lokačné dáta a časové metadáta.

Pre zabránenie kolízií, som sa rozhodol ponechať databázou generovaný identifikátor UUID - universally unique identifier, v preklade univerzálne unikátny identifikátor, ktorý reprezentuje 128-bitový objekt. Pri generovaní podľa štandardných postupov je šanca, že budú vygenerované dva rovnaké identifikátory blížiaca sa k nule [30].

Keďže je identifikátor náhodne generovaný, bolo by problematické rozlišovať medzi dokumentami aplikácie a pozvánky. Preto som pridal vlastnosť, `type`, v preklade typ, ktorá mi pri rozoznávaní pomôže.

Využitiu návrhu sa viac venujem v kapitole 4.1.1, kde opisujem transformáciu a implementáciu nižšie navrhnutých dokumentov.

Aplikácia

```
{
  "_id": "Generovane UUID",
  "_rev": "Generovane ciselne oznacenie revizie",
  "type": "app",
  "name": "Meno aplikacie",
```

```
  "admin": {
    "name": "Meno administratora",
    "mail": "Email administratora"
  },
  "users": {
    "Email pouzivatela": "Pocet este volnych, neposlanych, pozvanok",
    .
    .
    .
  }
}
```

Pozvnánka

Odoslaná pozvánka

```
{
  "_id": "Generovane UUID",
  "_rev": "Generovane ciselne oznacenie revizie",
  "type": "inv",
  "appID": "Meno aplikacie",
  "from": "Email pouzivatela, ktory pozvanku odoslal",
  "to": "Email pouzivatela, ktoremu bola pozvanka zaslana",
  "sent": "Datum odoslania"
}
```

Prijatá pozvánka

```
{
  .
  .
  .
  "accepted": {
    "date": "Datum priatia",
    "location": "Geo-lokacne informacie"
  }
}
```

3.3 Návrh back-end časti

V kapitole 3.1 som predstavil pojem full-stack a model MVC. Oba princípy využijem pri výbere vhodnej množiny obsahujúcej návrhové vzory spolu s implementovanými funkcionalitami. Takúto množinu nazývame framework [32].

3.3.1 Využitie JavaSciptu za pomoci Node.js

Aby som dokázal využívať JavaScript aj pri serverovej časti, musím použiť Node.js[3]. Jedná sa o runtime postavený na „V8 JavaScript engine“ z webového prehliadača Chrome. Jeho

[3]Dostupné na https://nodejs.org/en/

kľúčové vlastnosti patrí, že využíva udalosťami riadený, neblokujúci vstup-výstupný model a open source licencia. Vďaka tomu o ňom hovoríme ako o minimalistickom a efektívnom runtime. Nakoľko je Node.js neblokujúci, nemusím sa obávať dead-locku. Ten nemôže nastať, pretože vôbec nevyužíva zámky a takmer žiadna funkcia nevykonáva priamo vstupno-výstupné procesy. Vďaka spomínaným vlastnostiam prostredia Node.js, je škálovateľnosť platformy veľmi rozumne dosiahnuteľná [12].

3.3.2 Middleware framework Koa

Ak hovoríme o prostredí Node.js ako o serverovej časti platformy, je potrebné zadefinovať pojem middleware. Podľa slovníka je jeho význam nasledovný: „*počítačový program, ktorý má sprostredkovateľskú funkciu medzi rôznymi časťami aplikácie a operačného systému*“[4][5]. V prípade Node.js je možné definíciu presnejšie špecifikovať ako framework, ktorý má prístup ako k HTTP požiadavke tak aj k HTTP odpovedi.

Za middleware framework som si zvolil framework Koa[5], ktorý je licencovaný ako open source. Ten je v Európe a v ďalších západných krajinách zatiaľ menej známy. Populárny je najmä v Číne [25]. Koa je navrhnutý tímom ľudí, ktorí stáli aj za momentálne najpopulárnejším middleware frameworkom Express. Jeho cieľom je byť minimalistickejší, expresívnejší a viac robustný základ pre webové aplikácie a API ako je Express. Elegantnosť a minimalita vyradili akúkoľvek middleware funkcionalitu z jadra frameworku. Nahradzujú ju úzko špecifikované balíčky, pomocou ktorých si programátor vyskladá finálny framework len z preňo podstatných častí [10].

3.3.3 Návrh štruktúry API

API, alebo aj application programming interface (aplikačné programové rozhranie), je rozhranie, ktoré hovorí o tom, akou formou môže jeden softvér komunikovať s druhým. Prenos sústredenia vývoja softvéru z desktop aplikácii na webové a mobilné aplikácie priniesol zvýšený dopyt po webových API. Moderné webové API nie je určené len pre integráciu systémov medzi organizáciami. Práve naopak, podporuje zdieľanie dát, open source princípy a vytváranie komunít [29]. Výhodou webového API je využívanie architekrúry rozhrania REST (Representational State Transfer = Reprezentatívny prevod stavu). Koncept REST zahŕňa množinu konštánt a vlastností založených na protokole HTTP. Za pomoci metód GET, POST, PUT, DELETE, ktoré reprezentujú zmenu stavu systému, používateľ prechádza aplikáciou volením odkazov, ako napríklad `/app/test_app`. Výsledkom je ďalší nový stav. Použitím REST API som schopný zabezpečiť prenosnosť a nezávislosť na systéme [35].

Pri návrhu API musím počítať s faktom, že jeho forma sa bude časom meniť. Pre zaručenie spätnej kompatibility využijem číslovanie verzií. Všetky požiadavky smerujúce na koncové body API sú predchádzané prefixom `/api/0`, kde `0` reprezentuje číslo verzie.

3.4 Návrh front-end časti

Pri návrhu grafického rozhrania platformy som začal samotným dizajnom a grafickým rozhraním ako takým. GUI som navrhoval metódou „paper prototyping“, čo v preklade znamená vytváranie prototypov na papier. V zásade ide o seba vysvetľujúcu metódu, kedy

[4] Vlastný preklad z anglického originálu
[5] Dostupné na http://koajs.com/

dizajnér kreslí skice obrazoviek na papier. Papierový dizajn navrhnutý vysvetlenou metódou je priložený v prílohe A.

Rozhodol som sa, že pre zjednodušenie návrhu použijem open source sadu nástrojov. Najpoužívanejším toolkitom je sada nástrojov Bootstrap[6] zastrešovaná sociálnou sieťou Twitter. Obsahuje rôzne responzivné systémy mriežky pre usporiadanie komponentov na obrazovke, vopred vytvorené komponenty a preddefinované premenné. Prílišné venovanie sa grafickej stránke nie je predmetom návrhu tejto webovej aplikácie, nakoľko farbu tlačidiel a tomu podobné vlastnosti je možné kedykoľvek zmeniť bez zásahu do funkcionality. Preto som sa rozhodol využiť vopred vyladenú šablónu obsahujúcu potrebné komponenty podľa návrhu na papier. Šablóna mi naviac dopomôže k rýchlejšej orientácii sa vo zvolenom frameworku.

3.4.1 Reaktívny prístup

Reaktívne programovanie je programové paradigma zaoberajúce sa dátovými tokmi a propagáciou zmeny dát. Programovanie front-endu sa zaoberá takmer výlučne propagáciou dát a ich zmeny. Všetky dátové toky sú asynchrónne volania. Práve kvôli týmto spomínaným faktom, je reaktívny prístup využívaný pri tvorbe front-endu webových aplikácií. Reaktívne aplikácie stoja na štyroch princípoch. Sú „responsive“, v preklade rýchlo reagujúce, „scalable“ (škálovateľné) a zároveň „resilient“ (odolné) a nakoniec ich architektúra je „message-driven“, slovensky architektúra poháňaná správami. Princíp rýchlo reagujúcej aplikácie hovorí, že aplikácia rýchlo reaguje na všetkých svojich používateľov za akýchkoľvek podmienok s cieľom priniesť konzistentný pozitívny používateľský zážitok. Rýchlosť a pozitívny zážitok za všetkých podmienok zabezpečujú princípy škálovateľnosť a odolnosť. Vďaka architektúre poháňanej správami je možné, aby tento systém fungoval vo svete asynchrónnych volaní.

Framework Vue.js

Na základe popularity som sa rozhodoval pre jeden z troch najpopulárnejších frameworkov. Zoradené od najpopulárnejšieho sú to menovite React, Angular 1 a Vue.js [25]. Všetky spomenuté frameworky implementujú reaktívny prístup k tvorbe single-page (jedna stránka) aplikácií. O webovej aplikácií hovoríme ako o single-page aplikácii ak jej interakcia s používateľom spočíva v prepisovaní jednej stránky, namiesto presmerovávania na nové stránky servera [23]. Vybral som si Vue.js[7] na základe mnou stanovených špecifikácií v kapitole 3.1. Vue.js je open source framework zameraný na prispôsobivosť. Rovnako ako aj Koa, je aj Vue oveľa populárnejší vo východných krajinách, ako Čína. Tento fakt môže byť spôsobený tým, že originálny autor, Evan You, pochádza z Číny. Je možné povedať, že Vue.js spája funkcionalitu frameworkov React a Angular, zatiaľ čo si zachováva svoju minimalicistickosť. Na rozdiel od dvoch populárnejších frameworkov, pri Vue.JS je možne používať čistý JavaScript a nie je vyžadovaná odo mňa nadmnožina programovacieho jazyka JavaScript ako TypeScript ci JSX.

Vue.js pristupuje k reaktívnemu prístupu pomocou deklaratívneho vykresľovania. Všetky dáta a DOM (Document Object Model = Objektový model dokumentu)[8] sú prepojené a každá zmena dát vyvolá prekreslenie elementu v DOM. Toto prepojenie je realizované

[6] Dostupné na https://getbootstrap.com/

[7] Dostupné na https://vuejs.org/

[8] Pri načítaní webovej stránky prehliadač vytvorí objektovy model dokumentu, DOM, ktorý je reprezentovaný stromovou štruktúrou elementov.

pomocou „template“ syntaxe, slovensky šablónovej syntaxe. Celá aplikácia sa nachádza v koreňovej Vue inštancii, voliteľne organizovanej do stromovej štruktúry subkomponentov. Pri vytvorení inštancie sa všetky dáta patriace k danej inštancii pridajú do reaktívneho systému. Zmenou hodnoty jednej z vlastností systém reaguje na vyvolanú zmenu a aktualizuje element novou hodnotou.

Spolu s výberom front-end frameworku som si vybral aj šablónu. Táto šablóna by mala obsahovať minimum funkcionality spolu s množstvom už vopred nadizajnovaných komponent. Rozhodol som sa použiť open source šablónu „Vue Light Bootstrap Dashboard“[9] založenú na knižnici Bootstrap.

Vuex

Vuex je model riadenia stavov a knižnica pre Vue.js aplikácie. Slúži ako centralizovaný zdroj dát pre všetky komponenty v aplikácii. Spolu s definovanými pravidlami zaručuje, že stav dát môže byť zmenený len za predpokladaných okolností.

Pri zložitejších a robustnejších aplikáciách môže na jednom stave závisieť viacero zobrazení a akcie z rôznych zobrazení chcú modifikovať rovnaký stav. Preto Vuex extrahuje všetky zdieľané stavy mimo komponenty, kde ich spravuje globálne ako singleton. Vďaka tomu môžeme stromovú štruktúru komponentov považovať za jedno veľké zobrazenie, v ktorom môže ktorýkoľvek komponent pristupovať k zdieľanému stavu alebo spúšťať akcie, ktoré následne menia stavy, bez ohľadu na to, kde v stromovej štruktúre sa nachádza.

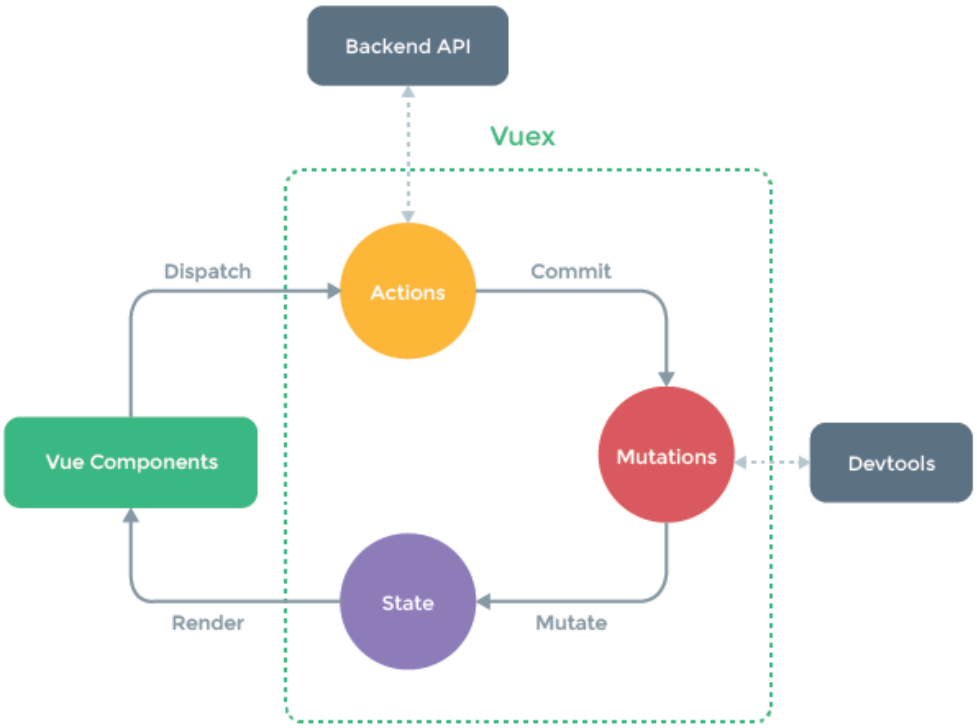
Základom každej Vuex aplikácie je „store“. Jedná sa kontajner obsahujúci stavy aplikácie. Tieto kontajnery sú reaktívne, čo znamená, že v prípade ak komponent závisí na stave, bude reaktívne a efektívne aktualizovaný podľa nového stavu. Nie je dovolené priamo meniť stavy v kontajnery. Vykonať zmeny je možné len za pomoci mutácií. Mutácie nám zabezpečujú, že každá zmena stavu zanecháva spätne vyhľadateľnú stopu. Na obrázku 3.2 je model zobrazujúci aplikáciu využívajúcu knižnicu Vuex. Vue komponenty sú renderované podľa stavu. Následne je v jednej alebo viacerých komponentoch spustená akcia, v ktorej aplikácia posiela požiadavku na koncový bod API. Akcia vyvolá mutáciu a mutácia následnú zmenu stavu [18].

3.5 Zabezpečenie

Zabezpečenie platformy vyvára bezpečnostnú vrstvu, na ktorú sa môžu prípadné aplikácie používajúce platformu spoľahnúť. Dôležitým faktorom preto ostáva zabezpečenie pred nepovoleným vstupom alebo užívaním aplikácie. Administrátor musí byť overovaný a zároveň komunikácia platformy s aplikáciou administrátora musí byť istou formou zabezpečená.

Všetka komunikácia si preto bude vyžadovať autorizačný token. Bearer token, formálnejšie nazývaný autorizačný token je časťou autentifikácie, ktorá je autorizačnou schémou protokolu HTTP. Využívaná schéma OAuth 2.0 je štandardný protokol na autorizáciu, ktorý sa zameriava na jednoduchosť pri poskytovaní špecifických tokov autorizácie pre webové aplikácie, desktopové aplikácie či mobilné aplikácie. Tento protokol a jeho rozšírenia sa vyvíjajú v rámci pracovnej skupiny IETF OAuth. Token pozostáva z kryptovaného reťazca, najčastejšie generovaného na serverovej časti aplikácie. Najbežnejší postup obnáša server, ktorý vygeneruje token pre používateľa ako odpoveď na požiadavku s prihlásením používateľa do aplikácie. Takto autorizovaný používateľ je následne povinný odoslať token

[9] Dostupné na https://github.com/cristijora/vue-light-bootstrap-dashboard



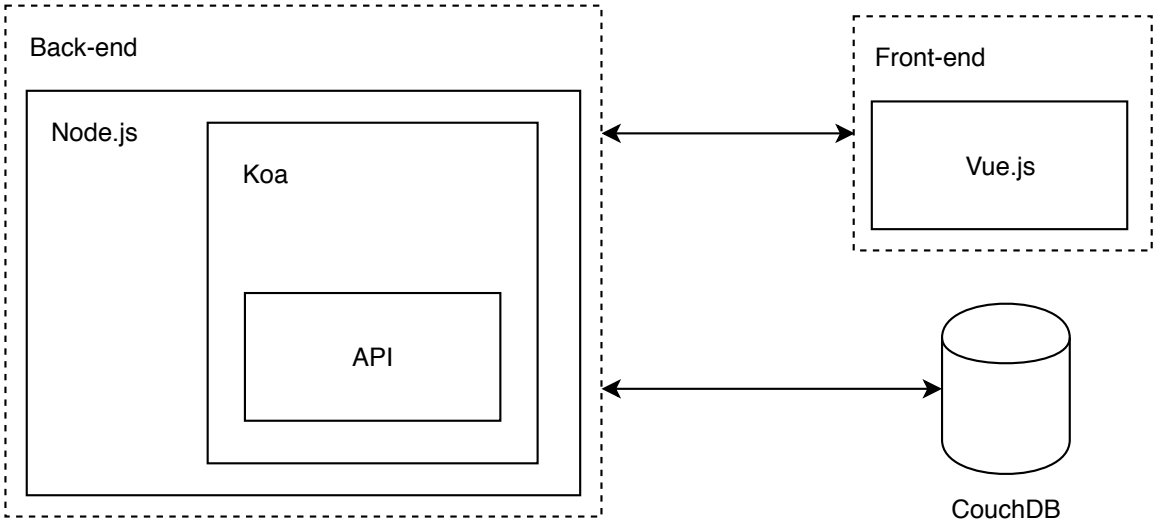
Obr. 3.2: Model zobrazujúci aplikáciu využívajúcu knižnicu Vuex. Obrázok prevzatý z [18]

v autorizačnej hlavičke počas všetkých jeho nasledujúcich požiadaviek. Tento token môže byť náhodný reťazec znakov alebo kódovaná štruktúra JWT (JSON Web Token) [7].

Takéto zabezpečenie však nie je zďaleka dostatočné. Pre útočníka je až priveľmi jednoduché odchytiť komunikáciu, a teda aj požiadavku spolu s autentifikačným tokenom. Token môže následne zneužiť ako falošnú identitu pre vlastné účely. Práve pre ochranu pred odchytením autentifikačného tokenu je nutné pri používaní autorizačnej schémy bearer autentifikácia používať šifrovaný protokol HTTPS. Obe protokoly, teda ako TLS tak aj SSL, fungujú na princípe asymetrickej kryptografie. HTTPS vytvára zabezpečené spojenie na nezabezpečenej sieti. Vyššie spomínané spojenie garantuje prijateľnú ochranu pred odchytením a útokom man-in-the-middle za predpokladu, že je použitá vhodná sada šifier a server oplýva platným a dôveryhodným certifikátom [20]. Pre správny chod, ako už bolo naznačené, je potrebný platný a overený certifikát. Tento certifikát však nie je možné získať ľahko a zadarmo. Preto v svojej bakalárskej práci zanedbám HTTPS a budem používať obyčajný nezabezpečený protokol HTTP.

3.6 Model návrhu platformy

Obrázok 3.3 zobrazuje model platformy podľa návrhu.



Obr. 3.3: Finálny model návrhu platformy.

Kapitola 4

Implementácia návrhu aplikácie

Podľa návrhu aplikácie, ktorý som opisoval v kapitole 3, som vytvoril webovú aplikáciu, ktorej model je zobrazený na obrázku 3.3. Špecifikácie platformy, podľa ktorých som postupoval pri návrhu aj pri implementácií, som uviedol v kapitole 2. Pri implementácii platformy som používal operačný systém Linux s distribúciou Fedora 27.

4.1 Implementácia databázového systému

Vybral som si najvyššiu dostupnú stálu verziu databázy CouchDB. V čase vytváranie aplikácie to bola verzia 2.1.1. Keďže som sa rozhodol pracovať na distribúcií Fedora 27, ako som uviedol v predchádzajúcom odseku, musel som databázu inštalovať kompilovaním zdrojových súborov. Softvérový balík pre mnou zvolenú distribúciu v tomto čase neexistoval. Z bezpečnostného hľadiska je odporúčané vytvoriť nového používateľa, pod ktorým bude databáza spúšťaná. CouchDB ponúka praktický spôsob správy a konfigurácie databázy za pomoci webovej aplikácie Fauxton, ktorú som využíval pri všetkých konfiguráciách databázy. Fauxton ponúka grafické užívateľské prostredie prostredníctvom webovej aplikácie v prehliadači. Databázu CouchDB je možné nakonfigurovať pre produkčné používanie buď ako „single node“ (jeden/osamotený uzol), alebo ako „cluster“ (skupina serverov). Pri ostrom zavadzaní by som volil cluster, čo je aj dôvod výberu databázy CouchDB. Cluster umožňuje škálovať databázu do šírky tak ako som to špecifikoval v požiadavkach (viď. 2.2). V testovacom prostredí úplne postačuje konfigurácia single node. Odporúčané bezpečnostné nastavenia hovoria o nastavení administrátora pre databázu. Bez tohto nastavenia, môže ktokoľvek zasielať požiadavky na databázový server a získať tak informácie, ktoré mu nepatria. Do premenných prostredia (enviroment variables) som nastavil administrátorov účet a jeho heslo. Toto opatrenie mi umožňuje využívať databázu prakticky a zároveň zabezpečene.

4.1.1 Úprava návrhu dokumentov databázy

Počas implementácie back-endu a front-endu som zistil, že návrh dokumentov z kapitoly 3.2.3 nie je dostačujúci. Nebolo možné zobrazovať všetky dáta a prístup bol značne komplikovaný. Práve preto som návrh počas implementácie zmenil z dvoch entít na štyri entity. Používateľov som extrahoval z entity aplikácia a vytvoril som každému vlastný dokument. Naviac som pridal entitu pre zoznam čakateľov (waiting list), ktorý som počas návrhu zanedbal. Dokument pozvánky zostal nezmenený. Implementovanú úpravu návrhu dokumentov je možné vidieť nižšie.

Aplikácia

```
{
  "_id": "Meno aplikacie",
  "_rev": "Generovane ciselne oznacenie revizie",
  "type": "app",
  "admin": "Email administratora",
  "listsize": Maximalna velkost zoznamu cakatelov
}
```

Zoznam čakateľov

```
{
  "_id": "Generovane UUID",
  "_rev": "Generovane ciselne oznacenie revizie",
  "type": "waitlist",
  "app": "Meno aplikacie",
  "users": [
    "Email pouzivatela",
    .
    .
    .
  ]
}
```

Používateľ

```
{
  "_id": "Email",
  "_rev": "Generovane ciselne oznacenie revizie",
  "type": "user",
  "password": "Hash pouzivatelskeho hesla",
  "apps": [
    {
      "id": "Meno aplikacie",
      "used_inv": Pocet pouzitych pozvanok,
      "free_inv": Pocet zostavajucih pozvanok
    },
    .
    .
    .
  ]
}
```

Pozvánka

Viď. 3.2.3

4.1.2 Zobrazenia

Upravené dokumenty už obsahovali všetky potrebné informácie potrebné pre zobrazenie administrátorovi, či používateľovi v grafickom používateľskom prostredí. CouchDB dovoľuje vytvorenie špeciálnych dokumentov, ktoré slúžia ako zobrazenia v relačnom databázovom systéme. Pre potreby platformy som vytvoril niekoľko takýchto dokumentov.

Na uvedenom príklade nižšie prehľadávam všetky dokumenty typu používateľ a vytváram pseudo-dokumenty, kde kľúč dokumentov je názov aplikácie a hodnota je štruktúra obsahujúca dáta o používateľovi. Pre zobrazenie všetkých používateľov v danej aplikácii stačí zaslať požiadavku s filtrom obsahujúcim hľadaný kľúč, teda názov aplikácie.

```
function (doc) {
  if(doc.type === "user") {
    for (app in doc.apps) {
      emit(
        doc.apps[app].id,
        {
          id: doc._id,
          used: doc.apps[app].used_inv,
          free: doc.apps[app].free_inv
        }
      )
    }
  }
}
```

4.2 Back-end

Node.js udržiava softvérový balíček pre distribúciu Fedora 27. Jeho inštalácia je preto veľmi jednoduchá. Node.js je sám o sebe veľmi obmedzený. Zabezpečuje len nutnú funkcionalitu pre správny chod webového servera. Pokiaľ chce developer využiť JavaScriptový modul, čo predstavuje softvérovú knižnicu, musí použiť správcu balíčkov. Pre správu balíčkov sa štandardne používa nástroj npm[1]. Npm je správca JavaScriptových balíčkov a zároveň najväčší register týchto balíčkov. Alternatívou je správca balíčkov yarn[2]. Yarn využíva paralelizovaný prístup k sťahovaniu balíčkov, čo značne urýchľuje inštaláciu nových balíčkov. Tomu napomáha aj fakt, že už stiahnuté balíčky ukladá do cache pamäte a pri nasledujúcej inštalácii, už nie je potrebné balík opätovne sťahovať. Vyššiu rýchlosť dokazuje viacero porovnaní [13]. Na základe týchto faktov som sa rozhodol použiť, ako správcu balíčkov, yarn.

Navzdory tomu, že komunikácia s databázou CouchDB prebieha pomocou REST API, rozhodol som sa pre jednoduchosť používať balíček nano[3]. Spomínaný balíček je oficiálnou knižnicou pre databázy CouchDB pre Node.js. Počas používania tohto balíčku som narazil na obmedzenia. Nano nepodporuje novú funkciu JavaScriptu, operátor `await`. JavaScript pre potreby synchronizácie asynchrónnych volaní používa „callback“. Asynchrónny callback je spúšťateľný kód, ktorý sa posiela ako argument inému kódu. Kód z argumentu sa spustí po až po určitom čase [21]. Pri vyššom počte zanorených callbackov vzniká tzv. „callback

[1] Dostupné na https://www.npmjs.com/
[2] Dostupné na https://yarnpkg.com/en/
[3] Dostupné na https://github.com/apache/couchdb-nano

hell“ (callback peklo). Tento stav značne narúša čitateľnosť kódu. Objekt `Promise` ponúka alternatívny prístup. `Promise` reprezentuje konečný stav asynchrónneho volania. Tento objekt využíva operátor `await`, ktorý je možno použiť len v asynchrónnej funkcii. Operátor `await` umožňuje pozastaviť exekúciu asynchrónnej funkcie až do momentu, kedy je objekt `Promise` naplnený. To znamená, že asynchrónne volanie je úspešne dokončené alebo skončilo s chybou. Kvôli tomuto obmedzeniu som počas implementácie prešiel na balíček nano-blue[4]. Jedná sa o knižnicu nano pozmenenú pre využívanie objektov `Promise`.

Pri každej zmene kódu je potrebné webový server reštartovať, aby boli zmeny viditeľné. Preto som pri spúšťaní servera použil namiesto softvéru npm utilitu nodemon[5]. Nodemon čaká na akúkoľvek zmenu v kóde a automaticky reštartuje webový server.

4.2.1 Koncové body rozhrania API

V rámci back-endu som implementoval aplikačné rozhranie, ktorého návrhu som sa venoval v kapitole 3.3.3. Implementované API je znázornené v zjednodušených tabuľkách 4.1 a 4.2. Vďaka knižnici koa-bodyparser[6] platforma podporuje telo (body) požiadaviek kódované ako application/json, application/x-www-form-urlencoded a text/plain. Telo HTTP odpovedí na požiadavky je vo väčšine prípadov application/json. V niektorých prípadoch obsahuje telo len celé číslo.

Prefix `/api/0`	GET	POST

`/users`	`/{USER}`
	`/{USER}/adminto`
	`/{USER}/apps`
	`/{USER}/waitlist`
	`/{USER}/sentinvs`
		`/{USER}`

`/apps`	`/{APP}`
	`/{APP}/users`
	`/{APP}/users/count`
	`/{APP}/invs/sent`
	`/{APP}/invs/available`
	`/{APP}/waitlist`
	`/{APP}/waitlist/occupied`
		`/{APP}`

`/inv`		

Tabuľka 4.1: Znázornenie REST API pre požiadavky GET a POST.

Vytváranie API jednoznačne uľahčuje utilita nodemon spomenutá v predchádzajúcom odseku. Počas implementácie aplikačného rozhrania som potreboval vidieť spätnú väzbu k vytváraným zmenám. Pre tento účel som použil prostredie Postman[7]. Postman je softvér, ktorý pomáha pri vytváraní REST API. Obsahuje nástroje pre testovanie, vytváranie mockupov, vytváranie a zdieľanie dokumentácie a ďalšie. Vďaka praktickému grafickému užívateľskému prostrediu je vytváranie HTTP požiadaviek v prostredí Postman maximálne zjednodušené. Využívanie softvéru Postman mi umožnilo vytvoriť sadu HTTP volaní, ktorú som ďalej využíval pri debugovaní a testovaní API.

[4]Dostupné na https://github.com/bdchauvette/nano-blue
[5]Dostupné na https://nodemon.io/
[6]Dostupné na https://github.com/koajs/bodyparser
[7]Dostupné na https://www.getpostman.com/

Prefix `/api/0`	PUT	DELETE
`/users`	`/{USER}/{APP}/sendinv/{INVITED}`	

`/apps`	`/`
	`/{APP}/users/{USER}`
	`/{APP}/users/{USER}/newinvs/{SIZE}`
	`/{APP}/waitlist/acceptall`
	`/{APP}/waitlist/{MAIL}`
		`/{APP}`
		`/{APP}/users/{USER}`
		`/{APP}/waitlist/{MAIL}`

`/inv`	`/{ID}`	`/{ID}`

Tabuľka 4.2: Znázornenie REST API pre požiadavky PUT a DELETE.

4.2.2 Implementovaná funkcionalita API

Mnou implementované REST API obsahuje značnú časť funkcionality oproti špecifikáciám z kapitoly 2.3. Dôvod je zapuzdrenie alebo zanedbanie niektorých prípadov užitia. Administrátor alebo systém je schopný založiť nový program a zároveň zrušiť starý program, pridať používateľa do programu pomocou používateľovej emailovej adresy alebo odstrániť používateľov, zobraziť si detaily o používateľovi a upraviť používateľovi počet pozvánok. Ďalej je administrátor alebo systém schopný vylistovať počet používateľov v programe, celkový počet poslaných pozvánok v programe a celkový počet ešte voľných pozvánok. Administrátor má možnosť vyžiadať si API token prostredníctvom REST API a systém vie overiť zaslaný token. Čo sa týka zoznamu čakateľov, systém alebo administrátor dokáže zmeniť veľkosť zoznamu, prijať vybraných alebo všetkých používateľov čakajúcich na zozname do programu a vylistovať počet obsadených miest v zozname.

Používateľ má možnosť registrovať sa a následne je schopný vidieť všetky aplikácie, do ktorých má prístup, alebo v ktorých je prípadne administrátorom. Samozrejme používateľovi je umožnené zasielať pozvánky, prijať pozvánku alebo ju odmietnuť, vidieť, komu každému používateľ už zaslal pozvánku v rámci programu a zapísať sa na zoznam čakateľov.

4.3 Front-end

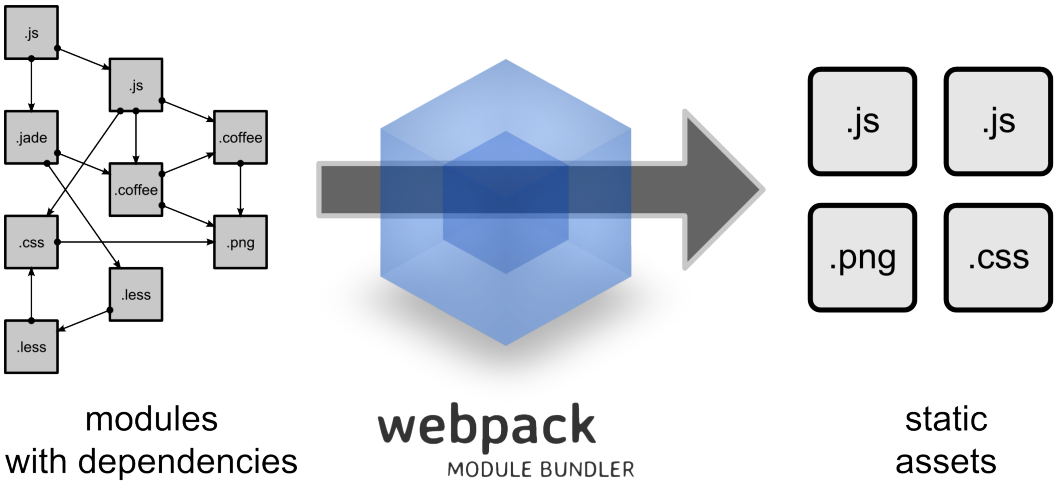
Pri návrhu aplikácie som sa chcel opierať o model MVC (viď. 3), no výberom reaktívneho prístupu a frameworku Vue.js, so zameraním na tvorbu single-page webových aplikácií, nie je model MVC úplne najvhodnejší. Pri vytváraní single-page webových aplikácií je oveľa vhodnejšia voľba modifikácia spomínaného modelu, model Model-View-ViewModel (Model-Zobrazenie-Model zobrazenia), známym pod akronymom MVVM. V modeli MVC pôsobí kontrolér ako nástroj, ktorý priamo upravuje dáta pre daný model. Tento prístup nie je vhodný pre modrené webové aplikácie navrhnuté ako single-page aplikácie, kde potrebuje front-end časť komunikovať priamo s back-endom časťou. Naproti tomu ViewModel z modelu MVVM nepôsobí ako kontrolér, ale zastrešuje spojenie medzi dátami zobrazenia a dátami modelu. Takéto spojenie je nazývané data-binding (spojenie dát) a umožňuje priamu komunikáciu medzi užívateľským prostredím, teda front-endom a serverom, teda back-endom [11].

Framework Vue.js je z časti inšpirovaný modelom MVVM. Vo Vue.js sa všetky komponenty skladajú z troch častí, a to `template`, `script` a `style`. V časti `tempalte` developer definuje za pomoci HTML štruktúru komponentu, do časti `script` vloží logiku ako pristupovanie k backend API a nakoniec v časti `style` môže developer upraviť výzor komponentu. Komponenty je možné zanorovať a vytvárať tak stromovú štruktúru. Najjednoduchší prí-

stup k implementácii aplikácie s frameworkom Vue.js je vytvárať práve spomínané stromové štruktúry. Takýto prístup môže fungovať pri menších aplikáciách kde sa JavaScript využíva pre vylepšenie len zopár zobrazení. Ak je celá aplikácia poháňaná JavaScriptom vynárajú sa nedostatky. Konkrétne programátor je nútený vytvárať unikátne identifikátory pre každý komponent, template časť je definovaná len za pomoci string reťazca a vyžaduje si škaredé zalomenia a časť style je absenčná. Pri väčších aplikáciách ako je táto paltforma je oveľa výhodnejšie zvoliť prístup single-file (v jednom súbore) komponentov. Single-file komponenty s príponou `.vue` spájajú všetky tri spomínané časti do jedného súboru. Vďaka prístupu single-file komponentov sa vyššie spomenuté nedostatky eliminujú. Princíp oddelenia zodpovednosti sa v tomto prípadne nerovná oddeleniu súborov. Práve kvôli vyššie spomenutým faktom som implementoval front-end časť aplikácie ako stromovú štruktúru single-file koponentov.

Balíčkovač Webpack

Viac súborov a veľa statických komponentov značne komplikuje vývoj a následné zavádzanie aplikácie. Preto som sa rozhodol použiť nástroj Webpack[8]. Webpack je možné opísať ako balíkovač statických modulov určený pre JavaScriptové aplikácie. Pri spracovávaní aplikácie vytvára interný graf dependencií, v ktorom mapuje každý modul potrebný pre aplikáciu a generuje jeden alebo viac balíčkov. To znamená, že všetky statické obrázky a referencie na komponenty Webpack prepíše na aktuálnu cestu k daným komponentom a následne všetko zjednotí do pár súborov. Opisované spracovávanie je demonštrované na obrázku 4.1. Webpack prichádza s implementovaným serverom - „dev server“. Jedná sa o malú aplikáciu bežiacu na frameworku Express, ktorá spúšťa front-end server. Tento server sa automaticky reštartuje rovnako ako nodemon (viď. 4.2) a dovoľuje tak pohodlné vytváranie a debuggovanie používateľského prostredia za pochodu. V rámci dev servera som za pomoci proxy nastavil prístup k back-end serveru na rovnaký port, ako na ktorom beží aj front-end.



Obr. 4.1: Model zobrazujúci funkciu nástroja Webpack. Obrázok prevzatý z [19]

sub

[8]Dostupné na https://webpack.js.org/

HTTP klient Axios

Vytvoreniu dizajnu a naprogramovaniu komponentov nadchádza načítanie údajov. Tento úkon sa vykonáva za pomoci HTTP klienta. Niektoré webové frameworky majú vstavaného klienta, ako napríklad Angular 2 alebo JQuery. Vue.js obsahoval taktiež vstavaného HTTP klienta, no od verzie 2.0 ho nepodporuje v prospech knižníc tretích strán. Takou knižnicou je aj Axios[9]. Jedná sa o HTTP klienta, využívajúceho objekty Promise (viď. 4.2), pre prehliadače a Node.js.

Využitie šablóny

Ako som zmienil v kapitole 3.4.1, rozhodol som sa použiť front-end šablónu, ktorá už obsahuje množstvo CSS súborov popisujúcich tvar komponentov, „Vue Light Bootstrap Dashboard“. Šablónu som upravil podľa vlastných potrieb a vytvoril tak prehľadný dashboard ako pre administrátora tak aj pre používateľa. Screenshoty z aplikácie je možné vidieť v prílohe B. Konkrétnym pohľadom zobrazených na obrázkoch sa venujem v časti „Implementovaná funkcionalita“ v tejto kapitole (viď. 4.3).

Problémy

Počas implementácie front-end časti platformy som narazil na problém so stavmi vo Vuex inštancii (viď. sekcia Vuex v kapitole 3.4.1). Mnou implementovaný stav `currentApp` (práve vybratá aplikácia) bol menený v komponente horného menu. Potreboval som však na túto zmenu reagovať v obsahu stránky, čo je v mojom prípade iný komponent. Problém som nakoniec vyriešil pomocou Vuex funkcie `watch state`. Pri využití spomínanej funkcie, sa pri vytvorení komponentu zároveň vytvorí aj tzv. strážca, ktorý reaguje na zmenu stavu pričom nezáleží v akom komponente táto zmena nastala.

Ďalším mnou riešeným problémom bol neperzistentný stav v lokálnej pamäti prehliadača. Tento problém sa prejavoval pri každom znovu načítaní stránky, a to tak, že sa lokálna pamäť premazala a aplikácia sa s tým nevedela vysporiadať. Problém sa mi podarilo vyriešiť využitím Vue.js pluginu vuex-persistentstate[10]. V kombinácií s cookies som bol schopný nevyžiadane správanie eliminovať. Aplikácia si naviac pamätá svoj stav aj po reštartovaní webového prehliadača.

4.3.1 Implementovaná funkcionalita

Grafické používateľské prostredie umožňuje komukoľvek vytvoriť účet. Tento účet nie je presne definovaný ako administrátorský alebo používateľský. Každý používateľ môže začať nový program, alebo sa prihlásiť do zoznamu čakateľov. Prepínanie medzi administrátorskými funkciami a používateľskými je vykonávané pomocou drop-down menu (menu, ktoré sa rozvíja smerom dole). Táto akcia je zachytená na obrázku B.4. Ako administrátor tak aj používateľ má možnosť zvolenia si aktívnej aplikácie zo zoznamu jeho aplikácií. Aktívna aplikácia je tá, pre ktorú sa zobrazujú všetky informácie. Možnosť prepínania medzi aplikáciami je demonštrovaná na obrázku B.5.

Administrátor má k dispozícií informácie vo forme sady widgetov, ktoré sú pripnuté na dashboarde (v prípade tejto aplikácie, domovská stránka). V prehľadných kartách sú zo-

[9] Dostupné na https://github.com/axios/axios
[10] Dostupné na https://github.com/robinvdvleuten/vuex-persistedstate

brazené informácie o stave používateľov, počte zaslaných pozvánok, počte pozvánok, ktoré sú ešte k dispozícii a percentuálnom naplnení zoznamu čakateľov. Pri pohľade na dashbaord, obrázok B.2 a B.3, nasledujú widgety zobrazujúce štatistiky. Tieto widgety nie sú implementované a sú zobrazené len ich mockupy. V dolnej časti domovskej obrazovky je administrátor schopný založiť nový program alebo pridať ďalšieho používateľa. Prepnutím v ľavom menu na záložku „Users“ (používatelia) sa zobrazí tabuľka používateľov pre danú aplikáciu. Tu môže administrátor meniť počet pozvánok jednému alebo viacerým používateľom naraz alebo odstrániť vybraných používateľov. Zoznam používateľov je zobrazený na obrázku B.6. Ďalšie zobrazenie, „Waiting list“ (zoznam čakateľov) zobrazuje modifikovaný komponent z predchádzajúceho zobrazenia. Administrátor tak môže vidieť zoznam čakateľov a priať vybraných. Spomínané zobrazenie sa nachádza na obrázku B.7. Posledné zobrazenie administrátora ukazuje nastavenia programu. Administrátorovi je dovolené zmeniť veľkosť zoznamu čakateľov alebo email administrátora a zrušiť ľubovoľný program. Uvedené zobrazenie je možné vidieť na obrázku B.8.

Používateľské zobrazenie, obrázok B.9, je veľmi podobné tomu administrátorskému. Podobné widgety zobrazujú počet ešte voľných pozvánok, počet už zaslaných pozvánok a zoznam pozvaných používateľov. Dashboard dopĺňa karta umožňujúca pozvanie nového používateľa. Po prepnutí zobrazenia výberom záznamu „Waiting lists“ (zoznamy čakateľov), sa môže používateľ zapísať do ľubovoľných nezaplnených zoznamov. Táto možnosť je zachytená na obrázku B.10. Obrázkok B.11 ukazuje zobrazenie zakladania prvého programu.

4.4 Testovanie

Pre debugovanie a testovanie som používal nástroj Postman. Ako som písal v kapitole 4.2.1, nástroj Postman umožňuje vytvoriť sadu HTTP požiadaviek za pomoci grafického užívateľského prostredia. Keďže som tento nástroj používal pri debugovaní, rozhodol som sa ho využiť aj pri testovaní. Zo sady HTTP volaní, ktoré som vytváral počas implementácie, som vytvoril sadu unit testov, ktoré na seba nadväzujú. Takéto testy nie sú v praxi najlepším riešením. Pri zlyhaní testu sú takmer všetky nasledujúce testy ovplyvnené a preto rovnako zlyhajú. Tento problém som objavil až pri testovaní. Postman síce umožňuje korigovať workflow testov, a teda preskočiť niektoré testy pri zlyhaní konkrétnych testov, no neumožňuje vytvoriť setup a teardown fázy. Bez použitia setup a teardown fázy nie je možné zabezpečiť stále nové a izolované prostredie. Taktiež bolo obťažné porovnávať informácie zanorené v štruktúre JSON. Možnosť porovnať celú štruktúru neprichádza do úvahy, pretože vďaka záznamu revízie, bude výsledná štruktúra stále odlišná. Preto som testoval len status kód odpovede. Do budúcna by bolo vhodnejšie použiť nástroje určené len na testovanie a kontinuálnu integráciu. Automatické testovanie používateľského prostredia je náročné a venuje sa tomu viacero publikácií a nástrojov. Rozhodol som sa nepoužiť žiaden nástroj a ani som nevytvoril žiadnu sadu testov. Front-end časť som testoval manuálne v priebehu implementácie platformy.

4.5 Návrh budúceho rozvoja

Táto kapitola sumarizuje rozšírenia aplikovateľné na implementovanú webovú aplikáciu, ktoré nedostali priestor počas návrhu alebo implementácie. Práca nemala za úlohu vytvoriť platformu pripravenú na komerčné použitie, ale navrhnúť a implementovať open source variant, ktorý je ďalej možné rozširovať komunitou. Práve z tohto dôvodu, som sa venoval

len aspektom nutným pre úspešnú implementáciu kostry platformy. Počas implementácie a testovania som narazil na zopár obmedzení alebo vylepšení, ktorých riešenie by zlepšilo používateľskú skúsenosť.

Geo-lokačné a štatistické údaje

V návrhu hovorím o ukladaní a filtrovaní geo-lokačných dát. Zber týchto dát som implementoval v REST API back-end časti, no prezentácia nazberaných dát nebola implementovaná. Bolo by vhodné podporovať túto funkciu. Štatistiky podľa demografie umožňuje selektívny výber používateľov z konkrétnej krajiny a následne rozšírenie v danej krajine pomocou rozšírenia pozvánok vybratým používateľom. Zobrazenie detailných štatistických údajov ako graf vývinu stavu používateľskej základne môže pomôcť administrátorovi s rozhodovaním ako ďalej škálovať aplikáciu alebo so zistením trendu popularity aplikácie.

Zasielanie emailov

Aplikácia v súčasnom stave generuje pozývací odkaz po zaslaní pozvánky, no neposiela reálne emaily. Súčasťou nasadenej paltformy určite musí byť aj fungujúci systém zasielania emailov obsahujúcich vygenerovaný pozývací odkaz.

Zvoliť obľúbenú aplikáciu

Po prihlásení do aplikácie je v prípade, že používateľ má prístup do viacero aplikácií alebo je administrátorom viacerých aplikácií, za aktívnu aplikáciu zvolená prvá podľa abecedy. Takéto chovanie nemusí vyhovovať každému používateľovi. Už pri testovaní som zistil, že väčšinou sa chcem pozerať len na jednu aplikáciu a pokiaľ nie je prvá v abecede, musím si ju zvoliť za aktívnu. Cookies zabezpečujú, že po zvolení mnou preferovanej aplikácie, táto aplikácia ostane aktívnou aj po reštartovaní prehliadača. Problém nastáva pri odhlásení a následnom prihlásení alebo pri prepínaní medzi administrátorským dashboardom a používateľským dashbaordom. Tento problém by mohla vyriešiť voľba obľúbenej aplikácie, ktorá by prevážila výber podľa abecedy.

Pozývať ľudí v skupinách

Ako administrátor môže používateľ platformy pozývať ľudí len po jednom. V prípade, že ide o novo založený program, mohla by byť žiadaná funkcia pozývania ľudí v skupinách.

Integrácia sociálnych sietí a emailových klientov

Registrácia a prihlasovanie sa do webovej aplikácie je zaistené pomocou emailovej adresy a hesla. V dnešnej dobe, kedy ľudia využívajú sociálne siete čím ďalej tým viac, sa žiada registrácia a prihlasovanie za pomoci účtov zo sociálnych sietí ako Facebook, GitHub, LinkedIn. Nie všetci ľudia však obľubujú sociálne siete. Títo používatelia by mohli mať možnosť pozvať ľudí priamo zo svojho emailového adresára.

Mobilné rozhranie

Použitá šablóna obsahuje mobilné rozhranie, ktoré nebolo v rámci práce implementované. Zvýšený dopyt po mobilných aplikáciách a webových aplikáciách uspôsobených pre mobilné

zariadenia je badateľný všade. Práve kvôli tomuto faktu sa žiada implementovať aj mobilné rozhranie.

Kapitola 5

Záver

Cieľom tejto práce bolo oboznámiť sa s princípmi tvorby webových aplikácií a potrebnými technológiami pre ich tvorbu. Následne využiť získané vedomosti pri návrhu a implementácií platformy. Funkcionalitu vytvorenej aplikácie analyzovať a vytvoriť návrh budúceho rozvoja. V rámci tejto práce som vytvoril webovú aplikáciu, ktorá slúži ako platforma pre správu používateľov. Práca nemala za úlohu vytvoriť platformu pripravenú na komerčné použitie, ale navrhnúť a implementovať open source variant, ktorý je ďalej možné rozširovať komunitou.

Mnou navrhnutá a implementovaná aplikácia umožňuje zakladať nové programy v rámci ktorých môže administrátor sledovať stav používateľskej základne, stav odoslaných aj neodoslaných pozvánok v programe a naplnenie zoznamu čakateľov. Do programu je možné priamo pridávať alebo pozývať nových používateľov, prijímať vybraných používateľov zo zoznamu čakateľov. Ako administrátor môže používateľ upravovať počet pozvánok pre každého používateľa v jeho programe. Nový používatelia majú prístup k aplikáciám, do ktorých boli pozvaní a smú do nich pozývať nových používateľov. Každý používateľ má možnosť zapísať sa na zoznam čakateľov.

Hodnotím, že ciele boli v rámci tejto práce dostatočne naplnené ako pri oboznámení sa s princípmi tvorby webových aplikácií a otrebnými technológiami tak aj pri následnom návrhu, implementácii a zhodnotení funkcionality spolu s návrhom budúceho rozvoja.

Literatúra

[1] Introducing JSON. [Online; cit. 2018-04-30].
URL https://www.json.org

[2] The Linux Information Project. 2006, [Online; cit. 2018-04-27].
URL http://www.linfo.org/vendor_lockin.html

[3] Model View Controller. 2014, [Online; cit. 2018-04-30].
URL http://wiki.c2.com/?ModelViewController

[4] DB-Engines. 2018, [Online; cit. 2018-04-28].
URL https://db-engines.com/en/ranking/document+store

[5] Dictionary.com. 2018, [Online; cit. 2018-05-01].
URL http://www.dictionary.com/browse/middleware?s=t

[6] Firebase. 2018, [Online; cit. 2018-04-27].
URL https://firebase.google.com/

[7] The general HTTP authentication framework. 2018, [Online; cit. 2018-05-02].
URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

[8] Hoodie. 2018, [Online; cit. 2018-05-05].
URL http://hood.ie/

[9] HotFrameworks. 2018, [Online; cit. 2018-04-30].
URL https://hotframeworks.com

[10] Koa. 2018, [Online; cit. 2018-05-01].
URL http://koajs.com/

[11] MVC vs. MVVM: How a Website Communicates With Its Data Models. 2018, [Online; cit. 2018-05-14].
URL https://hackernoon.com/mvc-vs-mvvm-how-a-website-communicates-with-its-data-models-18553877bf7d

[12] Node.js. 2018, [Online; cit. 2018-05-01].
URL https://nodejs.org/

[13] npm-vs-yarn. 2018, [Online; cit. 2018-05-10].
URL https://github.com/appleboy/npm-vs-yarn

[14] Parse Server. 2018, [Online; cit. 2018-05-05].
URL http://parseplatform.org/

[15] Referral Saasquatch. 2018, [Online; cit. 2018-04-27].
URL https://www.referralsaasquatch.com/

[16] ReferralCandy. 2018, [Online; cit. 2018-04-27].
URL https://www.referralcandy.com

[17] Rocket Referrals. 2018, [Online; cit. 2018-04-27].
URL https://www.rocketreferrals.com/

[18] Vuex. 2018, [Online; cit. 2018-05-01].
URL https://vuex.vuejs.org/en/

[19] Webpack. 2018, [Online; cit. 2018-05-14].
URL https://webpack.github.io/assets/what-is-webpack.png

[20] What is HTTPS? 2018, [Online; cit. 2018-05-02].
URL https://www.instantssl.com/ssl-certificate-products/https.html

[21] Wikipedia, the free encyclopedia: Callback (computer programming). 2018, [Online; cit. 2018-05-11].
URL https://en.wikipedia.org/wiki/Callback_(computer_programming)

[22] Wikipedia, the free encyclopedia: Front and back ends. 2018, [Online; cit. 2018-04-30].
URL https://en.wikipedia.org/wiki/Front_and_back_ends

[23] Wikipedia, the free encyclopedia: Single-page application. 2018, [Online; cit. 2018-05-13].
URL https://en.wikipedia.org/wiki/Single-page_application

[24] Anderson, J. C.; Lehnardt, J.; Slater, N.: *CouchDB: The Definitive Guide*. O'Reilly Media, Inc., prvé vydanie, 2010, ISBN 978-0-596-15589-6, 272 s., [Online; cit. 2018-04-28].

[25] Benitte, R.; Greif, S.; Rambeau, M.: The State of JavaScript. 2017, [Online; cit. 2018-05-01].
URL https://stateofjs.com/2017/introduction/

[26] Cattell, R.: Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, 2011: s. 12–27, ISSN 0163-5808, doi:10.1145/1978915.1978919, [Online; cit. 2018-04-28].

[27] Chandra, D. G.: BASE analysis of NoSQL database. *Future Generation Computer Systems*, 2015: s. 13–21, [Online; cit. 2018-04-28].

[28] Codd, E.: A relational model of data for large shared data banks. *Communications of the ACM*, 1970: s. 377–387, ISSN 0001-0782, doi:10.1145/362384.362685, [Online; cit. 2018-04-28].

[29] Higginbotham, J.: *Designing Great Web APIs*. O'Reilly Media, Inc., prvé vydanie, 2015, ISBN 978-1-491-92459-4.

[30] Leach, P.; Mealling, M.; Salz, R.: A Universally Unique IDentifier (UUID) URN Namespace. 2005, doi:10.17487/RFC4122, [Online; cit. 2018-04-30].
URL https://tools.ietf.org/html/rfc4122

[31] Madison, M.; Barnhill, M.; Napier, C.; aj.: NoSQL Database Technologies. *Journal of International Technology and Information Management*, 2015: s. 1–I, ISSN 15435962, [Online; cit. 2018-04-28].

[32] Meiert, J. O.: *The Little Book of HTML/CSS Frameworks*. O'Reilly Media, Inc., prvé vydanie, 2015, ISBN 9781491920169.

[33] Molinaro, A.: *SQL : kuchařka programátora*. Computer Press, 2009, ISBN 978-80-251-2617-2, 573 s.

[34] Pratt, P. J.; Last, M. Z.: *Concepts of Database Management*. 8 vydanie, 2014, ISBN 978-1285427102, [Online; cit. 2018-04-28].

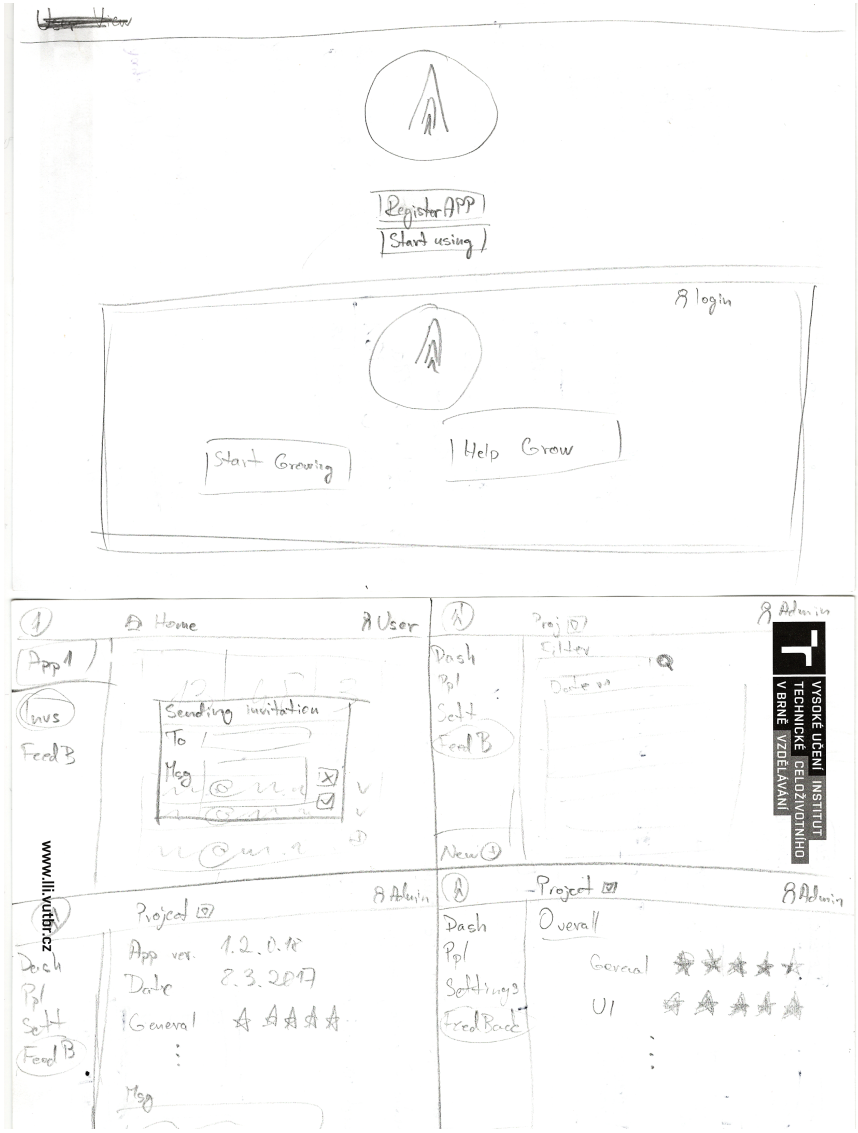
[35] Richardson, L.; Amundsen, M.; Ruby, S.: *RESTful Web APIs: Services for a Changing World*. O'Reilly Media, Inc., 2013, ISBN 978-1449358068, 406 s.

Prílohy

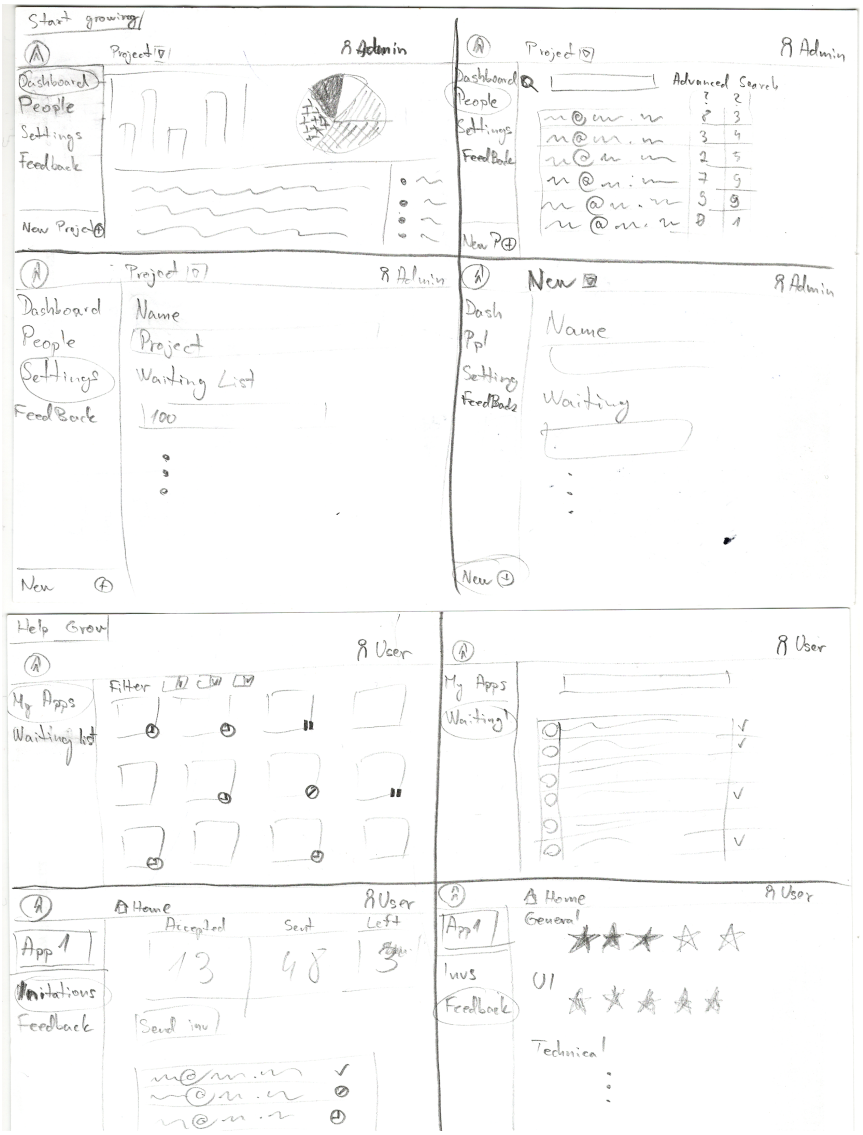
Príloha A

Návrh GUI

Papierový dizajn je zobrazený na obrázkoch A.1 a A.2.



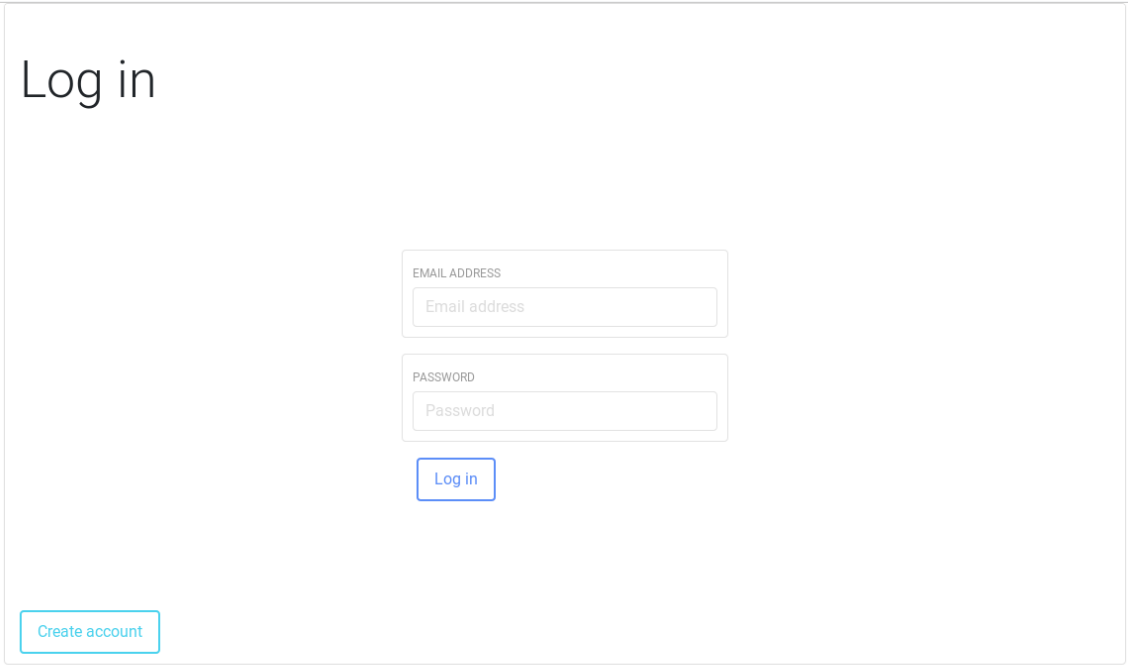
Obr. A.1: Úvodná obrazovka a zobrazenia dahsboardu.



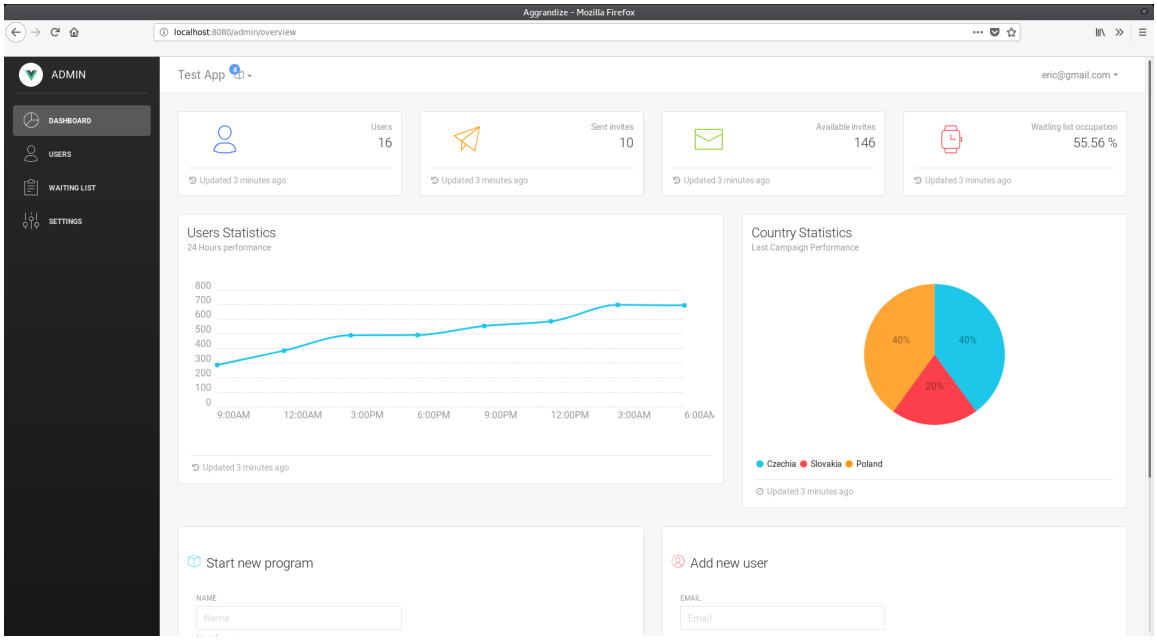
Obr. A.2: Dashboard zobrazenia.

Príloha B

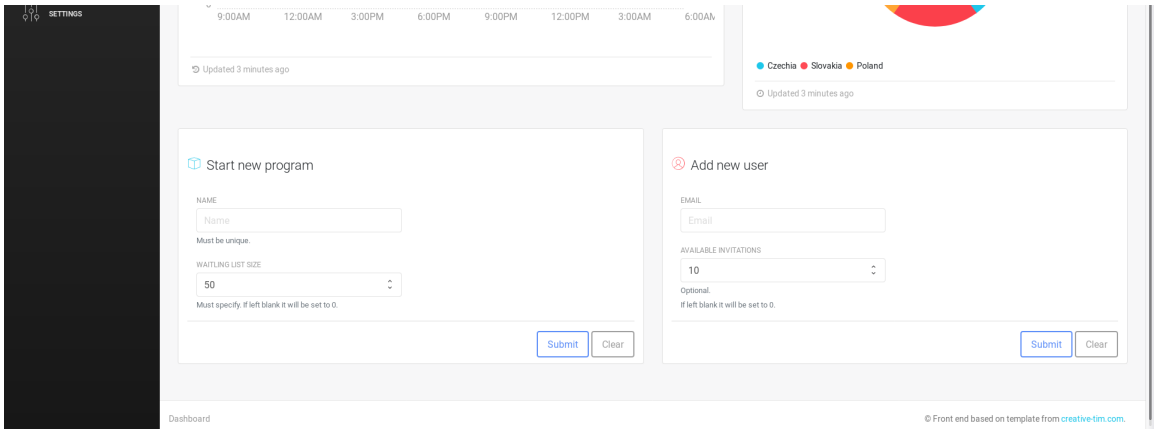
Screenshoty aplikácie



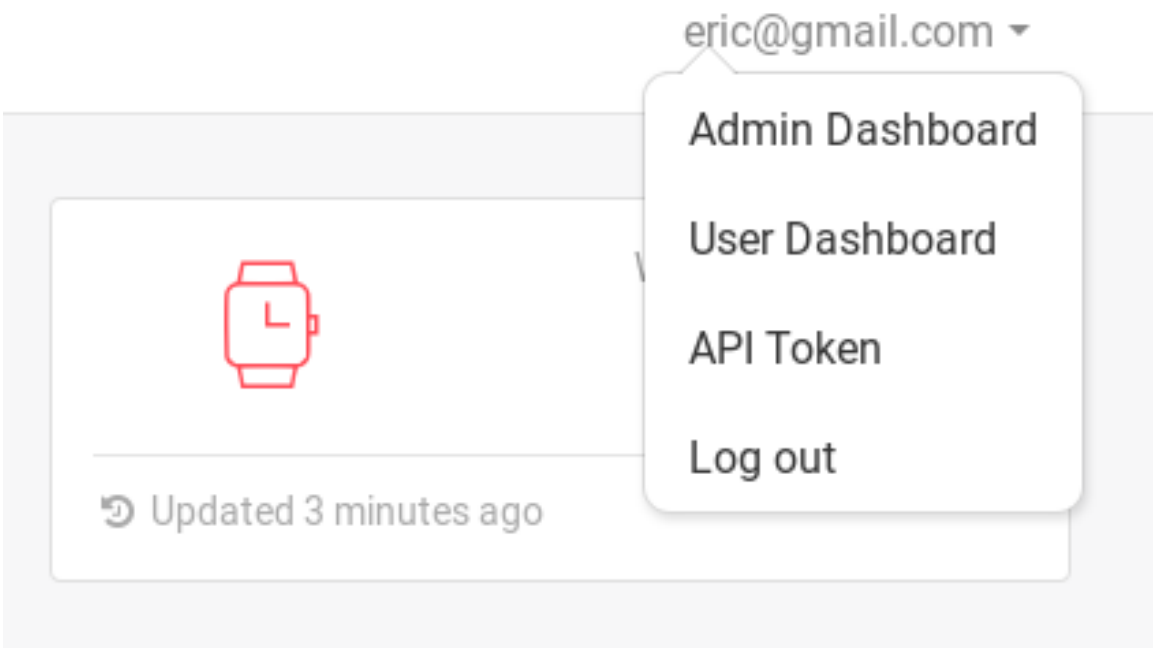
Obr. B.1: Úvodná obrazovka prihlásenia a registrácie.



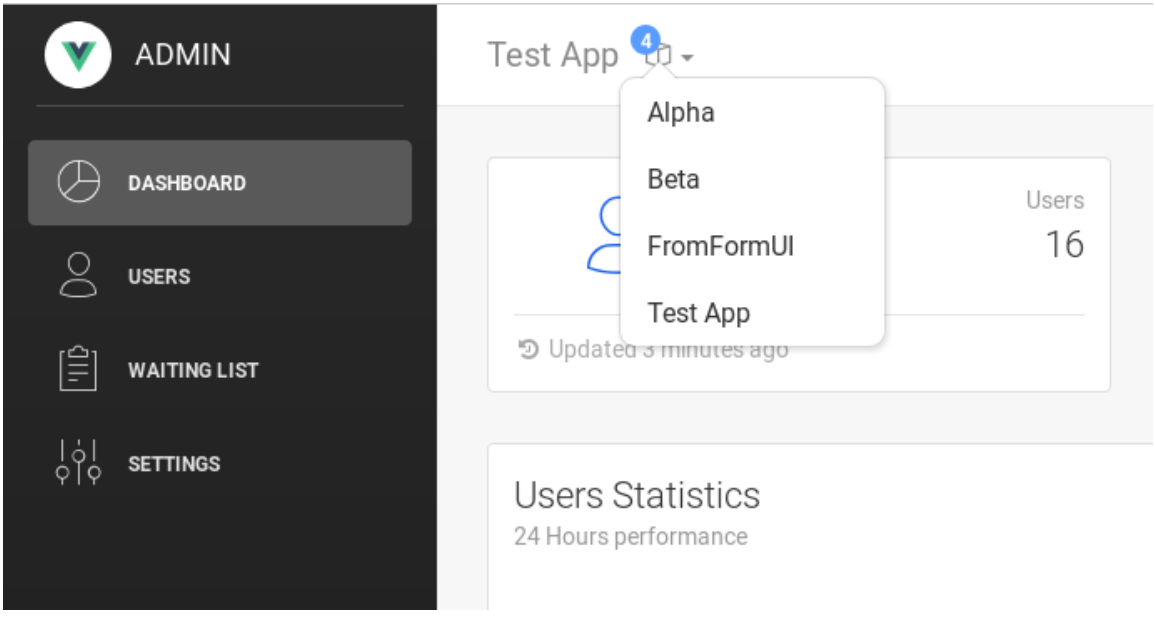
Obr. B.2: Úvodná obrazovka po prihlásení administrátora. Základné zobrazenie - dashboard.



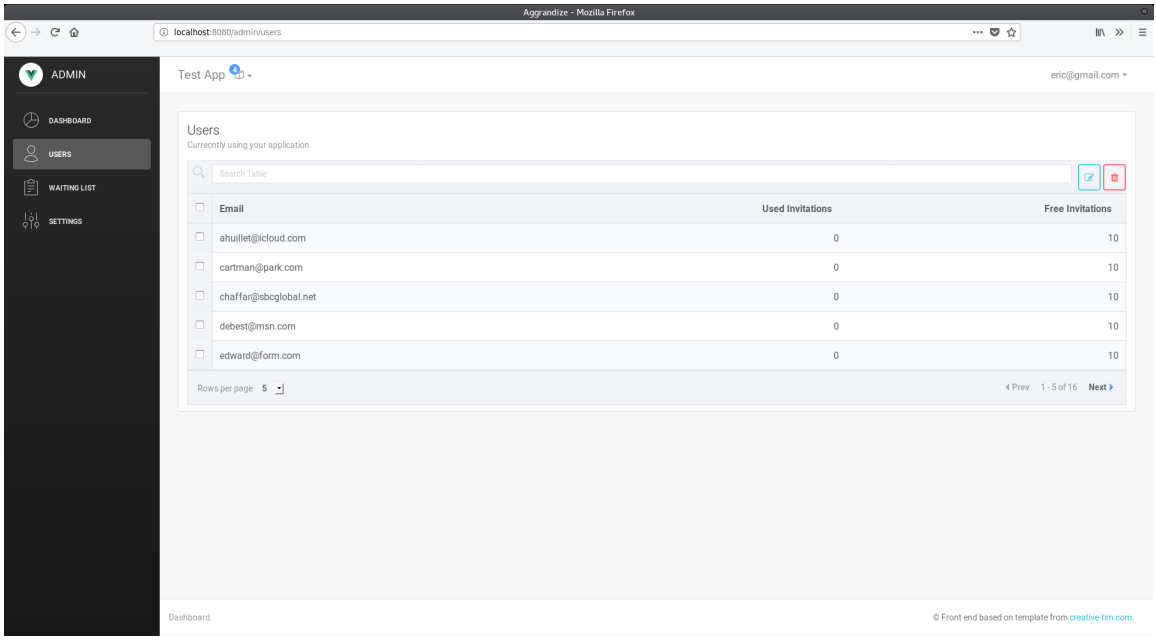
Obr. B.3: Pokračovanie dashboardu administrátora.



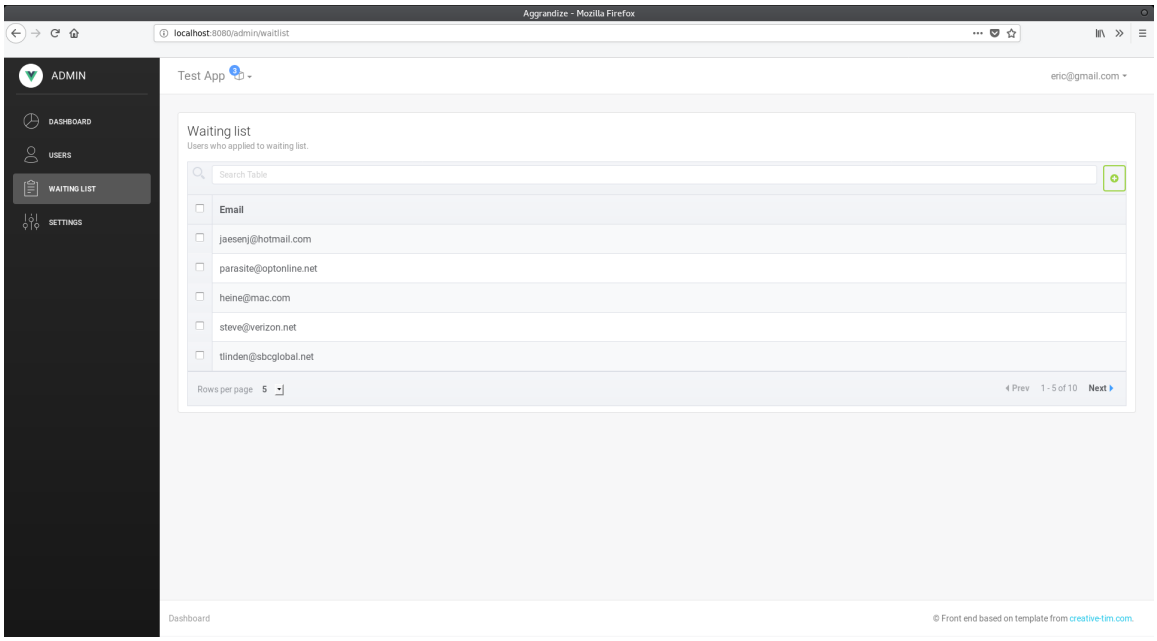
Obr. B.4: Používateľské menu admistrátora.



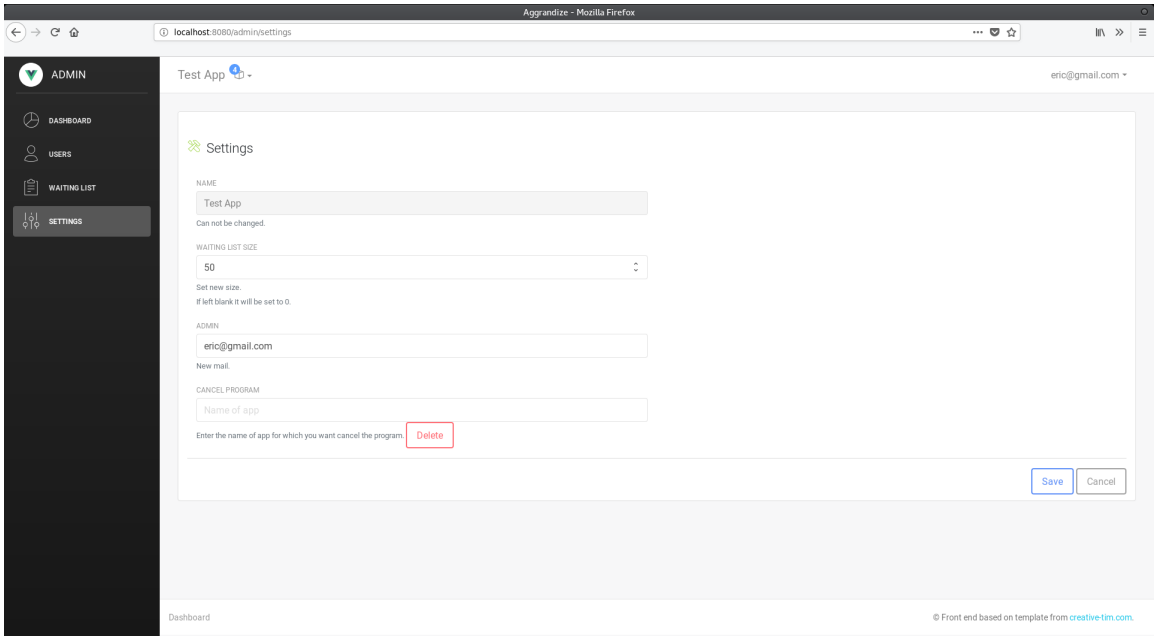
Obr. B.5: Výber aplikácie. Rovnaké pre administrátorské obrazovky aj pre používateľské.



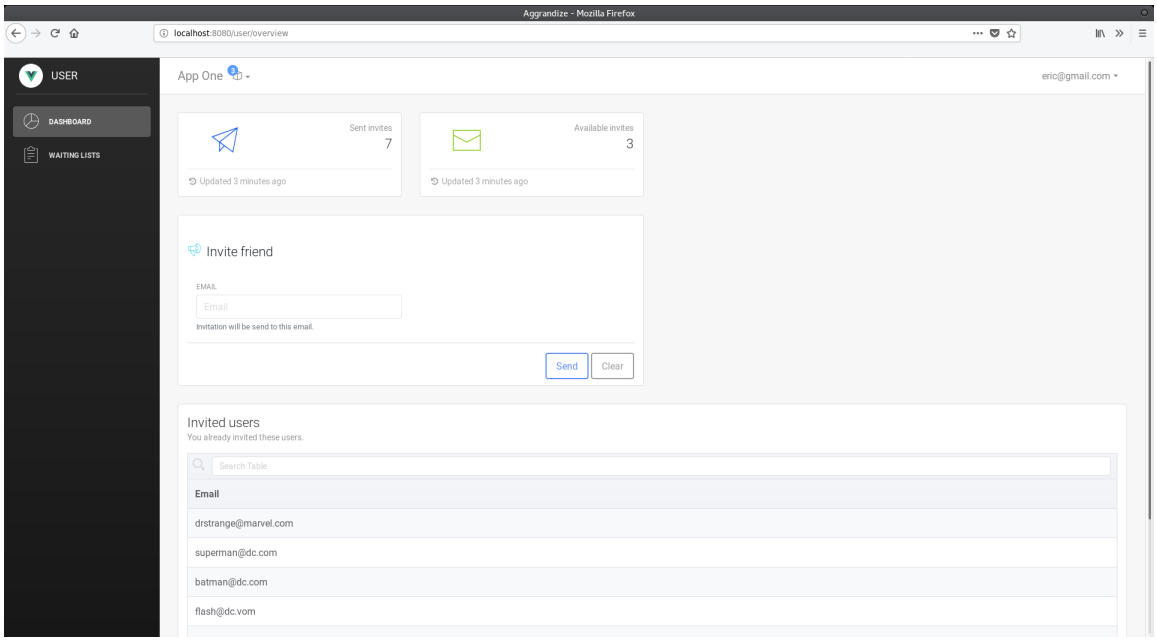
Obr. B.6: Zobrazenie používateľov vybranej aplikácie.



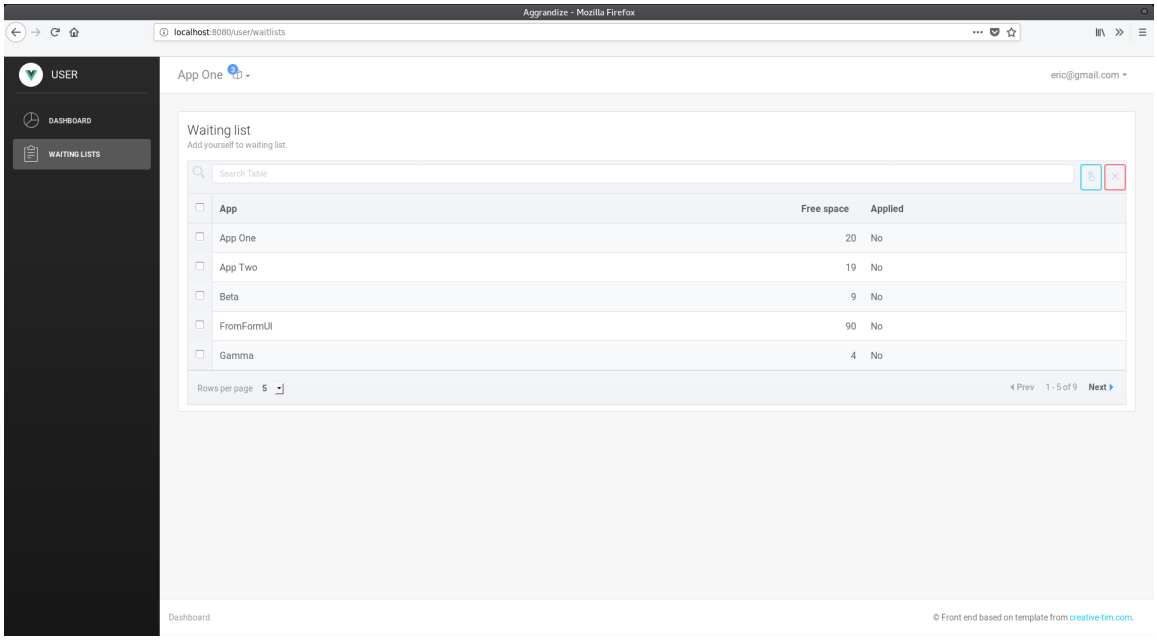
Obr. B.7: Zobrazenie zoznamu čakateľov vybranej palikácie.



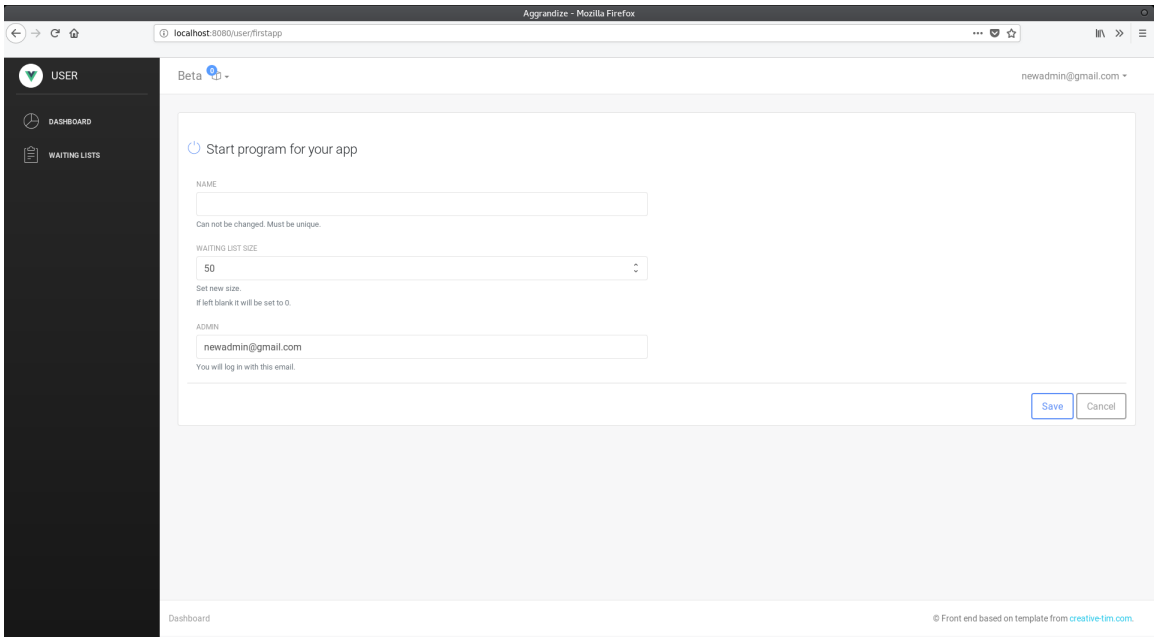
Obr. B.8: Zobrazenie nastavení vybranej aplikácie.



Obr. B.9: Úvodná obrazovka po prihlásení používateľa. Základné zobrazenie - dashboard.



Obr. B.10: Zobrazenie všetkých dostupných zoznamov čakateľov.



Obr. B.11: Zobrazenie vytvárania prvého programu.

Príloha C

Obsah priloženého pamäťového média

- `client` – zdrojové súbory front-end časti
- `docs` – API dokumentácia, model dokumentov databázy, prípady užitia, táto práca a jej zdrojové kódy
- `server` – zdrojové súbory back-end časti
- `.gitattributes` – konfiguračný súbor webovej aplikácie GitHub
- `.gitignore` – konfiguračný súbor systému verzovania git
- `LICENSE` – licenčný súbor; MIT open source licencia
- `README.md` – návod na inštaláciu a odkazy na dokumentáciu
- `Tests.postman_test_run.json` – výsledky spustených testov z nástroja Postman
- `aggrandize.postman_collection.json` – kolekcia HTTP požiadaviek pre nástroj Postman
- `db_views.js` – dokument obsahujúci všetky potrebné databázové zobrazenia.