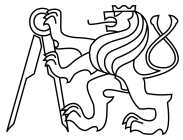
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Služba pro zasílání push notifikací
Student:	Bc. Jiří Levý
Vedoucí:	Ing. Jan Václavík
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Cílem práce je analyzovat, navrhnout a vytvořit službu pro zasílání push notifikací na mobilní platformy Android, iOS a Windows Phone. V této službě lze registrovat uživatele, z nichž každý může vytvořit notifikace pro neomezený počet aplikací. V uživatelském rozhraní serveru pak bude možno vložit příslušné klíče nutné pro zasílání notifikací ke konkrétní aplikaci a také zde ručně rozesílat notifikace nebo si prohlížet statistiky. Součástí služby bude také návrh a implementace API, pomocí něhož bude možno ovládat veškerou funkčnost této služby – registraci/odstranění zařízení, posílání notifikací (všem nebo pouze vybraným zařízením), plánování pravidelných rozesílek notifikací a podobně.

Hotová práce bude zveřejněna jako open-source na Githubu.

K implementaci bude použit NodeJS a JavaScript.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 17. února 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Služba pro zasílání push notifikací

Bc. Jiří Levý

Vedoucí práce: Ing. Jan Václavík

16. února 2017

Poděkování

Rád bych poděkoval panu Ing. Janu Václavíkovi za významnou pomoc při tvorbě diplomové práce. Mimo jiné bych chtěl poděkovat za jeho přívětivý přístup, ochotu a cenné rady, bez kterých by nebylo v mých silách práci dokončit.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 16. února 2017 .....

Odkaz na tuto práci

Levý, Jiří. *Služba pro zasílání push notifikací.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Hlavním tématem této diplomové práce jsou push notifikace. Jedná se o krátké informativní zprávy, které upozorňují uživatele cílového zařízení na nové události. Práce se zabývá problematikou odesílání těchto zpráv pro dané aplikace na cílové platformy Android, iOS a Windows Phone. Analyzuje jejich možnosti a následně implementuje službu pro zasílání těchto notifikací. Realizovaná služba obsahuje API pro vzdálené ovládání. Součástí práce je webový klient, který umožňuje registrovaným uživatelům spravovat aplikace a jejich notifikace. Definované notifikace jsou automaticky odesílány na cílová zařízení.

Klíčová slova Push notifikace, API, NodeJS, React, Android, iOS, Windows Phone

Abstract

The main subject of this master thesis are push notifications. Thiese short informative messages notify user devices about new events. The main goal of this work is managing and sending push notifications to the target applications on the Android, iOS and Windows Phone platforms. We are analyzing and implementing features for sending push notifications. Implemented service provides API for remote control of all features. Part of the result program is web client, which allows signed users to manage application and their notifications. Created notifications are sent to the targeted devices.

Keywords Push notifications, API, NodeJS, React, Android, iOS, Windows Phone

Obsah

Seznam obrázků

Úvod

V současné době, kdy má většina lidí neustále u sebe chytrý mobilní telefon s připojením k internetu, vedle telefonu používá tablety, chytré hodinky, notebooky a jiná přenosná zařízení, jsou push notifikace všeobecně známé a rozšířené. Jedná se o krátké textové zprávy, které se zobrazují na cílovém zařízení, jsou vždy přidružené k nějaké aplikaci a informují o nových událostech nebo jsou marketingového charakteru. Jak vlastně push notifikace vznikají? Tato práce se zabývá tématikou zasílání push notifikací na mobilní platformy Android, iOS a Windows Phone a implementuje jednoduchou službu, která umožňuje vytvářet a odesílat push notifikace na cílová zařízení. Součástí služby je veřejné API, pomocí něhož lze vzdáleně ovládat funkcionality celé služby. Pro uživatelsky přívětivé používání naší služby byl vytvořen také webový klient, který poskytuje základní funkcionality pro vytváření, správu a rozesílání push notifikací.

Kapitola 1

Cíl práce

Cílem této diplomové práce je vytvořit aplikaci pro zasílání push notifikací pro nejrozšířenější mobilní platformy Android, iOS a Windows Phone. Pro vytvoření aplikace je nejprve nutné provést analýzu konkurenčních řešení, poučit se ze slabin těchto projektů a inspirovat se jejich silnými stránkami, následně navrhnout samostatnou službu z technologického hlediska i z hlediska uživatelského rozhraní. Dalším hlavním cílem je návrh a implementace API, aby bylo možné celou aplikaci ovládat vzdáleně právě pomocí tohoto API. Výsledkem bude open-source projekt zveřejněný na Githubu.

1.1 Motivace

V dnešní době existuje spousta služeb, které nabízejí stejné nebo podobné funkcionality jako aplikace, která je cílem této práce. Jen nepatrné množství z nich je bohužel open-source, což znamená, že pokud by někdo chtěl k dispozici zdrojové kódy existujících řešení, aby nemusel začínat svůj projekt od nuly nebo aby mohl pouze rozšířit stávající řešení o nové funkce pro něho využitelné, nastává problém. Motivací je tedy v neposlední řadě vytvořit základ pro budoucí úpravy této aplikace či pokud bude někdo potřebovat pouze základní funkcionality, může využít přímo náš projekt.

Dalším bodem je přítomnost kompletní API, což umožňuje vývojářům jiných platforem využít náš projekt a implementovat pouze libovolného klienta. Jelikož se dnes rozmáhají mobilní technologie, vývojář může pomocí API našeho projektu například jednoduchým způsobem vytvořit mobilní aplikaci, která umožňuje správu push notifikací.

Použité technologie jsou moderní, jedná se o javascript, který se postupně dostává mezi špičku programovacích jazyků v oblasti webových technologií. Zejména pak nástroje pro tvorbu klientského kódu, kde se jedná o ReactJS nebo AngularJS. Pro serverovou část aplikace se jedná především o NodeJS a jeho knihovny. Poněvadž jsou tyto technologie obecně rozšířené a moderní,

nebude pro vývojáře problém orientovat se v našem open-source kódu, modifikovat jej ani nasazovat do produkce.

1.2 Požadavky

Požadavky na systém plynou ze zadání této diplomové práce. Cílová aplikace bude schopna zasílat push notifikace na mobilní platformy Android, iOS a Windows Phone. Push notifikace je krátká informační zpráva, která je zobrazená na mobilním nebo desktopovém zařízení, slouží hlavně pro upozornění cílového uživatele na novou událost, ovšem může mít také obchodní či marketingový význam.

Námi vytvořená aplikace bude ze dvou částí. V první řadě bude nutné vytvořit aplikační server, který bude poskytovat veřejné rozhraní (API). Pomocí tohoto rozhraní bude možné celou aplikaci ovládat. Druhou částí bude klientský kód pro webové prohlížeče, který se bude připojovat k serverové části. Serverová část bude vytvořena v NodeJS a klientská část bude vytvořena pomocí JavaScriptu a poběží v okně prohlížeče.

Aplikace bude umožňovat registrace a přihlášení uživatelů, bude tedy zapotřebí nějaké rozdělení těchto uživatelů a vytvoření a přiřazení uživatelských rolí. Není pro nás zásadní žádná administrátorská role s větším oprávněním apod., pravděpodobně budeme potřebovat pouze rozdělit přihlášeného a nepřihlášeného uživatele.

Přihlášený uživatel bude moci vytvořit libovolný počet aplikací a k nim vytvořit push notifikace. Aplikace budou tedy sloužit jako kontejnery pro jednotlivé notifikace. Uživatel bude pravděpodobně vytvářet aplikace dle aplikací nainstalovaných na jednotlivých zařízeních. Aby bylo možné rozesílat notifikace, je nutné umožnit uživateli vložit přístupové údaje k službám pro rozesílání push notifikací na dané platformy (Android, iOS, Windows Phone).

Z vytvořených a získaných dat budou vytvořeny základní přehledy a statistiky. Tyto statistiky bude možné filtrovat dle období. Dále bude služba umožňovat registraci zařízení a samozřejmě také odstranění přihlášeného zařízení.

Při vytváření notifikací bude možné určit datum a čas odeslání, čímž lze odeslat notifikaci ihned nebo naplánovat její odeslání do budoucna. Dále umožníme v rámci vytváření aplikací určit cyklus opakovaného odeslání notifikace. Notifikace se vždy odesílá na přihlášené zařízení, je tedy nutné u každé vytvořené notifikace zvolit cílová zařízení. Umožníme tedy zvolit cílová zařízení při vytváření aplikace — buď vybraná nebo všechna přihlášená.

Výsledek naší diplomové práce bude zveřejněn jako *open-source* na Githubu. Bude tak možné v budoucnu pokračovat ve vývoji i nad rámec diplomové práce, dále bude umožněno všem využívat naši službu. V neposlední řadě díky *open-source* mohou zájemci rozšířit či modifikovat naše zdrojové kódy pro své účely nebo mohou poskytnout nové funkcionality.

Kapitola 2

Vybraná konkurenční řešení

Kapitola se zabývá podrobným prozkoumáním webových aplikací, které poskytují stejné či podobné služby jako naše aplikace. Vytipovali jsme taková konkurenční řešení, která splňují alespoň část požadavků, které přímo plynou ze zadání (viz 1).

Tyto aplikace byly analyzovány z hlediska funkcionalit i uživatelského prostředí. Hlavním cílem analýzy jednotlivých konkurenčních řešení je udělat si celkovou představu o tématice z pohledu vnějšího světa, inspirovat se funkcionalitou, která by mohla být opomenuta, a následně převzít některé klíčové funkce. Zhodnocením webových aplikací jako takových lze také zjistit největší přednosti těchto konkurenčních řešení, které by bylo možné aplikovat na náš problém, a zároveň zásadní slabá místa, jichž bychom se při tvorbě naší aplikace měli vyvarovat.

Z hlediska uživatelského rozhraní lze zjistit, jak jsou informace prezentovány uživateli, na co konkurenti kladou největší důraz. Dále nás bude zajímat, jakým způsobem jsou jednotlivě poskytované funkcionality předkládány uživatelům a jak je složité je využívat. Výstupem této části analýzy je podklad pro návrh uživatelského rozhraní (viz 4.3).

Jelikož konkurenčních řešení je značné množství, vybrali jsme pro analýzu několik zástupců, na které se dále zaměříme. Jedná se o následující aplikace:

Pushwoosh - `https://www.pushwoosh.com/` [1]

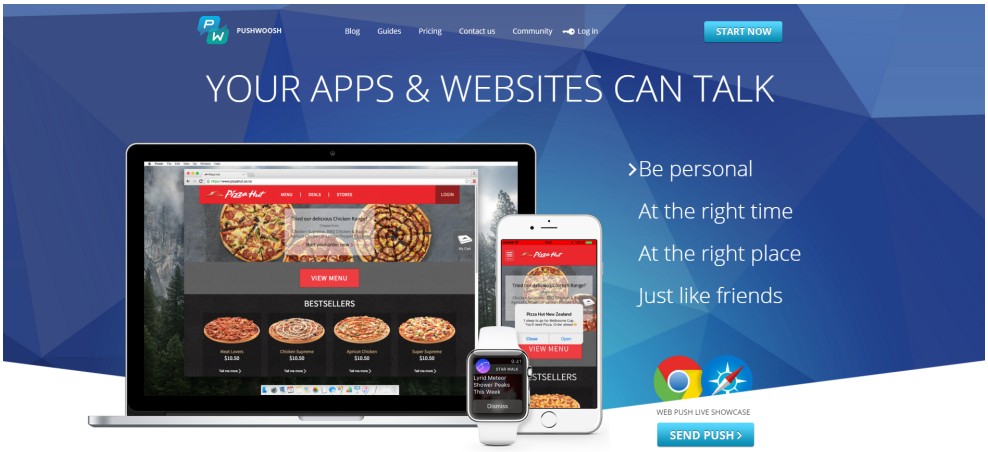
OneSignal - `https://www.onesignal.com/` [2]

Pushover - `https://pushover.net/` [3]

Carnival - `http://carnival.io/` [4]

2.1 Pushwoosh

Tato aplikace slouží pro všestranné rozesílání notifikací na nejrůznější platformy - celkem 21 různých platforem [1], na které je možné zasílat push no-



Obrázek 2.1: Pushwoosh - Ukázkový obrázek aplikace

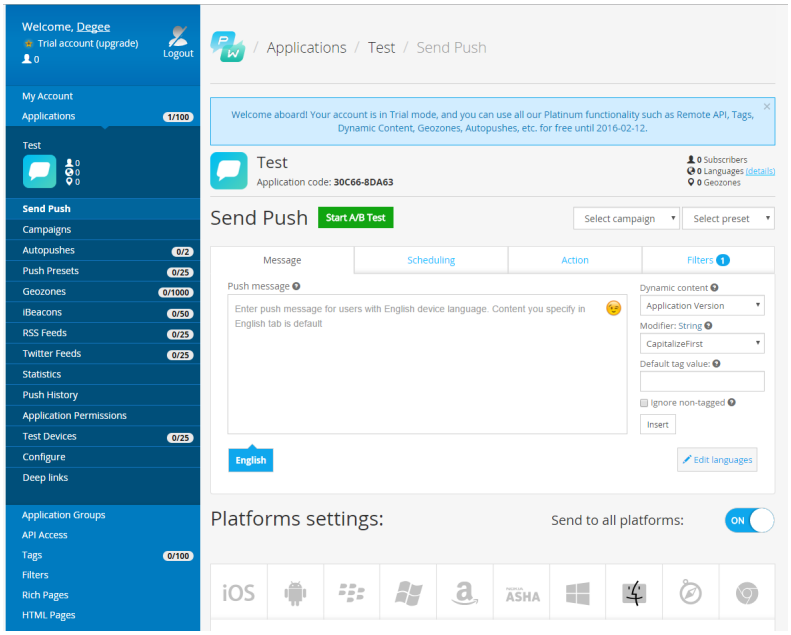
tifikace. Na obrázku 2.1 je zachycena úvodní obrazovka aplikace, na které je zřetelné tlačítko „SEND PUSH“. Pokud uživatel přijde z dané platformy na stránku a klikne na toto tlačítko, do jeho zařízení je zaslána push notifikace, což umožní uživateli okamžitě vyzkoušet nabízenou službu. Služba je dostupná zdarma pro registrované uživatele, ovšem pouze v základní verzi. Nabízí možnost zaplatit si různé typy uživatelských účtů, které zpřístupní nové funkce a rozšiřují velikost osobního prostoru, počtu aplikací apod. My se budeme zabývat pouze funkcemi zdarma.

2.1.1 Funkcionality

Hned na úvodní stránce [1] jsou také vypsány základní funkcionality, které aplikace zprostředkovává. Jedná se například o různé statistiky a analytické nástroje či plánovač push notifikací. Služba se chlubí hlubokou segmentací, osobním cloudem, vysokou rychlostí, zohledňováním časových pásem a zeměpisné polohy. Tyto informace lze získat bez jakékoliv registrace.

Abychom mohli prozkoumat podrobněji funkce, které služba poskytuje, je nutné se zaregistrovat. Vytvořili jsme si účet zdarma, který neobsahuje takové možnosti jako prémiový účet. Aplikace po zakoupení předplatného nabízí daleko více funkcí, než jsou dostupné z účtu zdarma, na čemž je založený její business model. Pro naše účely je neplacená verze postačující.

Po přihlášení se zobrazí seznam aplikací, kde lze námi vytvořené aplikace spravovat. Pro vytvoření aplikace zadáme její název, volitelně ikonu a výběr SDK pro dané platformy. Takto vytvořené aplikace lze třídit do skupin, jednotlivé skupiny lze taktéž spravovat. Kliknutím na námi vytvořenou aplikaci se nastaví jako vybraná, čímž je nám umožněno spravovat push notifikace a různá jejich nastavení, obecné nastavení aplikace včetně přidání testovacích



Obrázek 2.2: Pushwoosh - Ukázkový obrázek aplikace pushwoosh

zařízení, prohlížet statistiky, nastavovat oprávnění apod. Dále se budeme zabývat push notifikacemi.

Push notifikace vytvoříme zadáním textu, do kterého lze vložit přednastavený dynamický obsah (např. město, stát, jazyk). Dále je nutné zadat, kdy se má push notifikace odeslat — lze odeslat ihned nebo pomocí kalendáře zvolit datum a čas odeslání (zde je přínosem možnost zohlednit časové zóny cílového zařízení). Dále lze vybrat platformy, na které se bude push notifikace odesílat. Výbornou funkcí, kterou vytváření push notifikací nabízí, je typ akce — co se má stát po kliknutí na push notifikaci. Tato volba nám umožní například otevřít přidruženou aplikaci, URL v prohlížeči, konkrétní část aplikace apod. Poslední položka při vytváření push notifikace je filtr, pomocí kterého lze vyfiltrovat cílová zařízení — např. podle státu, ve kterém se nacházejí, podle stavu aplikace, verze aplikace, modelu cílového zařízení apod.

Mezi obecné funkcionality tohoto serveru patří správa aplikací, skupin, tagů, filtrů, nastavení API přístupu, zobrazení historie push notifikací a statistik. Statistika nám zobrazuje pouze kolik bylo v jednotlivých dnech odesláno notifikací a kolik z nich jich bylo otevřeno za posledních 30 dní.

2.1.2 Uživatelské prostředí

Z hlediska uživatelského rozhraní je stránka celkem přívětivá. Vlevo se nachází menu, které je stručně a přehledně rozdělené. Stránka bohužel není responzivní, takže na zařízeních s menší zobrazovací plochou je obtížné ji ovládat.

Výběrem aplikace se nám otevře celkem obsáhlé podmenu v hlavním menu, což není úplně intuitivní pro uživatele, kteří s aplikací začínají (viz obrázek 2.2). Obrovským mínusem uživatelského rozhraní je to, že se neukládá průběžně stav vytvářených push notifikací. Pokud chce člověk přiřadit filtr, který ještě nemá vytvořený, je mu nabídnut odkaz na vytvoření filtru, který bez jakéhokoliv upozornění zahodí doposud nastavené informace u nově vytvářených push notifikací a přesměruje uživatele do sekce pro správu filtrování. Dále se uživatelské rozhraní stává poněkud nepřehledným z důvodu využití tabů a množstvím různých tlačítek na jednotlivých podstránkách. Informace nejsou uživateli předkládány stručně a jasně.

2.1.3 Zhodnocení

Uživatelské rozhraní vzhledem k množství funkcí a informací prezentovaných uživateli je nepřehledné. Rozhodně bychom se tedy měli vyvarovat složitosti při návrhu GUI naší aplikace. Dále bychom také měli ukládat mezistavy vytvářených položek, případně alespoň zobrazit upozornění s potvrzením akce, aby uživatel nepřišel o doposud vyplněné údaje.

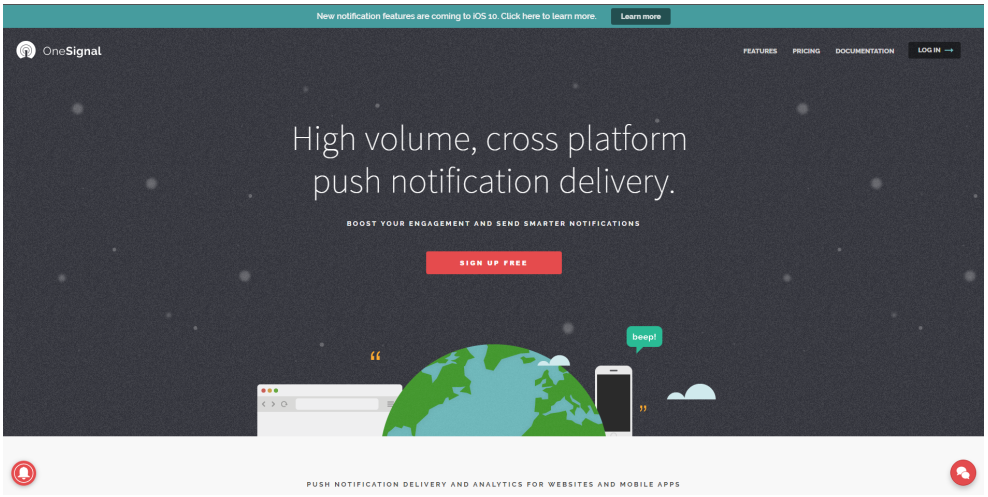
Tento server nabízí mnoho zajímavých funkcí, které se nám jeví jako užitečné i pro naší aplikaci a při návrhu aplikace je zohledníme. Jedná se o filtrování, zohlednění časových pásem cílových zařízení a typy akcí po kliknutí na notifikaci.

2.2 OneSignal

Webová služba OneSignal slouží pro multiplatformní doručování push notifikací. Nabízí také některé analytické nástroje pro webové stránky a mobilní aplikace. Na úvodní stránce je vypsáno hned několik poměrně velikých společností, které využívají této služby. Jedná se například o známou hudební televizní stanici MTV (`https://www.mtv.com/`), herní portál zynga (`https://zynga.com/`) či portál zábavy 9gag (`https://www.9gag.com/`). Landing page (obrázek 2.3) je vytvořena jako one page site (webová stránka, která má celý svůj obsah zobrazen pod sebou a odkazy vedou pouze na jednotlivé sekce na stránce pomocí scrollování) a sděluje uživateli, že je stoprocentně zdarma. Dále jsou zde vypsány hlavní funkcionality každého účtu zdarma (více v sekci 2.2.1).

2.2.1 Funkcionality

Aplikace na úvodní stránce uvádí své hlavní vlastnosti a funkcionality. Jedná se například o neomezený počet cílových zařízení, neomezený počet notifikací, možnost lokalizace, neomezené segmentování, plánovač doručení notifikací, realtime analytické nástroje, automatické doručování notifikací, možnosti importů a exportů dat. Dále aplikace nabízí plně využitelné API pro vzdálenou



Obrázek 2.3: OneSignal - Landing page webu

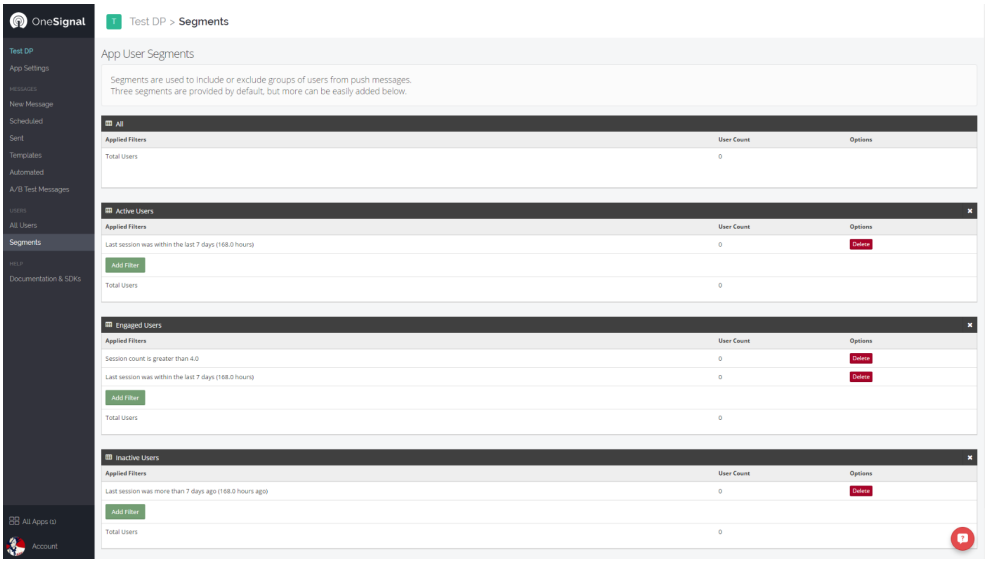
správu. Za účelem zvýšení konverzí poskytuje také A/B testování (vytvořeno více verzí, které jsou odeslány klientům a dle zpětné vazby se použije nejúspěšnější verze). Tyto poskytované funkce lze vyčíst bez nutnosti přihlášení do systému, nás však zajímá poněkud komplexnější rozbor aplikace, proto je nutné se přihlásit.

Pro přihlášení jsme využili možnosti přihlášení pomocí github účtu, dále stránka vedle klasické registrace umožňuje přihlášení pomocí facebookového účtu. Po přihlášení se zobrazilo plovoucí okno sloužící jako základní tutoriál a popis vlastností a funkcionalit. Zde jsme zjistili veškeré podporované platformy, dočetli jsme se o možnosti vytváření šablon pro notifikace, dozvěděli jsme se o rychlém automatickém odesílání zpráv, seznámili se podrobněji s možnostmi segmentace cílových zařízení (např. podle jazyku, aktivity uživatele a mnoho dalších kombinovatelných filtrů) a naučili se vytvořit první aplikaci. Po vytvoření první aplikace jsme byli vyzváni pro nakonfigurování alespoň jedné platformy z nabízených pro zasílání notifikací - tento krok jsme přeskočili.

Mimo již zmíněné funkce dále uživatelské rozhraní umožňuje vypsat seznam uživatelů, tedy cílových zařízení, které lze řadit a filtrovat. Tímto jsme shrnuli veškeré možnosti webové služby OneSignal.

2.2.2 Uživatelské prostředí

Tato webová služba se prezentuje jednoduchým, přehledným GUI (obrázek 2.4). Na první pohled je zřejmé, že při vytváření uživatelského rozhraní byl využit Twitter Bootstrap (nástroj pro tvorbu uživatelského rozhraní, jedná se o balík předdefinovaných kaskádových stylů s dokumentací, jak je správně využít) nebo podobný nástroj, což plyne z rozložení stránky, jejích ovláda-



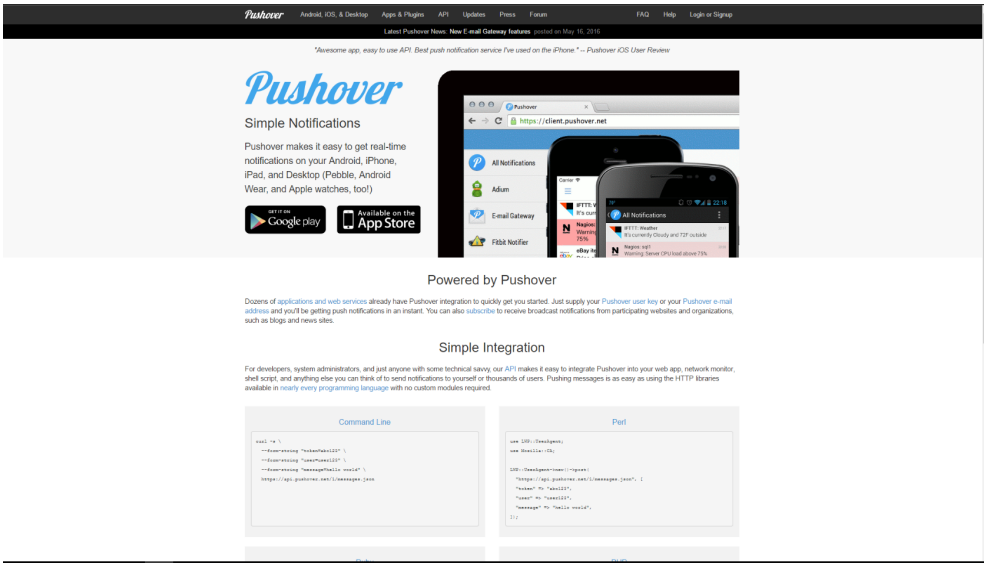
Obrázek 2.4: OneSignal - Administrátorské prostředí

cích prvků a jednotlivých bloků, ze kterých je stránka složená. Aplikace je rozdělena na několik fragmentů dle zobrazených informací, což ji činí přehlednou. Menu se nachází v levé části obrazovky, je přehledné a neobsahuje dále žádné submenu, proto je celkem intuitivní. V levém spodním rohu obrazovky jsou dvě tlačítka - pro správu aplikací a pro správu uživatelského účtu. Většina uživatelů je zvyklá podobná tlačítka hledat v pravém horním rohu, což by mohlo být poněkud matoucí, ale barevné oddělení těchto ovládacích prvků celkem dobře upoutá pozornost uživatele a na umístění tlačítek si rychle zvykne. U akcí s nevratným dopadem je použit potvrzovací dialog, což je dle základních principů tvorby GUI zcela správně. Stránka obsahuje drobečkovou navigaci, přestože by se mohla jevit jako zbytečná (nenašli jsme větší zanoření stránek než první úroveň). Drobečková navigace obecně není na škodu a je chytře schována do nadpisu dané podstránky. V menu je označena aktuální položka, která znázorňuje aktuální podstránku. Rozhraní také obsahuje prostor pro informační zprávy, jež se zobrazují v barevných boxech v horní části každé stránky.

2.2.3 Zhodnocení

Z hlediska uživatelského rozhraní nelze co vytknout. Aplikace je jednoduchá, přehledná a intuitivní pro koncového uživatele. Jednoznačně bychom měli při návrhu uživatelského rozhraní pro náš projekt zohlednit tyto aspekty.

OneSignal nabízí pouze hlavní funkcionality, ovšem pro naše účely by naprosto postačovaly. Při návrhu přihlédneme obecně k projektu OneSignal jako celku a budeme tvořit naší aplikaci ve stejném duchu.



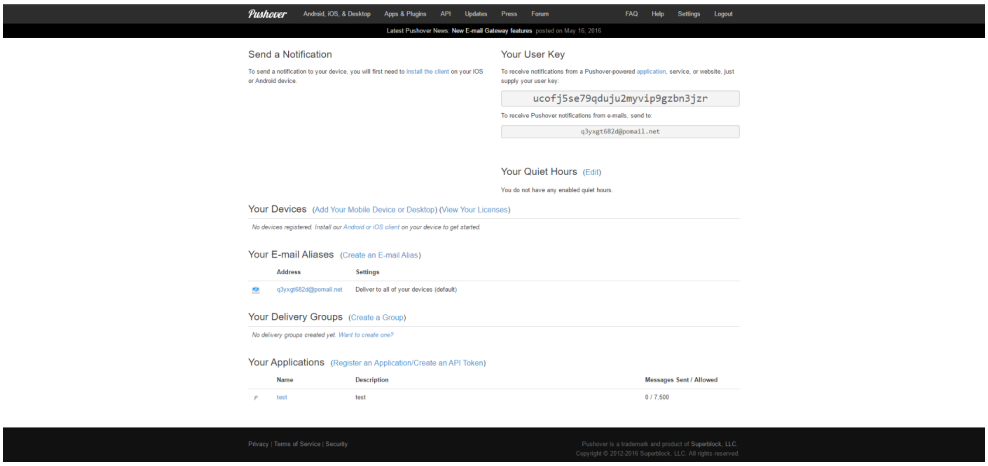
Obrázek 2.5: Pushover - Ukázka úvodní stránky aplikace

2.3 Pushover

Projekt Pushover na úvodní stránce (obrázek 2.5) slibuje jednoduché zasílání notifikací na platformy Android, iPhone, iPad a Desktop. Je zde také umístěn odkaz na obchody Google play a App store, kde lze nalézt mobilní aplikaci pro správu notifikací. Toto považujeme za značnou výhodu, jelikož mobilní aplikace umožňuje spravovat notifikace odkudkoliv jednoduchým způsobem pomocí přenosného zařízení. Podrobněji se však mobilní verzí nebudeme zabývat, zajímat nás bude hlavně webová část aplikace. Dalším zajímavým faktorem je, že pro správu notifikací poskytuje služba Pushover API s příklady použití v několika programovacích jazycích. Dále také upozorňuje, že lze vytvořit klienta pro správu notifikací v téměř jakémkoliv programovacím jazyce, stačí pouze podpora pro HTTP komunikaci. Dokumentace API je celkem obstojná, detailní a určitě bychom se měli při tvorbě našeho projektu dostat minimálně na takovou úroveň. Služba Pushover je placená, po týdenním používání zdarma na ukázku se uživatel může rozhodnout, zda si zakoupí jednorázovou licenci pro danou platformu. Pro naše analytické účely bude stačit tento zkušební účet, kde prozkoumáme webové rozhraní pro správu notifikací.

2.3.1 Funkcionality

Po detailnějším prozkoumání webu jsme zjistili, že služba je pouze jakýmsi zapouzdřením pro push notifikace. Aby bylo možné vůbec přijímat push notifikace v cílovém zařízení, je nutné mít nainstalovaného klienta, který zapouzdřuje veškeré notifikace. Znamená to tedy, že aplikace slouží pro sjednocení



Obrázek 2.6: Pushover - Otisk obrazovky uživatelské sekce

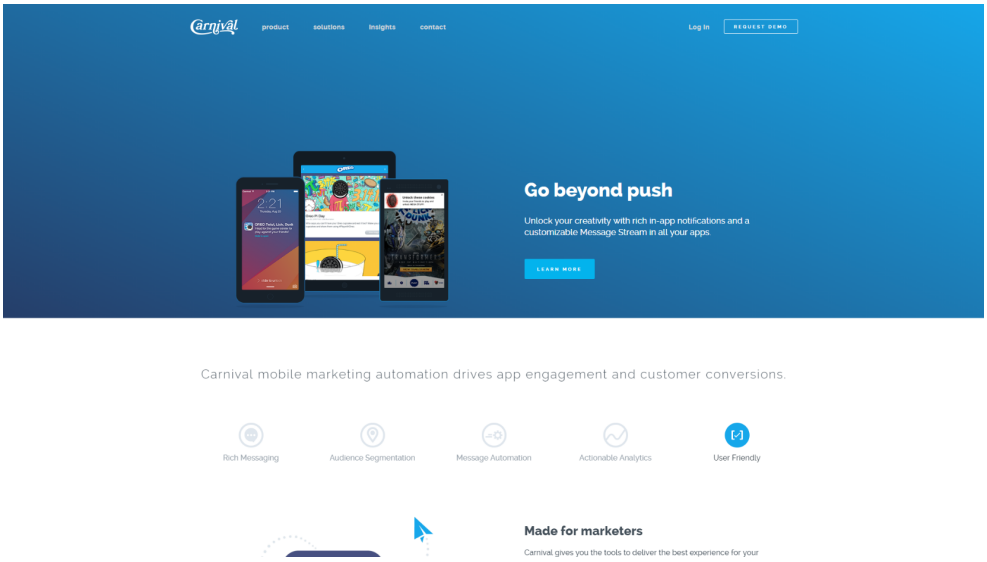
push notifikací na jedno místo, uživatel si může vybrat, jaké notifikace chce odebírat a od jakých projektů třetích stran. Tento koncept je zajímavý z hlediska uživatele, neboť má notifikace zobrazeny vždy pomocí klienta a může je jednoduchým způsobem spravovat, avšak není to přesně to, co od naší aplikace očekáváme. Nebudeme se tedy dále zabývat funkcionalitami tohoto webu.

2.3.2 Uživatelské prostředí

Uživatelské prostředí je responzivní, což umožňuje prohlížet stránky téměř ze všech zařízení. V horní části je jednoduché menu, které je vždy viditelné pro snadnou navigaci. Menu je logicky rozdělené intuitivně, uživatel ví na první pohled, pod jakým odkazem dohledá požadované informace. Některé stránky obsahují submenu, to je ovšem maximálně do druhé úrovně, třetí úroveň je již pouze pomocí odkazování se na danou sekci na stránce pomocí „kotvy“. Uživatelská sekce obsahuje příliš informací najednou, čímž se stává nepřehlednou (viz obrázek 2.6). Dále je celkem složité tuto sekci vůbec nalézt, neboť není v menu zobrazena (je dostupná na úvodní stránce webu pouze pro přihlášeného uživatele). Obecně se nám stránka opět jeví jako vytvořená pomocí technologie typu Twitter Bootstrap, což je patrně známka toho, že je tato technologie oblíbená, rozšířená a uživatelsky přívětivá.

2.3.3 Zhodnocení

Tento projekt nám nepřinesl nic nového z hlediska funkcionalit, protože je jeho koncept do značné míry odlišný. Myšlenka sjednocení notifikací pomocí jedné aplikace je sice zajímavá, ale není předmětem naší práce. Uživatelské rozhraní nás naopak utvrzuje v dojmu, že frontendový framework Twitter Bootstrap je rozšířený a využívají ho i větší projekty. Jedná se o již druhý analyzovaný pro-



Obrázek 2.7: Carnival - úvodní stránka

jekt, který má uživatelské prostředí realizovatelné pomocí Twitter Bootstrap. Jelikož je nám tato technologie známá, pravděpodobně přihlédneme k těmto faktorům při volbě nástroje pro tvorbu GUI.

2.4 Carnival

Služba Carnival se zaměřuje na automatizaci marketingových procesů pro mobilní zařízení. Nabízejí více než pouhé notifikace díky tzv. *rich in-app notifications*. Jedná se o bohaté možnosti typu obsahu push notifikací. Také nabízejí bohatou segmentaci cílových zařízení pro odeslání notifikace vždy správnému klientovi. V neposlední řadě můžeme využít různých statistik a analytických nástrojů. Do aplikace se nelze registrovat, lze pouze zažádat o demo účet a pravděpodobně se poté domluvit na konkrétním řešení s některým členem Carnival týmu. O demo účet jsme zažádali (pro vzdělávací účely), ale nedočkali jsme se žádné odezvy.

2.4.1 Funkcionality

Mezi hlavní funkcionality patří *Rich Messaging*, umožňuje zasílat pomocí push notifikací videa, obrázky, odkazy, text s emotikonami a podobně. Skládá se ze tří kanálů:

push notifikace umožní otevřít cílovou aplikaci

in-app notifikace notifikace přímo v aplikaci, slouží pro zvýšení konverzí

message stream doručuje obsah pro celou obrazovku (vyskakovací okno)

Uživatele lze rozdělit do různých segmentů založených na základě uživatelských preferencí, chování uživatelů, parametrů cílového zařízení či lokaci. Díky takovéto segmentaci lze obejít spamování uživatelů nevyžádanými zprávami a lze dané zprávy zacílit na konkrétní uživatele. Automatické odesílání předdefinovaných notifikací v předem určený čas umožňuje zastihnout uživatele ve správný moment. Každý uživatel může být zastižen dle následujících faktorů:

On-board nový uživatel během prvního týdne po stažení aplikace dostává užitečné in-app notifikace

Re-engage uživatel po určité době neaktivity dostává emotivní push notifikaci

Re-target založeno na předchozích akcích za účelem řízení konverzí

Geo-target zohlednění zeměpisné polohy uživatele, zacílení notifikací na základě umístění

Další funkcionalitou, kterou web Carnival nabízí, jsou analytické nástroje napojené na samotné odesílání notifikací, což klientovi umožňuje učit se na základě uživatelské aktivity a typu příspěvků, čímž lze docílit co největšího konverzního poměru.

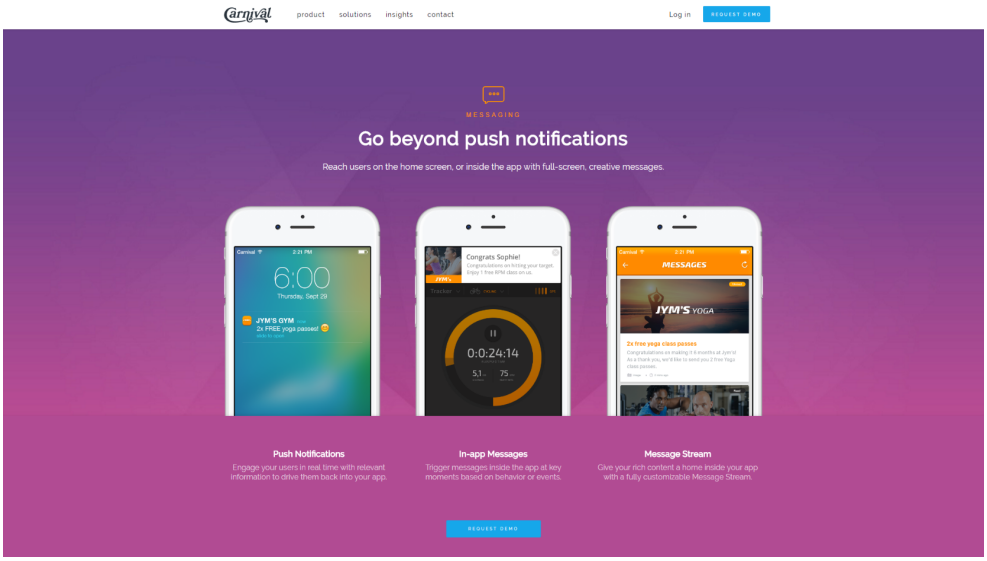
Služba Carnival z hlediska funkcionality nabízí vše, co konkurenční řešení, což z ní činí jednu z nejlépe využitelných aplikací pro správu a distribuci push notifikací.

2.4.2 Uživatelské prostředí

Z hlediska uživatelského rozhraní jsme produkt nemohli otestovat, protože nám nebyl udělen přístup do demo aplikace. Dočetli jsme se, že pro posílání multimediálního obsahu slouží drag-and-drop pole, což je uživatelsky velmi přívětivé, neboť pro přidání obrázku k notifikaci stačí pouze přetáhnout obrázek z lokálního úložistě do pole pro notifikace. Ohledně webové stránky produktu nemáme žádnou výtku, stránka je responzivní, moderní a přehledná.

2.4.3 Zhodnocení

Pokud bychom analýzu začali touto službou, ostatní by nám nepřinesly nic nového a my bychom se jimi zabývali zbytečně. Stránka se vyrovná všem vybraným konkurenčním řešením, avšak nic nového také neumožňuje (alespoň nic pro nás zajímavého). Z hlediska uživatelského rozhraní jsme aplikaci nemohli analyzovat, neboť nám nebyl udělen demo přístup.



Obrázek 2.8: Carnival - ukázka produktové podstránky

2.5 Shrnutí

Na základě analýzy konkurenčních řešení jsme identifikovali největší problémy a přednosti daných projektů. Právě díky této skutečnosti jsme schopni navrhnout náš systém co nejlépe tak, aby zohlednil veškeré nalezené přednosti a vyvaroval se záporů, na které jsme narazili.

Z hlediska uživatelského rozhraní se jedná zejména o využití ověřené technologie Twitter Bootstrap. Uživatelské prostředí by mělo být jednoduché a přehledné. Na stránce bychom neměli zobrazovat příliš mnoho informací, neboť s rostoucím počtem prezentovaných informací se snižuje přehlednost dané stránky. Menu by mělo být intuitivní, zanoření do více jak dvou úrovní není vhodné. Také by mělo být vždy viditelné pro snadnou navigaci na stránce. O výsledku všech akcí by měl být uživatel informován prostřednictvím boxů se zprávou na vždy viditelném místě. Každou akci, kterou nelze vzít zpět a má nějaký trvalý vliv na systém by mělo provázet potvrzení dané akce (např. „Opravdu chcete danou položku smazat?“). Toto jsou hlavní postřehy z hlediska GUI plynoucí z analýzy konkurenčních řešení.

Konkurenční řešení vynaložila daleko větší úsilí pro vytvoření daných služeb, ovšem v rámci diplomové práce není možné realizovat všechny funkcionality. Proto jsme se rozhodli pro pár základních, mezi které patří filtrované a řaditelné výpisy, filtrování cílových zařízení dle parametrů jako je například jazyk, lokace nebo aktivita uživatele. Dále umožníme vybrat typ akce, která se má provést po kliknutí na doručenou notifikaci. Vytvoříme základní plánovač odesílání a umožníme automatizované rozesílání notifikací. Zobrazíme také základní statistiky, které budou sloužit pro analytické účely.

Kapitola 3

Analýza

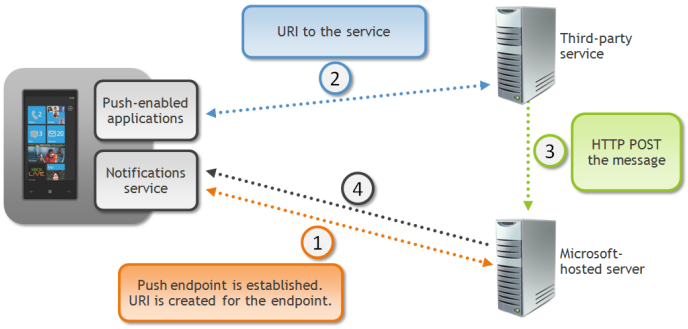
Kapitola se zabývá obecnou analýzou aplikace, která má být vytvořena v rámci této diplomové práce. Veškeré doposud zjištěné informace jsou zde rozřazeny do jednotlivých logických celků. Tato část obsahuje především grafické modely a jejich slovní popis. Jsou zde podrobněji popsány požadavky na aplikaci, vypsány veškeré požadované funkcionality výsledné aplikace a zachyceny entity, které se budou vyskytovat v naší aplikaci. V neposlední řadě kapitola analyzuje možnosti zasílání push notifikací na cílová zařízení. Tvoří podklad pro další práce, zejména pak pro sekci návrhu (4) a implementace (5).

3.1 Push notifikace

Jelikož je naším cílem zasílání push notifikací pro mobilní zařízení, je nutné nejprve analyzovat, jakým způsobem lze na dané platformy tyto notifikace zasílat. Pro naše účely se jedná o platformy Android, iOS a Windows Phone. V této části zjistíme potřebné informace pro zasílání push notifikací na dané platformy a prozkoumáme rozhraní pro vzdálenou komunikaci pomocí API. Zjistíme, jaká jsou omezení pro dané platformy. V neposlední řadě prověříme funkcionalitu, která je nám pro dané platformy poskytnuta. Výstupem této analýzy bude podklad pro návrh aplikace.

3.1.1 Windows Phone

Platforma Windows Phone umožňuje třetím stranám zasílat push notifikace prostřednictvím *Microsoft Push Notification Service*. Cílová aplikace pro doručení push notifikace musí být uzpůsobena pro přijímání. Aplikace si vytvoří *HttpNotificationChannel* objekt, který slouží pro komunikaci se službou třetí strany. Tomuto objektu je předána URI dané služby, která je obohacena o unikátní identifikátor zařízení, na kterém je aplikace nainstalována. Po spuštění aplikace se zařízení ohlásí pomocí URI službě, jež má za úkol spravovat push



Obrázek 3.1: Znázornění procesu zasílání push notifikací — Windows Phone
Zdroj: `https://i-msdn.sec.s-msft.com/dynimg/IC505830.png`

notifikace pro danou aplikaci. Tato služba je běžně vytvářena přímo vývojářem dané aplikace. [5]

Služba pro správu push notifikací shromažďuje přihlášená zařízení. V případě, že má být zaslána push notifikace na cílové zařízení, je vytvořen požadavek obsahující speciálně sestavený XML obsah. Takto vytvořený požadavek je odeslán na zprostředkovatele *Microsoft Push Notification Service*, který detekuje, zda je cílové zařízení připojeno k síti. Požadavky jsou ukládány a čekají na vyřízení do doby, než bude cílové zařízení dostupné, poté jsou odeslány. Z důvodu optimalizace využití přenosového pásma může *Microsoft Push Notification Service* notifikace zasílat na cílová zařízení v dávkách. Pro zařízení s Windows Phone může být zasíláno několik typů notifikací:

Toast notifikace Krátké, textové zprávy, které uživatele informují o novinkách či změnách. Skládají se z dvou textových polí a zobrazují se v horní části displeje. Po kliknutí je otevřena přidružená aplikace. Pokud je aplikace spuštěná v popředí, je zobrazení této notifikace předáno přímo do aplikace a nedojde k zobrazení v oznamovací oblasti.

Tile notifikace Dlaždicové notifikace, obsahují nadpis, číselnou hodnotu a obrázek pozadí. Mohou změnit vzhled dlaždic v oblasti *Quick Launch* neboli rychlého spouštění.

Raw notifikace Nemají předdefinovaný vzhled ani formát obsahu. Data obsažená v těle notifikace jsou zvolena odesílatelem a cílová aplikace musí těmto datům rozumět a být připravena je zpracovat. Z důvodu vlastního obsahu dat je pro doručení této notifikace nutné, aby byla aplikace na cílovém zařízení spuštěna, jinak je taková notifikace systémem ignorována (není zde nativní zpracování).

Z hlediska autentifikace lze využít dvou módů — ověřený a neověřený. Pro neověřený mód je zde omezení na 500 notifikací za den pro jeden kanál. Pro

ověřené služby zde není omezení, ale je nutné zaregistrovat certifikát podepsaný certifikační autoritou *Microsoft-trusted root certificate authority*. Tento certifikát pak zabezpečuje SSL připojení mezi službou třetí strany a *Microsoft Push Notification Service*.

3.1.2 Android a iOS

Vedle možnosti vlastní implementace push notifikací pomocí dalších technologií, mezi které patří např. periodické dotazování na server (pooling, long pooling) nebo vytvoření persistentního připojení pomocí protokolu soketů, existuje služba, která dokáže zasílat na cílová zařízení notifikace, kdykoliv mají být zaslány. Tato služba se nazývá *Firebase Cloud Messaging* (dále FCM) a dokáže emitovat událost přímo do cílového zařízení, takže nedochází k žádné další režii při komunikaci. FCM je multiplatformní, umožňuje zasílat notifikace na zařízení s iOS i Android. Služba FCM je následníkem GCM (*Google Cloud Messaging*), GCM je označeno za deprecated a postupně by veškeré služby měly přejít na využívání FCM (v dokumentaci lze nalézt migrační manuál). Z těchto důvodů se budeme v této části zabývat právě touto službou. [6]

Pro zpracování push notifikace na cílovém zařízení musí být aplikace uzpůsobena, na stránkách s dokumentací je popsáno, jakým způsobem lze aplikaci modifikovat tak, aby byla schopna zpracovávat notifikace. V rámci této diplomové práce ovšem řešíme pouze serverovou část zapouzdřující vytváření notifikací a dále tedy budeme předpokládat, že pokud někdo bude chtít využít tento projekt, je obeznámen s modifikací aplikace pro přijímání notifikací. Pro kompletní návod modifikace aplikace navštivte pro Android – `https://firebase.google.com/docs/cloud-messaging/android/client` a pro iOS – `https://firebase.google.com/docs/cloud-messaging/ios/client`. Jelikož se cílové zařízení musí nahlásit našemu serveru jako aktivní, bude pravděpodobně nutné v aplikaci provést nějaké úpravy, zejména pak přidat URI pro přihlášení zařízení k odběru notifikací.

Firebase Cloud messaging umožňuje zasílat až 4KB uživatelských dat v rámci zprávy, lze zaslat notifikační zprávu nebo datovou zprávu. Notifikační zpráva je zobrazena uživateli a je informativního charakteru. Datová zpráva vyžaduje implementaci zpracování v cílové aplikaci a může měnit obsah aplikace. Na API FCM se zasílá zpráva ve formátu JSON, která obsahuje klíč „to“ na top-level úrovni. Tento klíč rozhoduje o cílovém zařízení. Dalším klíčem na stejné úrovni je typ zrpávy s obsahem dané zprávy.

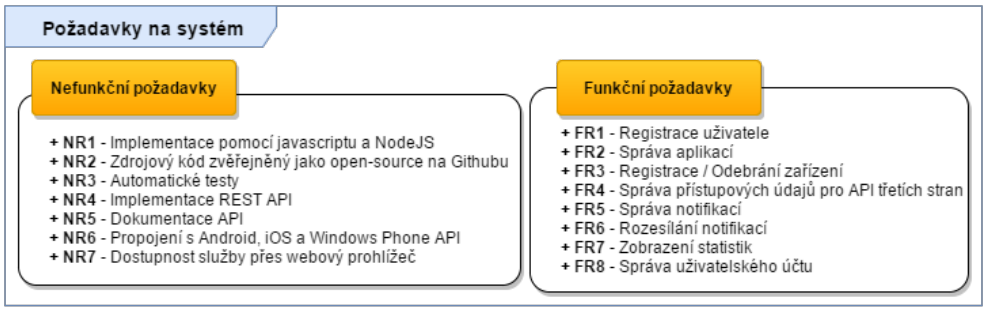
Notifikační zpráva Klíč „notification“, obsahuje tělo tělo, titulek a ikonu notifikace. Notifikace je zobrazena v oznamovací oblasti zařízení.

Datová zpráva Klíč „data“, obsahuje libovolná data do maximální velikosti 4KB. Je očekáváno, že cílová aplikace je schopná zpracovat tato data.

Obrázek 3.2: Znázornění principu FCM komunikace
Zdroj: `https://firebase.google.com/docs/cloud-messaging/images/messaging-overview.png`

Kombinace notifikace a dat Slouží k rozdílnému chování, pokud je aplikace spuštěna v popředí či v pozadí. Pokud je aplikace v pozadí, zařízení zobrazí notifikaci a data zpracuje pouze při kliknutí na notifikaci. Pokud je v popředí, aplikace obdrží objekt s notifikací i daty.

Notifikacím lze dále nastavit prioritu na normální či vysokou. Normální priorita nebudí uspané zařízení (čímž šetří baterii), vysoká priorita naopak probudí uspané zařízení a otevře internetové spojení (zejména instant messaging a chat). Notifikaci lze dále určit, zda může být nahrazena či nikoliv. Tento příznak slouží k tomu, aby novější zprávy mohli nahradit staré a uživateli se tak nezobrazovali nadále i neaktuální informační sdělení. Zprávám může být také nastavena životnost, po kterou se je pokouší FCM doručit — cílové zařízení není připojeno k internetu, je vypnuto apod. Tato funkcionalita může být použita například pro časově omezené pozvánky nebo kalendářní události. Zprávy mohou být zasílány na konkrétní cílové zařízení, skupině zařízení nebo zařízením, které mají přihlášený odběr daného tématu.



Obrázek 3.3: Funkční a nefunkční požadavky na systém

Jak to funguje? Proces je zachycen na obrázku 3.2, pojďme si jej tedy přiblížit. Implementace zahrnuje aplikační server třetí strany, která komunikuje s FCM pomocí HTTP nebo XMPP protokolu. FCM odesílá push notifikace na cílová zařízení. Pro vytváření a správu notifikací lze využít aplikační server nebo notifikační konzoli, kterou služba FCM nabízí.

3.2 Funkční a nefunkční požadavky

V této sekci jsou vypsány veškeré funkční a nefunkční požadavky na systém. Vycházíme ze sekce 1.2, avšak zabýváme se již konkrétně jednotlivými požadavky z analytického hlediska. Zároveň si vymezujeme jednotlivé požadavky na základě doposud zjištěných informací — některé mohou být přidány na základě analýzy konkurenčních řešení (kapitola 2), jiné naopak zavrženy z důvodu komplikované realizace přesahující rámec této diplomové práce. Tato sekce následně slouží jako podklad pro analýzu případů užití.

3.2.1 Nefunkční požadavky

NR1 – Implementace pomocí javascriptu a NodeJS
Serverová část bude vytvořena technologií NodeJS. Bude vytvořen samostatný server, který vyřizuje HTTP požadavky. Dále bude vytvořen klient implementovaný v javascriptu (konkrétně React).

NR2 – Zdrojový kód zveřejněn jako open-source na Githubu
Výsledný zdrojový kód bude vystaven pomocí verzovacího systému Git ve službě Github jako open-source. Každý bude moci kód volně využívat či modifikovat.

NR3 – Automatické testy
Serverová část bude pokryta automatickými testy. Jedná se o jádro celé aplikace, tudíž musí být řádně otestováno a plně funkční.

NR4 – Implementace REST API
Serverová část bude poskytovat rozhraní pro vzdálené ovládání aplikace. Bude tedy vytvořeno REST API pro komunikaci klient-server pomocí HTTP požadavků.

NR5 – Dokumentace API
Aby mohla být aplikace využívána systémy třetích stran pomocí REST API, musí být toto rozhraní řádně zdokumentováno.

NR6 – Propojení s Android, iOS a Windows Phone API
Za účelem zasílání push notifikací na cílová zařízení bude nutně aplikace propojena se službami poskytujícími tuto funkcionalitu.

NR7 – Dostupnost služby přes webový prohlížeč
Pro uživatelsky přívětivé ovládání aplikace bude vytvořen klient dostupný přes webové rozhraní. Klient bude komunikovat se serverem pomocí REST API.

3.2.2 Funkční požadavky

Funkční požadavky popisují požadavky na funkcionality systému, které bude naše aplikace umožňovat. Tyto funkční požadavky jsou podrobně popsány a následně využity při analýze případů užití 3.3.

FR1 – Registrace uživatele
Aplikace bude nepřístupná pro běžného uživatele. Příchozímu uživateli bude umožněno zaregistrovat se pomocí emailu a hesla. Zaregistrováním bude vytvořen účet, pomocí kterého se nadále může uživatel přihlásit a zpřístupnit si veškeré funkcionality aplikace. S registrací uživatele dále souvisí funkce zapomenutého heslo, pokud tedy uživatel zapomene heslo, může zadat svůj email, na který mu bude zasláno nově vygenerované heslo. Dále musí být umožněno přihlášenému uživateli odhlásit se z aplikace.

FR2 – Správa aplikací
Přihlášenému uživateli bude umožněno vytvářet, upravovat a mazat své aplikace. Aplikace budou vypsány v klientské části a slouží k odkazu na rozhraní se správou notifikací a registrovaných zařízení. Principem aplikací je rozdělení registrace zařízení do nějakých kontejnerů, aplikace by primárně měly být přidružené reálným aplikacím na cílových zařízeních, ale mohou být využity jako pouhé zaobalení skupiny zařízení. Dále bude aplikaci přiřazena jednoznačná URI pro registraci zařízení k odběru notifikací. Aplikaci lze deaktivovat, čímž dojde k potlačení odesílání push notifikací na cílová zařízení.

FR3 – Registrace/ Odebrání zařízení
Systém musí poskytovat zařízením funkci přihlášení k odběru notifikací. Obecně musí být do mobilní aplikace vložena URI, na které se mají cílová zařízení registrovat. Administrátorské rozhraní pro tento funkční požadavek nemá smysl, neboť identifikátor cílového zařízení zná pouze API umožňující zasílat push notifikace na cílové zařízení a zařízení samo. Přihlašovat ručně zařízení k odběru notifikací tedy není možné, dalo by se ovšem vytvořit administrátorské prostředí pro odebírání zařízení, přijde nám to však zbytečné a programátoři mobilních aplikací pravděpodobně pomocí nějakého ovládacího prvku v nastavení aplikace umožní zařízení registrovat a odhlašovat z odběru notifikací pomocí REST API.

FR4 – Správa přístupových údajů pro API třetích stran
Každý uživatelský účet bude mít přidruženou entitu s nastavením, kde provede veškerá nastavení pro vzdálené přístupy k API pro zasílání push notifikací. Pomocí těchto přístupů mohou být zasílány push notifikace na přihlášená cílová zařízení. Uživatel bude moci poprvé vyplnit údaje, později je pouze modifikovat. Systém poté bude implementovat komunikaci se vzdálenými API dle takto vyplněných přístupových údajů. Uživateli tedy stačí nadefinovat přístupy a vytvořit notifikace, které budou automaticky odeslány na cílová zařízení.

FR5 – Správa notifikací
Uživatel bude moci vytvářet notifikace pro jednotlivé aplikace. Každá notifikace bude svázána s právě jednou aplikací. Lze vybrat cílová zařízení, na která má být zaslána push notifikace, buď jednotlivě nebo všem přihlášeným. Notifikaci bude možné upravit nebo smazat. Odesílání notifikací bude možné naplánovat dopředu nebo je poslat ihned. Dále bude možné určovat cyklus opakování rozeslání notifikace — číslo vyjadřující počet dní mezi jednotlivými rozesílky od první odeslané notifikace. Čas odeslání zůstane vždy stejný i pro opakované rozesílky, pokud nebude modifikován uživatelem.

FR6 – Rozesílání notifikací
Notifikace je možné po vytvoření odeslat ihned nebo naplánovat rozesílku. Dále lze nastavovat pravidelný interval počtem dní, po kterém se notifikace budou opakovaně zasílat na cílová zařízení. Cílová zařízení lze vybrat libovolně z registrovaných zařízení aplikace.

FR7 – Zobrazení statistik
Přihlášený uživatel bude mít k dispozici základní přehledy zaslaných notifikací a registrovaných zařízení. Tyto statistiky lze pouze prohlížet. Slouží k informativním a analytickým účelům, zejména za účelem zjištění odezvy uživatelů a zvýšení konverzí.

Obrázek 3.4: Účastníci a bloky případů užití v přehledu

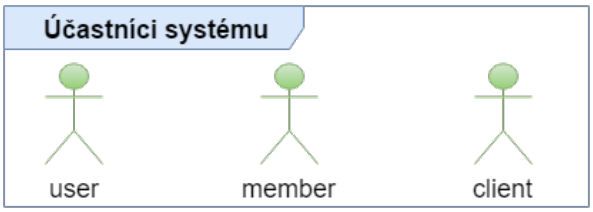
FR8 – Správa uživatelského účtu
Přihlášený uživatel má k dispozici rozhraní pro změnu hesla a aktivace/deaktivace účtu. Neaktivní účet nezasílá žádné notifikace, jedním kliknutím tedy lze pozastavit zasílání notifikací na všechna cílová zařízení přihlášená k odběru push notifikací ve všech aplikacích. Změna hesla je zde hlavně z důvodu funkcionality zapomenutého hesla — pokud uživateli přijde vygenerované heslo, pravděpodobně si jej bude chtít změnit na nějaké, které si bude lépe pamatovat.

3.3 Model případů užití

Případy užití charakterizují určité použití systému daným uživatelem. Nejprve si tedy definujeme účastníky, kteří se podílejí na případech užití (obrázek 3.4). Poté pomocí takto definovaných uživatelů vymodelujeme funkcionality systému tak, aby byly pokryty veškeré případy užití. Tato sekce slouží mimo jiné k detailní specifikaci požadavků na systém, zároveň tvoří podklady pro akceptační testy.

3.3.1 Účastníci

Celý systém má být přístupný až po přihlášení, je tedy nutné mít uživatelskou roli přihlášeného uživatele (*member*). Aby se uživatel mohl stát přihlášeným uživatelem, musí mít přístup k základním úkonům jako je registrace či přihlášení. Proto musíme definovat ještě obecnou roli, kterou má jakýkoliv návštěvník systému (*user*). Dále je nutné definovat abstraktní roli, která nám umožní zastoupit klientská zařízení pro zasílání push notifikací (*client*). Tato role reprezentuje cílová zařízení pro všechny platformy, slouží pouze pro registraci



Obrázek 3.5: Obrázek zachycuje účastníky systému — uživatelské role

cílového zařízení v naší aplikaci. Takto definované role budou pro naše účely postačující. Jednotlivé role zachycuje obrázek 3.5.

user každý návštěvník stránky, nemá žádné oprávnění kromě registrace a přihlášení

member rozšiřuje roli *user*, je oprávněný provádět veškeré úkony v aplikaci

3.3.2 Případy užití

Tato sekce obsahuje detailní specifikaci jednotlivých funkcionalit systému pomocí modelů případů užití. Jednotlivé případy užití jsou vždy svázány s nějakým účastníkem, který je oprávněn využít danou funkcionalitu. Jednotlivé případy užití jsou roztříděny do logických bloků a kompletní přehled případů užití lze nalézt na obrázku 3.6.

Registrace uživatele
Blok obsahuje základní případy užití pro všechny uživatele. Umožňuje vytvořit nového uživatele a tímto otevřít přístup k dalším případům užití. Grafické znázornění — obrázek 3.7.

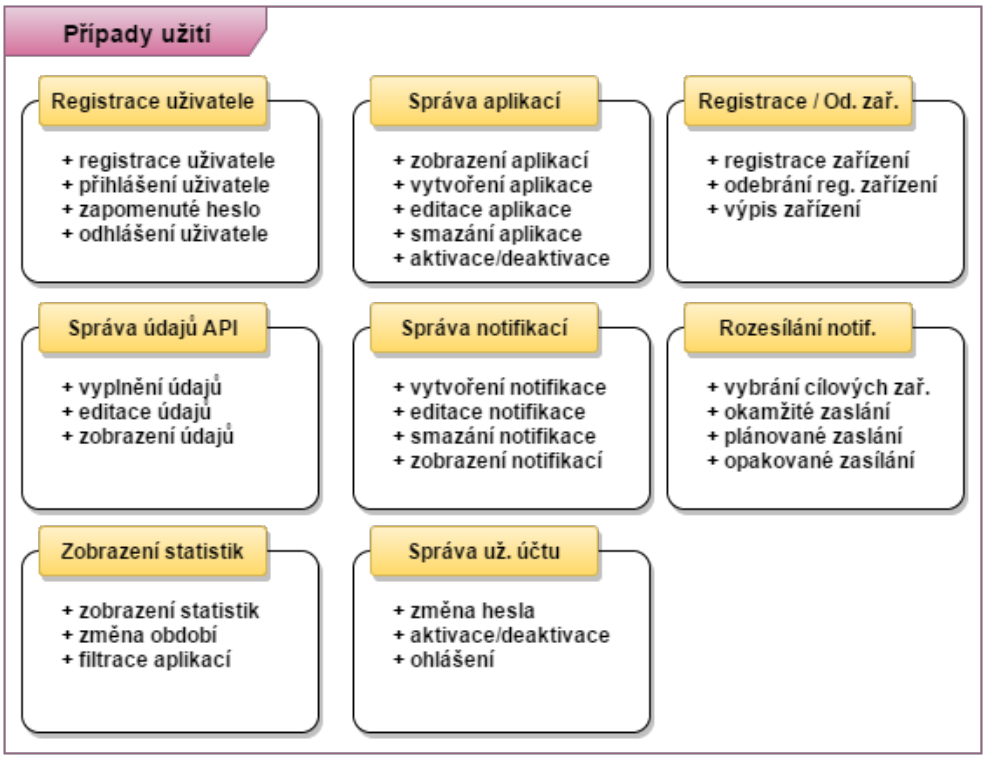
Registrace uživatele
Umožňuje vytvořit nového uživatele pro následné přihlášení.

Přihlášení uživatele
Umožňuje přihlásit uživatele a tím změnit uživatelskou roli z *user* na *member*.

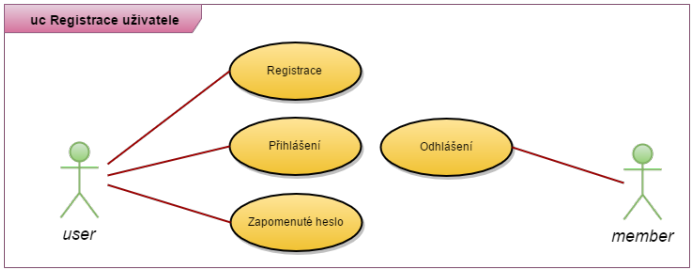
Zapomenuté heslo
Umožňuje zaslat uživateli na email nově vygenerované heslo pomocí zadání emailové adresy.

Odhlášení uživatele
Umožní přihlášenému uživateli odhlásit se ze systému.

Správa aplikací
Přihlášený uživatel má přístup pouze k údajům, které sám vytvořil. Blok obsahuje funkcionality spojené se správou aplikací a tvoří základ pro další případy užití — obrázek 3.8.



Obrázek 3.6: Seznam případů užití dle logických bloků



Obrázek 3.7: Logický blok případů užití - Registrace uživatele

Zobrazení aplikací
Umožňuje uživateli zobrazit výpis vytvořených aplikací.

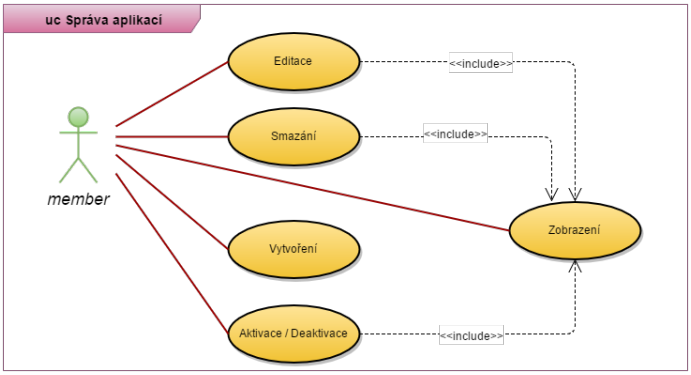
Vytvoření aplikace
Umožňuje vytvořit novou aplikaci. Takto vytvořená aplikace se automaticky zobrazí ve výpisu aplikací.

Editace aplikace
Umožňuje upravovat dříve vytvořené aplikace. Pro úpravu aplikace je nutné kliknout na tlačítko pro editaci ve výpisu aplikací — viz

include vazba znázorněná na obrázku 3.8.

Smazání aplikace
Umožní uživateli smazat dříve vytvořenou aplikaci. Pro smazání opět slouží tlačítko pro smazání ve výpisu aplikací — viz *include* vazba znázorněná na obrázku 3.8.

Aktivace / Deaktivace
Umožňuje rychlým způsobem aktivovat nebo deaktivovat aplikaci. Neaktivní aplikace nezasílá push notifikace na cílová zařízení. Ve výpisu aplikací lze změnit příznak aktivní pomocí přepínače — viz *include* vazba znázorněná na obrázku 3.8.



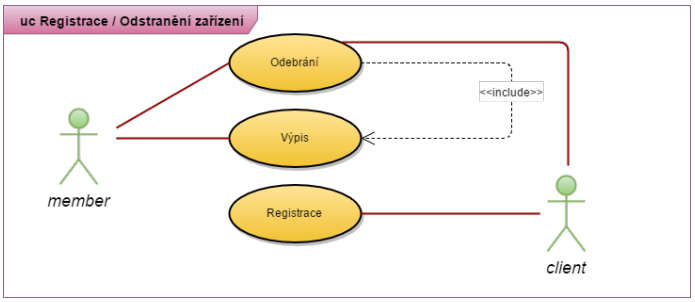
Obrázek 3.8: Logický blok případů užití - Správa aplikací

Registrace / odhlášení zařízení
Jedná se o jediný případ užití, kde se vyskytuje role *client*. Tato role zastupuje klientské zařízení, aplikaci, která se chce přihlásit k odběru push notifikací. Systém bude shromažďovat takto přihlášená zařízení a přihlášený uživatel poté bude moci vypsat a odebírat zařízení přihlášená k odběru — viz obrázek 3.9.

Registrace zařízení
Registrovat zařízení může pouze role *client*. Zařízení se registruje pomocí unikátního identifikátoru, který slouží pro komunikaci s API třetích stran pro rozesílání push notifikací. Je tedy zřejmé, že nelze zařízení přidávat ručně, neboť neznáme unikátní identifikátor. V rámci aplikace bude k dispozici URI, na které se bude moci cílové klientské zařízení přihlásit k odběru. Tato URI bude sdělena uživateli, který musí nějakým způsobem danou adresu předat aplikaci na cílovém zařízení — nejčastěji vložit přímo do zdrojového kódu. V rámci této práce bude mít vzorová aplikace vstupní pole pro zadání dané *subscribe* URI.

Výpis zařízení
Umožňuje vypsat cílová zařízení, která se přihlásila k odběru. Výpis bude obsahovat tlačítko pro zrušení odběru notifikací pro dané zařízení.

Odebrání registrovaného zařízení
Umožňuje odebrat cílové zařízení z přihlášených zařízení. Uživatel pomocí výpisu zařízení může zrušit odběr notifikací daného zařízení. *Client* může opět vyvoláním URI zrušit odběr notifikací na daném zařízení. Tato URI by měla být stejná, jako pro přihlášení, pouze bude obohacena o unikátní identifikátor zařízení a využije jinou metodu HTTP protokolu.



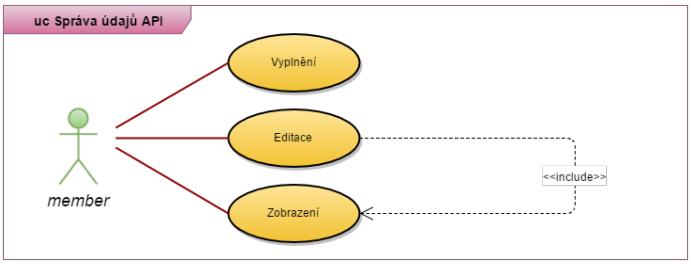
Obrázek 3.9: Logický blok případů užití - Registrace a odebrání zařízení

Správa údajů API
Pro zaslání push notifikace na klientské zařízení je nutné využít API třetích stran. Z tohoto důvodu existuje tento logický blok případů užití. Je nutné uložit přístupové údaje, abychom mohli s těmito API komunikovat. Jednotlivé případy užití zachycuje obrázek 3.10.

Vyplnění údajů
Umožňuje prvotní vyplnění přístupových údajů pro API třetích stran. Tento případ užití bude k dispozici pouze uživatelům, kteří doposud nevyplnili přístupové údaje.

Editace údajů
Umožňuje změnit přístupové údaje k API třetích stran. Doposud vytvořené položky zůstanou po změně nastavení stejné, pouze se změní přístupové údaje u daných záznamů (např. aplikace).

Zobrazení údajů
Umožňuje zobrazit aktuálně vyplněné přístupové údaje k API pro zasílání push notifikací. Citlivé údaje budou vidět až po potvrzení zobrazení údajů — např. tajný klíč k API bude zobrazen po kliknutí na tlačítko „zobrazit tajný klíč“.



Obrázek 3.10: Logický blok případů užití - Správa údajů API

Správa notifikací

Tento blok případů užití je zásadním pro splnění cílů této práce. Jedná se o kompletní správu push notifikací. Tyto případy užití jsou k dispozici po vytvoření nějaké aplikace — viz Správa aplikací, obrázek 3.8. Uživatel bude mít přístup striktně k vlastním notifikacím, tedy těm, které sám vytvořil. Takto předem vytvořené notifikace lze následně rozesílat. Případy užití pro správu notifikací zachycuje obrázek 3.11.

Vytvoření notifikace

Umožňuje vytvořit novou notifikaci. Tato notifikace bude vždy svázaná s dříve vytvořenou aplikací.

Editace notifikace

Umožňuje upravit údaje dříve vytvořené notifikace. Uživateli je umožněno měnit veškeré údaje notifikace. Pro editaci slouží tlačítko ve výpisu notifikací — viz *include* vazba znázorněná na obrázku 3.11.

Smazání notifikace

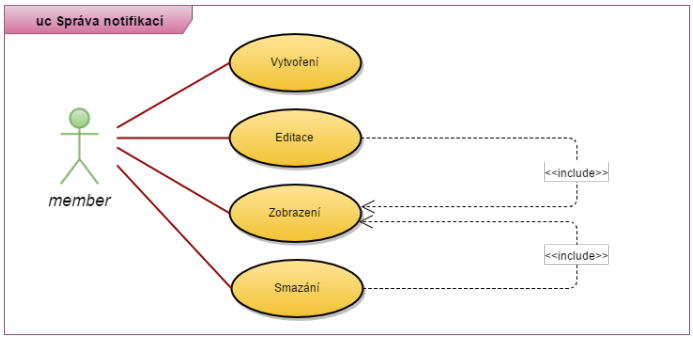
Umožňuje uživateli smazat dříve vytvořené notifikace. Smazání je trvalé, pro smazání je nutné potvrdit akci v potvrzovacím dialogu. Smazat lze pomocí tlačítka ve výpisu notifikací — viz *include* vazba znázorněná na obrázku 3.11.

Zobrazení notifikací

Umožňuje zobrazit výpis vytvořených notifikací. Umožňuje přístup k dalším případům užití (editace, mazání notifikací).

Rozesílání notifikací

Tyto případy užití úzce souvisejí s blokem správy notifikací (3.11). Jedná se zejména o nastavení rozesílek notifikací, z důvodu přehlednosti byly tyto dva logické bloky případů užití rozděleny. Ovšem pokud bychom bloky spojili, rozesílání notifikací by mělo obsahovat vazbu na správu notifikací, neboť rozšiřuje dříve zmíněné případy užití. Případy užití bloku rozesílání notifikací jsou zobrazeny na obrázku 3.12.



Obrázek 3.11: Logický blok případů užití - Správa notifikací

Vybrání cílových zařízení
Umožňuje přiřadit cílová zařízení dříve vytvořené notifikaci. Lze vybrat jednotlivá zařízení nebo veškerá přihlášená k odběru notifikací dané aplikace.

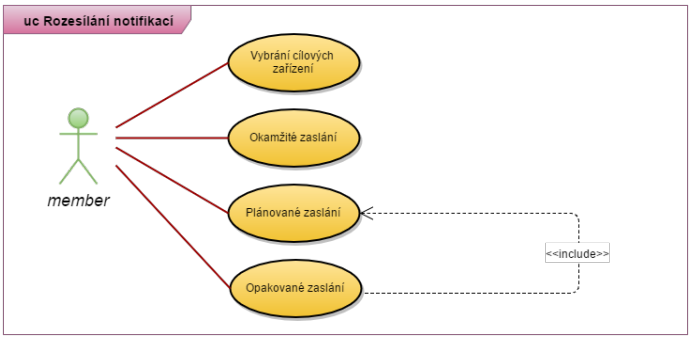
Okamžité zaslání
Umožňuje okamžitě odeslat předem definovanou notifikaci. Jedná se o jednorázové manuální zaslání notifikace kliknutím na tlačítko ve výpisu notifikací.

Plánované zaslání
Umožňuje uživateli nadefinovat datum a čas, kdy má být notifikace automaticky odeslána. Systém poté dle nadefinovaných záznamů automaticky komunikuje s API třetích stran a zasílá notifikace na cílová zařízení.

Opakované zaslání
Umožňuje uživateli nastavit notifikacím cyklus opakovaní zasílání. Uživatel nastaví počet dní, po kolika se má zasílání opakovat, a datum a čas prvního odeslání. Poté je notifikace automaticky opakovaně posílána na cílová zařízení po x dnech vždy ve stejný čas jako první odeslání. Počet dnů pro opakované zaslání notifikací je obsažen ve formuláři případu užití *Plánované zaslání* — vazba *include* na obrázku 3.12.

Zobrazení statistik
Uživateli je umožněno zobrazovat statistiky odeslaných notifikací s možností základního filtrování. Filtrování bude obsaženo při zobrazení statistik — viz obrázek 3.13.

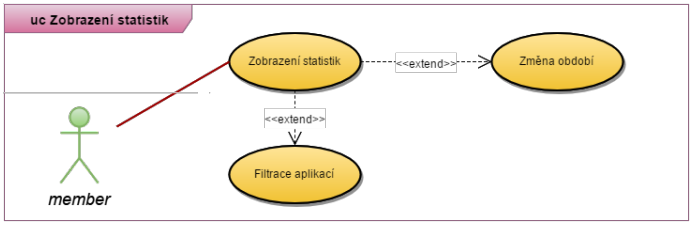
Zobrazení statistik
Umožňuje uživateli zobrazit přehledy a statistiky odeslaných notifikací za určité období.



Obrázek 3.12: Logický blok případů užití - Rozesílání notifikací

Změna období
Umožňuje uživateli změnit období pro zobrazení statistik — rozšiřuje případ užití *Zobrazení statistik*.

Filtrování dle aplikací
Umožňuje filtrovat zobrazení statistik dle vytvořených aplikací — rozšiřuje případ užití *Zobrazení statistik*.



Obrázek 3.13: Logický blok případů užití - Zobrazení statistik

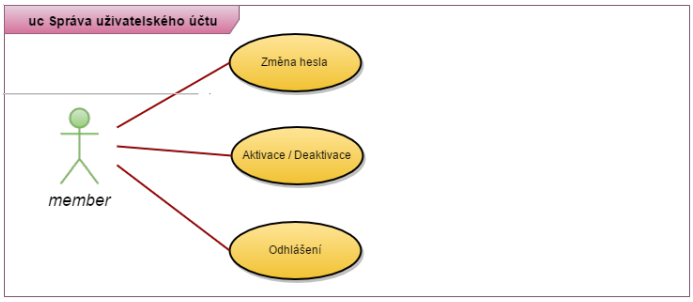
Správa uživatelského účtu
Umožňuje spravovat základní uživatelské údaje — viz obrázek 3.14.

Změna hesla
Umožňuje uživateli změnit přístupové heslo k systému. Uživatel zadá staré heslo a poté dvakrát nové heslo. Pokud se staré heslo shoduje s původním heslem a zároveň se obě nová hesla shodují, bude heslo změněno.

Aktivace / Deaktivace
Umožňuje uživateli jediným kliknutím aktivovat nebo deaktivovat uživatelský účet. Deaktivovaný účet nezasílá žádné notifikace, cílovým zařízením je však umožněno se nadále přihlašovat k odběru notifikací.

Odhlášení

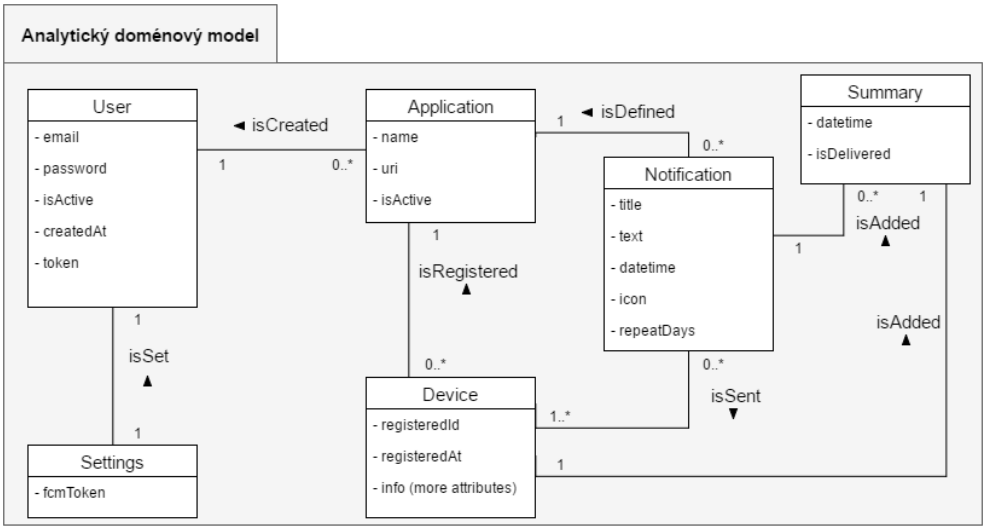
Umožňuje odhlášení uživatele ze systému — stejný případ užití jako u bloku *Registrace uživatele*.



Obrázek 3.14: Logický blok případů užití - Správa uživatelského účtu

3.4 Doménový model

Analytický doménový model (obrázek 3.15) patří do skupiny diagramů struktur a slouží pro základní popis dat. Obsahuje entity (třídy) vyskytující se v aplikaci a zachycuje jejich atributy a vazby mezi nimi. Tento diagram je základem pro vytvoření datového modelu a následně i samotné aplikace.



Obrázek 3.15: Analytický doménový model aplikace

3.4.1 Entity modelu

User

Reprezentuje každého registrovaného uživatele systému.

Atributy	Popis
`email`	Email uživatele. Slouží jako přihlašovací jméno.
`password`	Heslo uživatele. Uchováno jako otisk hashovací funkce.
`isActive`	Příznak, zda je účet aktivní. Pokud je neaktivní, nezasílají se notifikace.
`createdAt`	Datum registrace uživatele.
`token`	Autorizační token, generován po přihlášení, umožňuje přístup k zabezpečeným *endpointům* API

Application

Aplikace vytvořené uživatelem. Slouží jako kontejnery pro přihlašovaná zařízení.

Atributy	Popis
`name`	Název aplikace. Slouží jako identifikátor pro uživatele.
`uri`	URI, na které se registrují zařízení k odběru notifikací této aplikace.
`isActive`	Příznak, zda je aplikace aktivní. Pokud je neaktivní, nezasílají se notifikace.

Device

Zařízení, která se registrovala k odběru notifikací pro danou aplikaci.

Atributy	Popis
`registeredId`	Identifikátor zařízení, dle kterého je zaslána notifikace na dané zařízení pomocí API dané platformy.
`registeredAt`	Datum registrace zařízení k odběru notifikací aplikace.
`info`	Obecný atribut pro uchování informací o zařízení (platforma, model apod.). Pravděpodobně bude při návrhu rozložen na více atributů.

Notification

Notifikace vytvořené uživatelem. Každá notifikace náleží aplikaci. Může být vybráno jedno nebo více cílových zařízení pro zaslání notifikace.

Atributy	Popis
`title`	Titulek notifikace.
`text`	Text notifikace.
`datetime`	Datum a čas, kdy má být notifikace zaslána.
`icon`	Ikona notifikace. Zobrazena u notifikace na cílovém zařízení.
`repeatDays`	Cyklus opakování ve dnech. Lze nastavit libovolné opakování notifikace po „repeatDays“ dnech.

Summary

Statistika, přehled zaslaných notifikací. Záznam vždy náleží notifikaci a obsahuje právě jedno zařízení.

Atributy	Popis
`datetime`	Datum zaslání notifikace.
`isDelivered`	Příznak, zda byla notifikace doručena na cílové zařízení.

Kapitola 4

Návrh

4.1 Návrh systému

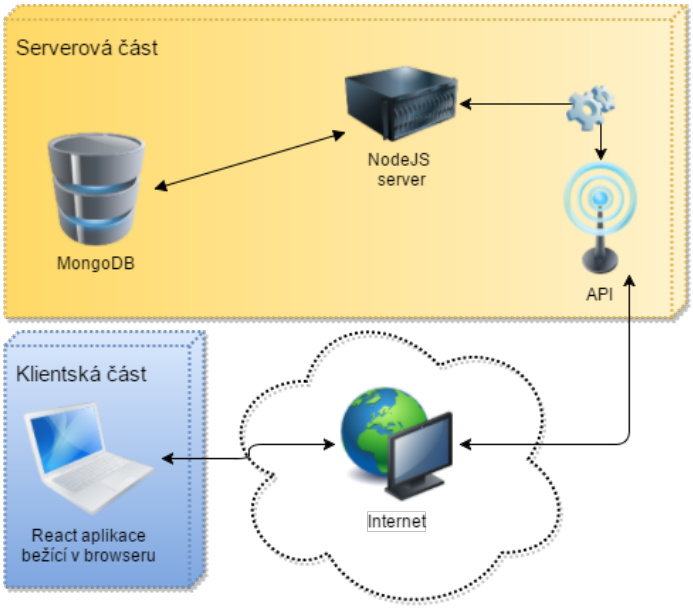
Systém se bude skládat z několika hlavních celků. Logicky bude rozdělen na dvě základní části – serverovou a klientskou část. Klientská část bude komunikovat prostřednictvím internetu se serverovou částí. Bude implementována v javascriptové knihovně react a interpretována v browseru koncového uživatele. Serverová část bude zapouzdřená a pro komunikaci s klientem bude k dispozici API. Serverová část se skládá z databázového stroje MongoDB, dále ze serveru běžícím na technologii NodeJS a API pro vzdálenou komunikaci se serverem.

4.2 Volba použitých technologií

Jak je patrné ze zadání 1, je nutné vytvořit server pro správu push notifikací s webovým klientem, dále vytvořit API pro vzdálené spravování, které bude zprostředkovávat ovládání serveru z klientské webové aplikace. Z těchto důvodů je nutné zvolit technologie pro vytvoření API serveru a GUI využívající tuto API. Oddělíme tedy serverovou část od klientské části. Naší aplikaci tedy zjednodušíme na samostatný server, který zprostředkovává API. Odděleně vytvoříme webovou klientskou část pro přehledné zobrazení informací (GUI). Klientská část bude komunikovat se serverem právě pomocí API. Další nutností je persistentní uchování dat, proto se při volbě technologií musíme zabývat i datovým úložištěm.

4.2.1 Server

Pro serverovou část se hned nabízí několik technologií, nad nimiž jsme uvažovali. Při volbě technologií jsme dbali na přehlednost výsledného řešení, vlastnosti dané technologie pro nás užitečné, naší znalost dané technologie a zda je technologie vhodná pro vytvoření API serveru.



Obrázek 4.1: Diagram systému

Python je pro nás neznámý jazyk, máme s ním minimální zkušenosti na úrovni přepisování algoritmů z tohoto jazyka do jiného, nám bližšího. Tento jazyk tedy není nejvhodnější pro vytvoření API serveru, jelikož by nám neznalost jazyka ztěžovala práci..

Jazyk PHP je obecně vhodný pro webové aplikace, ovšem pokud bychom nevyužili některý z frameworků tohoto jazyka (např. Symfony, Zend, Nette, Laravel), vývoj by byl dost komplikovaný. Jelikož chceme mít čistý kód zbavený veškerých přebytečných knihoven a funkcí, které frameworky nabízejí, rozhodli jsme se PHP nevyužít.

Stále více využíván je jazyk Ruby. Samotný jazyk Ruby není nejvhodnější pro tvorbu API serveru či webových aplikací. Bylo by opět nutné využít nějaký framework (např. Sinatra, Ruby on Rails) či další knihovny. Pokud bychom ale využili framework Ruby on Rails, většinu práce by za nás udělal rails generátor, který dokáže pomocí příkazů zadaných do konzole generovat REST API. Tento framework je naprosto vyhovující pro naše účely, přesto jsme se však rozhodli jej nezvolit.

Zbývající technologií, která připadá v úvahu, je NodeJS. Jelikož je vývoj webového API serveru v NodeJS celkem zajímavá záležitost, zejména díky správci balíčku NPM, který umožňuje instalovat jednoduchým způsobem různá rozšíření a knihovny, rozhodli jsme se využít právě tuto technologii. Budeme zajisté potřebovat nějakou knihovnu, která již obsahuje předpřipravený webový server (např. node-restify, express.js). Dále budeme používat další knihovny pro usnadnění vývoje, tyto knihovny lze nalézt v sekci *Knihovny,*

rozšíření a další technologie (viz 4.2.5).

4.2.2 Klient

S předpokladem, že máme dostupný server s API, můžeme volit technologie pro vytvoření klientské části. Zde můžeme využít téměř jakýkoliv jazyk podporující tvorbu GUI a získávání dat ze vzdáleného endpointu. Je pro nás důležité, aby nebylo nutné pro vytvoření GUI webového prohlížeče využívat další servery a jiné technologie. Z tohoto důvodu jsme se rozhodli rovnou zavrhnout jazyky jako PHP, Ruby apod. pro tvorbu prezentační části naší aplikace.

Jelikož webové prohlížeče dnešní doby nativně podporují javascript, rozhodli jsme se využít pro tvorbu klientské části javascriptovou knihovnu React. Tuto technologii vytvořil a také využívá Facebook. Budeme využívat také další javascriptové knihovny, o kterých si povíme více v sekci 4.2.5.

4.2.3 Databáze

NodeJS a jeho knihovny podporují nepřeberné množství persistentních datových úložišť. Chtěli bychom vytvořit aplikaci schopnou přepnout ovladač dané databázové platformy a tím využívat libovolný databázový stroj. Pro nás to znamená, že bychom měli psát databázové dotazy co nejjednodušším způsobem, pravděpodobně vytvořit nějaký interface pro komunikaci s databází – bohužel interface v javascriptu zatím není implementován. Každý, kdo bude později využívat takto vytvořený kód, si tedy může naimplementovat komunikaci s libovolným databázovým strojem.

NodeJS umožňuje spolupracovat například s MySQL, MongoDB, PostgreSQL, Redis, ElasticSearch, SQLite a dalšími databázovými stroji. My jsme se rozhodli pro implementaci využít databázi MongoDB, protože pomocí knihovny *mongoose* můžeme definovat přímo v aplikaci objekty, které jsou převedené do databáze. Tímto způsobem máme rovnou vytvořené entity, které můžeme používat v databázi a zároveň databázové schéma. Vyhneme se tedy složitému vytváření databáze a můžeme pracovat přímo s objekty, které se samy namapují na databázové tabulky. Pro migrace a budoucí použití to tedy znamená, že stačí mít spuštěný databázový server MongoDB, aplikace si sama vytvoří potřebné tabulky v databázi a je připravená pro používání z hlediska databáze. Dalším důležitým faktorem je, že pokud provedeme nějakou změnu v objektu entity, tato změna se automaticky projeví v databázi, aniž bychom museli řešit migrace apod.

4.2.4 Základní technologie

Node.js [7] webový API server

Node.js je javascriptový runtime, který umožňuje vykonávat javascriptový kód pomocí příkazového řádku. Využívá událostmi řízený, neblokující vstupně/výstupní model, který tuto technologii činí jednoduchou a

efektivní. Pro správu doplňků a knihoven využívá správce balíčků *npm*. Balíčky *npm* tvoří největší ekosystém open-source knihoven na světě [7].

React [8] klientská část aplikace
React je javascriptová knihovna, která umožňuje vytvářet uživatelské prostředí. Uživatelské prostředí se skládá z jednotlivých komponent, které mohou být závislé na stavu aplikace. Jednotlivé komponenty potom reagují na změny stavů aplikace. React umožňuje aktualizovat a překreslit pouze komponenty, kterých se daná změna stavu týká, což je jeden z důvodů, proč je využití této technologie efektivní. Každá react komponenta má svůj vnitřní stav, který sama spravuje. Výsledné uživatelské prostředí se potom skládá z většího množství námi definovaných komponent a tvoří celek. Vytvářením jednotlivých komponent zaručujeme jejich znovupoužitelnost a zapouzdřujeme jejich funkcionalitu pro tvorbu komplexního uživatelského prostředí.

MongoDB [9] persistentní úložiště dat
MongoDB je open-source databázový stroj. Jeho databáze je dokumentová — každý záznam v databázi je dokument, což je datová struktura podobná JSON objektům. Dokument je tvořen dvojicí záznamů *pole: hodnota*, kde hodnotou může být jiný dokument (dokumenty lze vnořovat do sebe). Jednotlivé dokumenty jsou uloženy v kolekcích. Kolekce jsou analogicky totožné s tabulkami v objektově relačních databázích. Každý dokument musí mít unikátní identifikátor *_id*, který slouží jako primární klíč. MongoDB poskytuje vysoký výkon, dostupnost a automatické škálování [9]. Usnadňuje vývoj tím, že odstraňuje potřebu ORM (objektově relačního mapování) jednotlivých záznamů.

4.2.5 Knihovny, rozšíření a další technologie

Tato sekce obsahuje krátký popis jednotlivých technologií, které jsme využili pro tvorbu naší aplikace. Základem pro vytvoření klientské i serverové části je *react-redux-universal-example* [10]. Jedná se o soubor knihoven sestavený ve vzorový projekt, kde klient je realizován pomocí reactu a server pomocí NodeJs API. Pro naše účely tedy naprosto vyhovující.

Redux universal example využívá zejména tyto technologie:

Isomorphic (Universal) rendering javascriptový kód je spustitelný na klientské i serverové části

React javascriptová knihovna pro tvorbu uživatelského rozhraní

React Router knihovna sloužící pro synchronizaci uživatelského prostředí a URL adresy, umožňuje navigaci uživatelským prostředím na základě změn URL adresy prohlížeče

Express jedná se o flexibilní Node.js framework pro tvorbu webových aplikací

Babel transformuje kód javascriptu nové generace (např. ES2015) do původního javascriptu, který je podporován cílovými zařízeními

Webpack slouží pro sestavování výsledného kódu, spravuje závislosti implementovaného kódu, obsahuje širokou škálu nastavení

Webpack Dev Middleware slouží pro automatické kompilování zdrojů a zdrojových kódu ihned při změně sledovaných souborů

Webpack Hot Middleware umožňuje automatickou obnovu zdrojových souborů za nově zkompilované, takže není nutné restartovat server či obnovovat okno prohlížeče pro aplikování nového kódu

Redux kontainer stavů pro javascriptové aplikace, pomáhá tvořit aplikace, které se chovají konzistentně, jsou spouštěny v různých prostředích a jsou snadno testovatelné [11]

React Router Redux propojení React/Redux routování

ESLint udržuje konzistentní styl kódu, obsahuje pravidlo pro zápis javascriptového kódu (např. odsazení, použití středníků, uvozovek a mnoho dalších)

redux-form propojuje formuláře se stavem aplikace, spravuje formuláře v Redux stavu

style-loader, sass-loader, less-loader umožňuje pracovat s kaskádovými styly v čistém css, a preprocesorech sass a less

bootstrap-sass-loader, font-awesome-webpack dovoluje přizpůsobovat balík stylů a fontů Bootstrap a FontAwesome

mocha umožňuje vytvářet a spouštět unit testy

Redux universal example samozřejmě neobsahuje všechny potřebné knihovny a doplňky, které jsme se rozhodli použít pro realizaci našeho systému. Mezi další technologie, které jsme se rozhodli využít patří *Chai* (url: `http://chaijs.com/`) a *chai-http*. Jedná se o knihovny poskytující pravidla pro tvorbu automatických testů, lze také v rámci testů zasílat reálné požadavky, čímž lze otestovat např. API pomocí reálných dotazů.

Pro uchování uživatelského hesla bylo nutné použít hashovací funkci, proto jsme se rozhodli pro balíček *bcryptjs*, který umožňuje využívat v javascriptu bcrypt hashovací funkci bez dalších závislostí. Bcrypt hashování používáme nejen na hesla, ale také na přístupové *tokeny*, které jsou v každém autorizovaném požadavku předávány v hlavičce (Authorization) HTTP požadavku.

Pro práci s MongoDB databází v Node.js je obvyklé použít knihovnu *mongoose*. Tato knihovna umožňuje komunikovat s MongoDB databází a vytváří nám databázovou kompletní vrstvu pro práci s MongoDB databází. [9]

Z důvodu funkce zapomenutého hesla je nutné z našeho serveru zasílat obnovené heslo na email uživatele. Pro tuto funkcionalitu jsme se rozhodli využít knihovnu *Nodemailer*. Tato knihovna umožňuje zasílat emaily pomocí mailových serverů. Pro naše účely jsme vytvořili gmail účet, jehož přístupové údaje jsme poskytli nodemaileru. Pro zaslání emailu z naší aplikace tedy pošleme požadavek na gmail a ten odešle na cílový email zprávu. [12]

Abychom mohli kontrolovat, zda se uživatelem vytvořené notifikace mají odeslat, potřebujeme nějaký periodicky se opakující skript, který bude automaticky odesílat notifikace. Zvolili jsme knihovnu *Node Cron*. Jedná se o minimalistický plánovač úloh pro Node.js využívající syntaxi stejnou jako linux *crontab*. Více na `https://github.com/merencia/node-cron`.

Pro práci s datem a časem v javascriptu jsme se rozhodli použít knihovnu *moment*. Jendá se o javascriptovou knihovnu poskytující množství funkcí pro práci s datem a časem, mezi které patří zejména různorodé formátování, modifikace datumů a časů, parsování vstupů apod. [13]

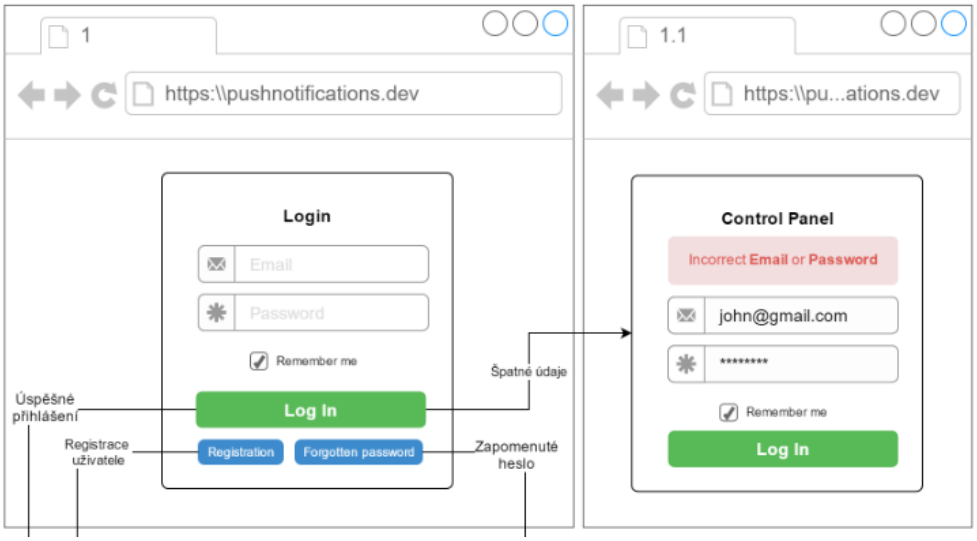
4.3 Návrh UI

Návrh uživatelského rozhraní je z důvodu vytvoření webové aplikace důležitou součástí této práce. Vycházeli jsme z analytické části této práce, konkrétně ze sekce 2.5. Vytvořili jsme si *wireframe*, tedy drátěný model, který nám usnadní vytvoření klienta pro push notifikační službu. Tento drátěný model zachycuje některé základní stavy aplikace, jako například přihlášený/nepřihlášený uživatel, proces přihlášení uživatele, výpisy jednotlivých stránek apod. Drátěný model neobsahuje formuláře pro vytváření nových položek, neboť v době tvorby *wireframu* ještě nebyl vytvořen databázový model — nebylo jednoznačně určené, které pole se zobrazí v daném formuláři. Také nám tyto formuláře nepřipadaly jako zásadní pro návrh uživatelského rozhraní. Tento model mimo svou hlavní funkci slouží jako podklad pro určení základních *endpointů* API. Díky rozložení stránek můžeme při návrhu API vytipovat, jaká data budou potřeba na jednotlivých stránkách.

Jelikož je kompletní drátěný model celkem rozsáhlý, je jeho kompletní zpracování uvedeno v přílohách této práce. Na obrázku 4.2 je pro ukázku znázorněna část drátěného modelu, jedná se o vstupní stránku aplikace, jsou zde zachyceny přechody mezi jednotlivými stranami. Dále si popíšeme jednotlivé části kompletního drátěného modelu:

Vstupní stránka aplikace (obrázek B.1)

Tato část zachycuje jednotlivé pohledy pro nepřihlášeného uživatele. Nalezneme zde přihlašovací formulář (záložka 1), ze kterého se můžeme dostat při úspěšném přihlášení do aplikace. Při neúspěchu se zobrazí



Obrázek 4.2: Ukázkový obrázek drátěného modelu

hlášení s popisem chyby (záložka 1.1), jedná se o totožnou stránku jako s přihlašovacím formulářem, ovšem doplněnou o popis chyby. Navíc nám zde přetrvají námi zadané údaje.

Při kliknutí na tlačítko *Registration* se zobrazí formulář pro registraci (záložka 1.2). Formulář obsahuje email nového uživatele a dva vstupy pro vyplnění hesla. Vyplněním vstupního pole pro heslo a následným vyplněním druhého vstupního pole pro heslo totožnými daty dochází k eliminaci chyb jako jsou například překlepy či různá rozložení klávesnice. Obě hesla se musí shodovat.

Při kliknutí na tlačítko *Forgotten password* se zobrazí formulář pro zadání uživatelského emailu (záložka 1.3). Na základě takto vyplněného vstupního pole a následném odeslání dojde k vygenerování nového hesla uživatelského účtu pod danou emailovou adresou, které bude zasláno na uživatelský email. V případě, že pod daným emailem nebude uživatelský účet nalezen, dojde k zobrazení chybové hlášky na stejné stránce.

Nástěnka přihlášeného uživatele (obrázek B.2)

Po úspěšném přihlášení uživatele se zobrazí stránka obsahující přehledy a statistiky daného uživatele. Jednotlivé statistiky a přehledy budou zapouzdřeny v boxem, bude se jednat o pouhé výpisy dat pravděpodobně s odkazy na detaily, pokud daný odkaz bude mít smysl. Budou zde informace jako naposledy registrovaná zařízení, naposledy odeslané notifikace, přehled aplikací uživatele, grafové souhrny odeslaných notifikací apod.

Na obrázku lze vidět rozložení stránky, které je pro přihlášeného uživatele pro všechny ostatní stránky stejné. V horní části se nachází navigační lišta, která obsahuje zleva menu a vpravo email přihlášeného uživatele s podnabídkou pro odhlášení a změnu nastavení uživatelského účtu. V menu bude vždy zvýrazněna aktivní položka podle aktuální stránky. Následuje nadpis aktuální stránky, vedle něhož se nachází u stránek první úrovně popis dané stránky a u stránek zanořených drobečková navigace. Poté je nadpis oddělen horizontální čarou a následuje obsah dané stránky. Za obsahem stránky bude další horizontální oddělení a následně patička s textovými informacemi.

Výpis aplikací (obrázek B.3)
Výpis uživatelem vytvořených aplikací slouží primárně jako přehled, avšak také jako rozcestník pro další podstránky. Nalezneme zde tlačítko pro vytvoření nové aplikace, které povede na stránku s formulářem pro vložení aplikace. Ve výpisu nalezneme také *Subscribe URI*, která slouží pro přihlášení odběru notifikací aplikace pro dané zařízení. Tato URI bude součástí cílové aplikace dané platformy, nejprve je tedy nutné vytvořit aplikaci, poté je možné zakomponovat přidělenou URI k cílové aplikaci pro obdržení notifikací. Dále ve výpisu budou u každého záznamu ovládací prvky, které ovšem na obrázku nejsou vidět. Jedná se zejména o přepínač pro příznak aktivní/neaktivní, dále tlačítko pro smazání, které bude obsahovat potvrzovací dialog. Kliknutím na název aplikace dojde k jejímu vybrání a tím se zobrazí stránka B.4.

Vybraná aplikace (obrázek B.4)
Výběrem aplikace z výpisu aplikací se dostaneme na tuto stránku. Vedle nadpisu stránky si můžeme všimnout drobečkové navigace. Stránka obsahuje přehled přihlášených zařízení, přehled vytvořených notifikací a přehled kanálů dané aplikace. Kanály jsou uvedeny pouze jako ilustrativní záležitost, v rámci aplikace je nadále nebudeme reflektovat. Každý výpis bude mimo základních údajů také obsahovat ovládací prvky, kterými bude možné záznamy upravovat, mazat apod. Na stránce budou také tlačítka pro vytvoření nových záznamů do jednotlivých přehledů. Dále můžeme kliknutím na daný přehled zobrazit detail přehledu, čímž se dostaneme na jednu z následujících stránek — B.4, B.5 a B.6.

Vybraný kanál (obrázek B.5)
Jedná se o filtr výpisu notifikací a přihlášených zařízení na základě vybrané aplikace a kanálu. Jak již bylo řečeno, funkcionalita kanálů nebude realizována, tudíž nemá smysl tento filtrovaný rozcestník dále popisovat. Jeho funkci převezme pohled na obrázku B.4.

Výpis notifikací dle aplikace (obrázek B.6)
Jedná se o výpis notifikací filtrovaný dle zvolené aplikace. Nalezneme zde

podrobnější informace o notifikacích včetně ovládacích prvků pro editaci, mazání apod. Také zde bude tlačítko pro vytvoření nové notifikace.

Výpis zařízení dle aplikace (obrázek B.7)
Jedná se o výpis zařízení přihlášených k odběru notifikací filtrovaný dle zvolené aplikace. Nalezneme zde podrobnější informace o zařízeních, ovšem bez jakýchkoliv ovládacích prvků. Přihlašování a odhlašovaní totiž řídí koncová aplikace, pro uživatele je důležitý pouze přehled.

Formulář pro novou notifikaci (obrázek B.8)
Stránka s formulářem pro vytvoření nové notifikace je na obrázku prázdná. Je tomu tak proto, že při tvorbě návrhu UI ještě neznáme konkrétní formulářové prvky. Budou zde zajisté vstupní pole pro titulek a text notifikace, dále pak nějaké nastavení plánovaných rozesílek apod. Důležitou součástí bude výběr cílových zařízení — možnost zvolit všechna zařízení či vybrat konkrétní zařízení.

Nastavení údajů API (obrázek B.9)
Stránka zobrazující nastavení přístupových údajů k API třetích stran pro zasílání push notifikací. Bude zde přehled nastavení s možností editace. Uživatel by měl být pravidelně upozorňován, že nemá tyto údaje vyplněné. Stránka bude obsahovat také informace o potřebných krocích, které povedou k úspěšnému zasílání push notifikací na danou platformu.

Výpis zařízení (obrázek B.10)
Jedná se o nikterak nefiltrovaný výpis zařízení, jsou zde tedy všechna zařízení přihlášená ke všem aplikacím uživatele. Každý záznam bude navíc obsahovat odkaz na detail aplikace, ke které je zařízení přihlášeno. Dále zde bude vyhledávací pole pro rychlé nalezení cílového zařízení či filtry dle aplikací, data přihlášení apod.

Uživatelské nastavení (obrázek B.11)
Stránka zobrazující nastavení uživatelského účtu. Na stránku se lze dostat pomocí odkazu ve vyskakovacím menu v pravé horní části stránky. Bude zde pouze umožněno uživateli změnit heslo a označit účet jako aktivní/neaktivní. Neaktivní účet poté neposílá žádné notifikace, i když má nějaké nastavené.

Námi vytvořený wireframe je pouze statický (obrázkový), neobsahuje žádné funkční ani ovládací prvky, které by umožňovali navigaci mezi jednotlivými pohledy apod. Slouží pouze jako podklad pro implementaci naší webové aplikace z hlediska uživatelského rozhraní či jako podklad pro návrh API. Informace zobrazené na jednotlivých pohledech drátěného modelu nemusí odpovídat následné realizaci webové aplikace. Drátěný model má hlavně ilustrativní význam.

4.4 Návrh databáze

Návrh databáze je důležitou součástí procesu návrhu systému. Měl by vycházet z analytického doménového modelu a ostatních poznatků získaných při analýze daného projektu. Tento model poté slouží jako dokumentace databáze, znázorňuje veškeré vazby mezi jednotlivými tabulkami a popisuje podrobně atributy daných tabulek. Podle tohoto modelu je poté jednoduché vytvořit odpovídající databázi, některé nástroje pro tvorbu databázových modelů dokonce umožňují vygenerovat skripty pro daný databázový stroj, pomocí kterých je databáze automaticky vytvořena podle všech náležitostí databázového modelu.

V našem případě je ovšem poněkud zbytečné vytvářet databázový model. Hlavním důvodem je využití databázového stroje MongoDB. Databázový stroj MongoDB je *open-source* dokumentová databáze. Záznamu v MongoDB databázi se říká dokument, který je tvořen datovou strukturou složenou dvojic z klíčů a hodnot. MongoDB dokumenty jsou podobné JSON objektům. Místo hodnot lze použít další dokumenty, pole apod. [9]

Díky využití MongoDB a jeho driveru pro node.js *mongoose* nám stačí nadefinovat jednotlivé javascriptové objekty, které odpovídají analytickému doménovému modelu. Tím nám vznikne databáze přesně podle našeho doménového modelu, čímž můžeme vynechat návrh databáze. Jednotlivé objekty navíc slouží jako entity pro použití v naší aplikaci — vyhneme se mapování databázových záznamů na objekty aplikace. Veškeré vnitřní záležitosti databáze jsou pro nás nedůležité, přistupujeme k databázi pouze jako k objektům v aplikaci, dochází tedy k abstrakci databázového stroje a databázová vrstva *mongoose* řeší vše za nás.

4.5 Specifikace API

Aby bylo možné využít poskytovanou funkcionalitu naší aplikace vzdáleně z různých zařízení či jiných aplikací, je nutné vytvořit rozhraní pro ovládání naší aplikace pomocí požadavků. K tomu slouží API (Application Programming Interface). Jedná se vlastně o souhrn připravených funkcí a příkazů, které může programátor využívat. Pokud tedy vytvoříme API, kdokoliv může ve své aplikaci komunikovat s naším serverem a propojit obě aplikace. V kontextu naší práce se jedná konkrétně o REST API (Representational State Transfer). REST je architektonický návrh, dle kterého bychom měli navrhnout naší API. Jelikož tvoříme webovou aplikaci, budeme pro naší aplikaci používat HTTP požadavky a vytvoříme tedy RESTful službu (službu, která se řídí REST principy).

REST architektonický styl popisuje komunikaci mezi klientem a serverem, je bezstavový, umožňuje využít cache (dočasná paměť uchovávající záznamy pro urychlení získávání dat) a má jednotné rozhraní. V RESTu vytváříme

zdroje (*resources* či *endpoints*), které mají svůj identifikátor (URI), reprezentaci (XML, JSON, ...) a klient může přistupovat k těmto zdrojům pomocí HTTP metod. Tyto zdroje korespondují s jednou nebo více entitami datového modelu.

Vlastnost jednotného rozhraní RESTful služby nám říká, jakým způsobem manipulovat se zdroji. Pro RESTful službu definujeme následující HTTP metody:

GET Získání dat (+ HEAD, OPTIONS)

PUT Modifikace či vytvoření dat (+ PATCH)

POST Vytvoření nových dat

DELETE Smazání dat

Metody jsou *bezpečné*, pokud nemění stav aplikace (nemodifikují data), výsledek může být uložen v mezipaměti — GET, OPTIONS, HEAD. Metody jsou *idempotentní*, pokud každé zavolání metody bude mít vždy stejný efekt — GET, PUT, DELETE. V naší aplikaci tedy budeme uvažovat tyto principy REST architektury a tímto způsobem vytvoříme celou API.

4.5.1 Návrh API

Základem pro návrh REST API našeho systému jsou sekce 3.3 a 4.3 — Model případů užití a Návrh UI. Díky jednotlivým případům užití a jednotlivým pohledům drátěného modelu lze vytipovat veškeré potřebné koncové body našeho API. Na základě tohoto návrhu poté vytvoříme automatické testy, abychom mohli provádět vývoj serverové části aplikace pomocí metody TDD (Test-driven development) — jedná se o vývoj řízený testy, tedy snažíme se implementovat funkcionality tak, abychom splnili veškeré předem definované automatické testy. Jak již bylo řečeno, při návrhu API se budeme držet principů REST tak, aby naše aplikace byla RESTful. Obecná URI pro API bude ve tvaru `{host}/api/`, aby bylo zřejmé, že požadavky míří právě na naší API.

Nejprve budeme potřebovat koncové body pro uživatele (případy užití — Registrace uživatele 3.7). Zde je nutné podotknout, že přihlášený uživatel získá další oprávnění v systému. Je tedy nutné nějakým způsobem uchovávat informaci o tom, že je uživatel oprávněn k těmto úkonům. Jelikož nemůžeme využít klasické *sessions* — komunikace probíhá na pozadí pomocí *XHR* požadavků — je nutné ke každému požadavku na server přidat informaci o tom, že požadavek vyvolal autorizovaný uživatel. Využijeme tedy *HTTP* hlavičku *Authorization*, která slouží přesně k tomuto účelu. Pokud se uživatel přihlásí do systému, systém vrátí přístupový *token*, který se bude přidávat ke každému požadavku do hlavičky *Authorization* — přístupový token bude vytvořen jako otisk kombinace uživatelských údajů pomocí hash funkce.

Pro tyto účely jsme definovali následující koncové body:

/users/login

POST Přihlášení uživatele do systému.

- **email** email uživatele
- **password** heslo uživatele

Response 401 Unauthorized

- **message** zpráva specifikující důvod selhání přihlašovacího procesu

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **user** objekt s informacemi o uživateli
- **token** token pro autorizaci následných požadavků

/users

POST Registrace nového uživatele do systému. Po úspěšné registraci bude uživatel automaticky přihlášen do systému.

- **email** email uživatele
- **password** heslo uživatele

Response 409 Conflict

- **message** zpráva specifikující důvod selhání registrace, typicky se jedná o již existující emailovou adresu

Response 201 Created

- **message** zpráva o úspěchu požadavku
- **user** objekt s informacemi o uživateli
- **token** token pro autorizaci následných požadavků

/users/logout

GET Odhlášení přihlášeného uživatele ze systému.

Response 205 Reset Content
Server neodesílá žádná data zpět, ale říká klientovi, že má resetovat *view* (např. refresh prohlížeče)

/users/forgottenPassword

POST Obnovení zapomenutého hesla uživatele.

- **email** email uživatele

Response 404 Not Found
uživatelský email zaslaný pro obnovu hesla nebyl nalezen

Response 200 Success

- **message** zpráva o úspěchu požadavku

Nyní definujeme koncové body pro případy užití — Správa aplikací 3.8. Jedná se o jednoduchý *CRUD*, tedy o vytváření, editace, mazání a čtení aplikací. Tyto koncové body jsme definovali následovně:

/applications

GET Získání všech vytvořených aplikací daného uživatele.

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **applications** pole objektů získaných aplikací

POST Vytvoření nové aplikace.

- **application** data aplikace získaná z klientského formuláře

Response 422 Unprocessable Entity
data odeslaná na server nejsou validní

- **message** zpráva o neúspěchu požadavku

Response 201 Created

- **message** zpráva o úspěchu požadavku
- **application** objekt nově vytvořené entity, obsahuje data včetně identifikátoru

/application/:id

GET Získání aplikace pomocí id.
Příklad: `GET /application/12`

Response 404 Not Found
záznam pod požadovaným id nebyl nalezen

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **application** objekt získané aplikace

PUT Editace dříve vytvořené aplikace.
Příklad: `PUT /application/12`

- **application** data aplikace získaná z klientského formuláře pro editaci záznamu

Response 404 Not Found
záznam pod požadovaným id nebyl nalezen

Response 422 Unprocessable Entity
data odeslaná na server nejsou validní

- **message** zpráva o neúspěchu požadavku

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **application** objekt entity s novými daty

DELETE Smazání dříve vytvořené aplikace.
Příklad: `DELETE /application/12`

Response 404 Not Found
záznam pod požadovaným id nebyl nalezen

Response 200 Success

- **message** zpráva o úspěchu požadavku

Koncový bod pro registraci zařízení je závislý na dříve vytvořené aplikaci. Odebrání zařízení probíhá již na základě identifikátoru daného zařízení. Výpis zařízení by měl být jednak kompletní a jednak filtrovaný dle dané aplikace. Tyto endpointy vyplývají z případů užití — Registrace / Odebrání zařízení 3.9. Na základě těchto informací definujeme následující koncové body:

/devices

GET Získání všech registrovaných zařízení daného uživatele.

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **devices** pole objektů získaných zařízení, entita obsahuje mimo dat o samotném zařízení také informaci o vazbě na aplikaci

/application/:id/devices

GET Získání přihlášených zařízení dané aplikace pomocí id aplikace.
Příklad: `GET /application/12/devices`

Response 404 Not Found
záznam pod požadovaným id nebyl nalezen

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **application** objekt získané aplikace

POST Registrace zařízení k odběru dané aplikace.
Příklad: `POST /application/12/devices`

- **device** data daného zařízení včetně identifikátoru pro komunikaci s API třetích stran

Response 404 Not Found
aplikace pod požadovaným id nebyla nalezena

Response 409 Conflict
zařízení je již registrováno k odběru

Response 201 Created

- **message** zpráva o úspěchu požadavku
- **device** objekt nově vytvořené entity

/application/:appId/device/:id

DELETE Odebrání cílového zařízení na základě jeho identifikátoru.
Příklad: `DELETE /application/12/device/23`

Response 404 Not Found
záznam pod požadovaným id nebyl nalezen

Response 200 Success

- **message** zpráva o úspěchu požadavku

Pro případy užití — Správa údajů API 3.10 je potřebný pouze koncový bod pro editaci a získání údajů. S registrací nového uživatele bude již vytvořena entita uchovávající nastavení pro komunikaci s API třetích stran. Tato entita bude unikátní pro každého uživatele, není tedy nutné pro definování koncových bodů uvažovat nějaký identifikátor pro určení entity údajů API. Prvotní vyplnění údajů bude tedy pouze editace prázdných polí. Konečné endpointy definujeme tedy takto:

/user/:id/apiConfig

GET Získání údajů pro API třetích stran daného uživatele.

- **:id**
  URL parametr může být nahrazen řetězcem „me“, který pomocí autorizační hlavičky dohledá přihlášeného uživatele, obecně bychom tento parametr nepotřebovali, ale chceme zachovat principy REST

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **settings** objekt obsahující nastavení údajů API třetích stran

PUT Změna údajů pro API třetích stran.
Příklad: `PUT /user/:id/apiConfig`

- **:id**
  URL parametr může být nahrazen řetězcem „me“, který pomocí autorizační hlavičky dohledá přihlášeného uživatele, obecně bychom tento parametr nepotřebovali, ale chceme zachovat principy REST
- **settings**
  data s údaji pro API třetích stran získaná z klientského formuláře pro editaci záznamu

Response 422 Unprocessable Entity
data zaslaná na server nejsou validní

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **settings** objekt obsahující nová data dle požadavku

Obecně správa notifikací je klíčovou vlastností našeho systému. Každá notifikace je opět svázaná s aplikací, tudíž je nutné na to myslet i při navrhování koncových bodů naší serverové části. V této části návrhu API jsme spojili dva logické bloky případů užití, neboť z hlediska API spolu úzce souvisejí a není nutné vytvářet redundantní endpointy. Na základě případů užití — Správa notifikací 3.11 a Rozesílání notifikací 3.12— jsme tedy určili veškeré potřebné endpointy pro *CRUD* notifikací následujícím způsobem:

/application/:id/notifications

GET Získání všech notifikací pro danou aplikaci přihlášeného uživatele. Filtrování aplikací na základě parametru *:id* v URL.
Příklad: `GET /application/12/notifications`

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **notifications** pole objektů získaných notifikací

Response 404 Not Found
aplikace dle zadaného id nenalezena

POST Vytvoření nové notifikace přiřazené aplikaci na základě parametru *:id*.
Příklad: `POST /application/12/notifications`

- **notification** data nově přidávané notifikace

Response 404 Not Found
aplikace pod požadovaným id nebyla nalezena

Response 422 Unprocessable Entity
data odeslaná na server nejsou validní

- **message** zpráva o neúspěchu požadavku

Response 201 Created

- **message** zpráva o úspěchu požadavku
- **notification** objekt nově vytvořené entity

/application/:appId/notification/:id

GET Získání dat notifikace na základě parametru *:id* pomocí identifikátoru aplikace (*:appId*).
Příklad: `GET /application/12/notification/5`

Response 404 Not Found
záznam pod požadovaným id nebyl nalezen

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **notification** objekt získané notifikace

PUT Úprava dat notifikace na základě parametru *:id* pomocí identifikátoru aplikace (*:appId*).
Příklad: `PUT /application/12/notification/5`

- **notification** nová data aplikace získaná z klientského formuláře

Response 404 Not Found
záznam pod požadovaným id nebyl nalezen

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **notification** objekt notifikace vyplněný novými daty

DELETE Smazání notifikace na základě identifikátoru *:id*.
Příklad: `DELETE /application/12/device/23`

Response 404 Not Found
záznam pod požadovaným id nebyl nalezen

Response 200 Success

- **message** zpráva o úspěchu požadavku

Případy užití logického bloku — Zobrazení statistik 3.13 — nám popisují možnost zobrazení různých statistik (např. odeslaných notifikací v daném časovém úseku). Jelikož se zde jedná hlavně o filtry, budeme používat getové parametry pro filtrování. Zatím ještě nevíme, jaké všechny údaje budeme z API získávat, proto se omezíme při návrhu na jeden koncový bod — jedná se pouze o získávání dat:

/summary

GET Získání statistik s možností filtrování dle parametrů.
Příklad: `GET /summary?dateFrom=2017-01-01&dateFrom=2017-02-01`

- **dateFrom** datum, od kterého se mají zobrazit statistiky
- **dateTo** datum, do kterého se mají zobrazit statistiky
- **applicationId** identifikátor filtrující statistiky dle aplikací

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **summary** objekt se statistikami

Jelikož máme již koncový bod pro odhlášení nadefinovaný, zbývá nám na základě případů užití — Správa uživatelského účtu 3.14 — navrhnout pouze koncový bod pro editaci uživatele:

/user/:id

PUT Editace dat přihlášeného uživatele. Místo parametru *:id* může být opět použit zástupný řetězec „me“ — uživatel je poté dohledán na základě tokenu v autorizační hlavičce.
Příklad: `PUT /user/me`

Response 200 Success

- **message** zpráva o úspěchu požadavku
- **summary** objekt obsahující entitu uživatele s novými daty

Response 422 Unprocessable Entity
data odeslaná na server nejsou validní

- **message** zpráva o neúspěchu požadavku

Nadefinováním základních koncových bodů nám vznikl podklad pro tvorbu automatických testů pro naší serverovou část systému. Nejprve vytvoříme dle navržených záležitostí automatické testy pro koncové body, které doplníme o konkrétní kontrolu přenášených dat. Následně opakovaným spouštěním testů se budeme snažit implementovat logiku jednotlivých endpoint tak, aby námi definované testy byly úspěšné.

Této metodě vývoje se říká *Test-driven development*, tedy vývoj řízený testy. Výhodou tohoto přístupu je, že máme do budoucna již definované testy, které musí být vždy úspěšné. Pokud tedy chceme předělat část kódu, nějaký endpoint nebo logiku systému, musíme vždy splnit testy a máme jistotu, že API bude tedy v každém případě konzistentní. Klientský kód tedy bude i při změnách našeho API použitelný.

Pro přehlednost ještě uvádíme soupis všech námi navržených endpointů a jejich metod a stručného popisu, abychom měli nějaký obecný přehled pro další práce:

/applications

GET získání všech aplikací

POST vytvoření nové aplikace

/application/:id

GET získání aplikace dle *:id*

PUT editace aplikace dle *:id*

DELETE smazání aplikace dle *:id*

/application/:id/devices

GET získání všech zařízení dle *:id* aplikace

POST registrace zařízení k aplikaci dle *:id* aplikace

/application/:appId/device/:id

DELETE odebrání zařízení s *:id*

/application/:id/notifications

GET získání notifikací dle *:id* aplikace

POST vytvoření nové notifikace dle *:id* aplikace

/application/:appId/notification/:id

GET získání notifikace s *:id*

PUT změna notifikace s *:id*

DELETE smazání aplikace dle *:id*

/devices

GET získání všech zařízení

/summary

GET získání statistik

/users

POST registrace uživatele

/users/forgottenPassword

POST obnovení zapomenutého hesla

/users/login

POST přihlášení do systému

/users/logout

GET odhlášení uživatele

/user/:id

PUT úprava uživatelských dat

/user/:id/apiConfig

GET získání údajů API třetích stran

PUT změna údajů API třetích stran

Kapitola 5

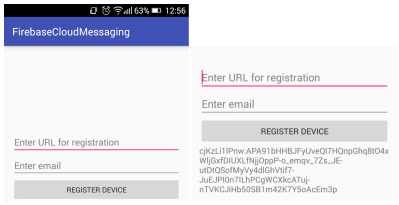
Realizace

Kapitola se zabývá procesem realizace cílového systému, který je předmětem této práce. Popisuje předpoklady pro vývoj, použité techniky vývoje, testování implementovaných funkcí, nestandardní části aplikace a zjištěné problémy a poznatky.

Základem pro implementaci služby pro zasílání push notifikací jsou předchozí kapitoly (zejména *Analýza* (3) a *Návrh* (4)). Díky nabytým poznatkům a přípravě v podobě návrhu aplikace z hlediska uživatelského rozhraní, API a jednotlivých entit systému jsme byli schopni úspěšně implementovat výsledný systém. Systém se skládá ze serverové části, která persistentně uchovává data, řidí logiku aplikace, komunikuje na pozadí se službou Firebase Cloud Messaging [6] a poskytuje API pro vzdálené ovládání funkcionalit. Další částí je webový klient, který poskytuje uživatelské rozhraní pro pohodlnou obsluhu uživatelem. Uživatel tedy není nucen volat složité příkazy na API, aby mohl vytvořit v našem systému uživatelský účet, aplikace, notifikace apod. Uživatelem definované notifikace jsou poté odesílána na cílová zařízení. Zde jsme narazili na jeden zásadní problém, nicméně více si jej přiblížíme v sekci *Problémy a nedostatky* 5.7.

5.1 Předpoklady pro vývoj

Pro vývoj jsme v první řadě potřebovali na lokálním stroji Node.js [7]. Jelikož s touto technologií pracujeme téměř denně, spoléhali jsme na to, že naše verze bude pro vývoj dostatečná. Ovšem po naklonování repozitáře *Redux universal exampl* [10] a následném spuštění příkazu pro instalaci závislostí pomocí správce balíčku — *npm install* — jsme zjistili, že pokud chceme spustit vývojový server a klienta (příkaz *npm run dev*), potřebujeme novější verzi Node.js. Nainstalovali jsme tedy nejnovější verzi (verze 6.9.5). Při opětovném spuštění vývojového serveru již proběhlo vše bez problému a nám se podařilo zobrazit v prohlížeči úvodní stránku vzorového projektu.



Obrázek 5.1: Ukázka testovací aplikace pro android

Dalším zásadním krokem bylo nainstalování a zprovoznění MongoDB [9] databázového stroje. Řídili jsme se instalačním návodem na stránkách projektu — viz `https://docs.mongodb.com/manual/administration/install-community/`. Na této stránce lze nalézt návod na platformy Windows, Linux i OS X. Jakmile jsme úspěšně zprovoznili MongoDB databázi, přidali jsme k našemu projektu závislost na balíček *mongoose*. Tento balíček poskytuje rozhraní pro práci s MongoDB v Node.js, my jsme nejprve však otestovali pouhé připojení k databázi. Tímto jsme úspěšně propojili vzorový projekt s MongoDB databázovým strojem.

Abychom mohli zdárně implementovat zasílání push notifikací na cílová zařízení, bylo nutné vytvořit testovací aplikaci pro mobilní platformy. Rozhodli jsme se, že vytvoříme testovací aplikaci pouze pro platformu Android, jelikož je v dnešní době nejrozšířenější mobilní platformou a sami jsme majitelem zařízení s tímto systémem. V principu bylo nutné vytvořit aplikaci, která bude mít implementovaný *Firebase Cloud Messaging* [6] a při svém spuštění zavolá URI pro přihlášení zařízení k odběru notifikací. Jelikož implementace takovéto aplikace by byla nad rámec této diplomové práce, rozhodli jsme se porozhlédnout po existujícím řešení. Uspěli jsme částečně. Nalezli jsme internetový článek, jehož autorem je Belal Khan [14].

Článek *Firebase Cloud Messaging for tutoriál for Android* [14] pro nás byl velmi prospěšný z hlediska pochopení principů propojení Android aplikace a služby Firebase Cloud Messaging. Abychom nemuseli vytvářet aplikaci od nuly, rozhodli jsme se stáhnout ukázkový projekt k tomuto článku. Jedná se o Android Studio projekt, který je připravený pro doimplementování push notifikací — zejména nastavení URI pro přihlášení/odhlášení odběru notifikací a doplnění údajů přidružené Firebase aplikace. Abychom si usnadnili vývoj, modifikovali jsme kód námi stažené aplikace tak, abychom mohli do vstupního pole zadat URL adresu pro přihlášení odběru push notifikací a do druhého pole zadat email. Následně jsme přidali tlačítko *REGISTER DEVICE*, které odešle POST požadavek na danou URL a přiloží k požadavku email a unikátní identifikátor zařízení, díky kterému lze zasílat notifikace pomocí služby FCM (obrázek 5.1). Po odeslání navíc aplikace zobrazí daný unikátní identifikátor sloužící pro kontrolu či další testovací případy.

Pomocí FCM testovací konzole `https://console.firebase.google.com/`

jsme si ozkoušeli odeslání testovací notifikace, která byla úspěšně doručena na naše zařízení. Dalším krokem tedy bylo zvolit vhodnou knihovnu pro komunikaci mezi naším serverem a službou Firebase Cloud Messaging. Podařilo se nám najít knihovnu *fcm-node*, která zapouzdřuje požadavky odesílané na FCM server (`https://github.com/jlcvp/fcm-node`). Práci s knihovnou jsme si nejprve vyzkoušeli v naše Node.js serveru. Po úspěšném doručení notifikace na naše testovací zařízení jsme připravili funkce pro vytvoření požadavku pro zaslání push notifikace pomocí FCM a ověření přístupového tokenu. Na ověření přístupového tokenu jsme navíc implementovali unit test, abychom věděli, že ověření probíhá správným způsobem.

Ověření FCM tokenu a zaslání požadavku na FCM službu

```
import FCM from 'fcm-node';

export const checkValidToken = (token, cb) => {
  const fcm = new FCM(token);
  fcm.send({to: ''}, function(err, response){
    err ? cb(err) : cb(null, response)
  });
}

export const sendNotification =
(serverKey, target, data, cb) => {
  const fcm = new FCM(serverKey);
  const message = {
    to: target,
    collapse_key: 'pushnotif',
    notification: data,
  };

  fcm.send(message, function(err, response){
    err ? cb(err) : cb(null, response)
  });
};
```

Tímto jsme splnili předpoklady pro úspěšný vývoj našeho systému. Nastavili jsme si vývojové prostředí a zprovoznili veškeré nástroje pro implementaci. Vytvořili jsme testovací aplikaci pro Android a vyzkoušeli v praxi doručování push notifikací.

5.2 Techniky vývoje

Na základě sekce návrhu API 4.5.1 jsme nejprve vytvořili automatické testy pro všechny potřebné koncové body pro API serverové části aplikace. Tyto testy kontrolují správnost implementace jednotlivých *ednpointů*, testují všechny možné typy odpovědí, které může daný koncový bod vracet. Tímto způsobem testování tedy můžeme ověřit správnost implementace API koncových bodů a tedy celkového průchodu požadavku naším serverem. Z hlediska dělení testů se tedy jedná o integrační testování, které testuje správnost interakce mezi dílčími bloky aplikace. Požadavek je vyhodnocován v naše případě postupně. Nejprve se dostane na router, který určí, jaká akce se má vykonat. Pak se aplikuje veškerý *middleware*, který je pro danou routu definován. Poté je požadavek (v některých případech obohacený o potřebné údaje pomocí middlewaru) předložen akci. Daná akce zpracuje požadavek — typicky provede operaci s databází a odešle odpověď klientovi. Kontrolou daného koncového bodu tedy testujeme veškeré dílčí části systému, tedy provádíme integrační testování. Také by se dalo na testy nahlížet jako na systémové, neboť zároveň testujeme aplikaci jako celek — na požadavek očekáváme danou odpověď i bez znalosti vnitřní logiky systému. Rozhodnutí o druhu testů je poměrně subjektivní, není však pro nás moc důležité, důležitým faktem je, že máme otestovanou API část našeho serveru a každá změna v implementaci této části nám musí splnit veškeré testy. API tedy zůstane konzistentní a zpětně kompatibilní.

Po nadefinování jednotlivých testů jsme začali dokola spouštět testy pomocí příkazu *npm run test-api*. Příkaz nalezne všechny soubory, které v názvu mají zakončení „-test.js“ a začne postupně spouštět jednotlivé testy. Pokud nějaký test selže, další se neprovádí. S opakovaným spouštěním jsme postupně implementovali funkcionalitu serverové části tak, aby jsme splnili vždy naposledy neúspěšný test. Tímto přístupem jsme vytvořili kompletně API serveru. Této metodě vývoje softwaru se říká „vývoj řízený testy“ z anglického *Test-driven development* (TDD). Každou další změnu API, která byla rozdílná oproti prvotní implementaci na základě návrhu, jsme podrobili opět automatickému otestování, čímž nám zůstala konzistentní API a zároveň dokumentace API.

Po vytvoření automatických testů a následném vytvoření API jsme potřebovali vytvořit dokumentaci pro jednotlivé koncové body. Koncový bod je definovaný pomocí routeru z frameworku *express*, kde ke každé URI je přiřazena funkce, která se zavolá a obslouží požadavek — takto nadefinovaná dvojice se nazývá „routa“. Jelikož jsme jednotlivé routy rozdělili dle logických celků na více souborů, aby byli přehlednější a tvořili jakési moduly, nabízela se nám možnost definovat dokumentaci pomocí komentářů v daných souborech s definicí rout. Za tímto účelem jsme se rozhodli využít nástroj *apiDoc* [15]. Tento nástroj nám vygeneroval kompletní dokumentaci API na základě námi definovaných informací v komentářích a vygenerovanou dokumentaci uložil do složky kořenového adresáře projektu — adresář „apidoc“. V adresáři se na-

chází soubor „index.html“, který obsahuje kompletní dokumentaci API našeho projektu.

Jelikož jsme použili Node.js framework *express*, bylo nám umožněno využívat middleware. Middleware je v našem kontextu funkce, která rozšiřuje základní funkcionalitu systému a je vykonána před obsluhou požadavku. Jednotlivé middleware funkce se dají řetězit a jejich využití je téměř všestranné. Middleware funkce má tři parametry — requset, response a next. Request je příchozí požadavek aplikace, response je objekt pro obsluhu požadavku a next je funkce, která se volá po vykonání middlewaru. Využili jsme middleware funkci, která kontroluje, zda je uživatel autorizovaný a má přístup k danému koncovému bodu. Tímto nám odpadl problém se zabezpečením každé routy zvlášť a naše middleware funkce slouží jako ACL — seznam pro řízení přístupu.

Ukázka middleware funkce pro autorizaci uživatele

```
const checkPermissions = (req, res, next) => {
  if (NOT_SECURED_ROUTES.indexOf(req.url) < 0) {
    const token = req.headers['authorization'];
    if (token) {
      UserModel.verifyToken(token, (err, user) => {
        if (err || !user) {
          return res.status(403).send({
            message: 'You are not allowed ...'
          });
        } else {
          // if everything is good, continue
          req.user = user;
          req.session.user = user;
          next();
        }
      });
    } else {
      return res.status(403).send({
        message: 'You are not allowed ...'
      });
    }
  } else {
  //not secured routes, continue
    next();
  }
};
```

Obrázek 5.2: Popis důležitých souborů a adresářů projektu

```
api/ ..... zdrojové kódy serveru
  __tests__/ ..... automatické testy
  actions/ ..... akce obsluhující požadavky a jejich routy
  models/ ..... soubory entit a práce s databází
  api.js ..... vstupní soubor serveru
  authorization.js ..... middleware pro autorizaci
  routes.js ..... definice rout
apidoc/ ..... dokumentace API
  index.html ..... soubor s dokumentací API
bin/ ..... vstupní soubory serveru a klienta
node_modules/ ..... knihovny stažené pomocí npm
src/ ..... zdrojové kódy klienta
  components/ ..... komponenty
  containers/ ..... jednotlivé pohledy
  redux/ ..... definované reducery a akce
  routes.js ..... definované klientské routy
  config.js ..... konfigurační soubor aplikace
  server.js ..... vstupní soubor pro klientský kód
package.json ..... soubor s balíky a informacemi o projektu
```

Při implementaci webového klienta jsme využili předem vytvořené dokumentace API a navrženého drátěného modelu 4.3. Začali jsme poté vytvářet jednotlivé react komponenty [8], které komunikují s API serverové části systému a tvoří uživatelské prostředí. Rozvržení UI a jednotlivé komponenty jsme vytvářeli podle jednotlivých pohledů drátěného modelu. Tímto způsobem jsme byli schopni vytvořit kompletního webového klienta.

5.3 Projekt

Projekt byl založen na *redux-universal-example* [10], který obsahuje ukázkový projekt se serverovou částí s API vytvořenou v Node.js a klientskou částí vytvořenou v reactu. Jedná se o kompletně nastavený balík knihoven, který umožní rychlou a efektivní implementaci projektu. V první řadě jsme nastavili konfigurační soubory a odstranili z projektu veškeré nepotřebné fragmenty kódu — ukázkové komponenty, routování, serverovou obsluhu apod. Vznikl nám tak prázdný projekt s námi požadovaným nastavením a propojenou serverovou a klientskou částí. Dále jsme již implementovali čistě naší funkcionalitu. Adresářová struktura projektu je zobrazena na obrázku 5.2

5.4 Server

Jak již bylo řečeno, serverová část byla implementována pomocí přístupu TDD. Jednotlivé požadavky jsou obsluhovány pomocí *akcí*. Tyto akce tvoří jakousi řídící vrstvu a dále komunikují s modelovou vrstvou, která má za úkol především komunikovat s databází. Jedná se o velmi jednoduchou interakci mezi těmito dvěma vrstvami, akce především delegují CRUD požadavky na modely. Akce se nacházejí ve složce *api/actions/* a jsou rozděleny dle logických celků na moduly. Modely se nacházejí ve složce *api/models/* a především se jedná o definování entit pro MongoDB a jejich restrikcí.

Ukázka kódu akce —- získání všech aplikací

```
export const getApps = (req, res) => {
      let query = AppModel.find({user: req.user._id});
      query.exec((err, apps) => {
        if(err) res.send(err);
        res.json({applications: apps});
      });
    }
```

Ukázka kódu modelu — model aplikací

```
// Defining application entity
    const Schema = mongoose.Schema({
      name: {
        type: String,
        required: true,
      },
      ...
      ,
      user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true,
      },
      devices: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Device',
      }]
    });
```

Důležitou součástí serveru je definování rout. Nejprve definujeme masky pro URI, se kterými je požadovaná URI porovnávána, a k nim přiřadíme

akci. tato akce má za úkol obsloužit požadavek a odeslat uživateli odpověď. K jednotlivým routám se ještě definuje HTTP metoda, pro kterou je daná routa přístupná. Definice rout poté vypadá přehledně a programátor přesně ví, co se v aplikaci při příchozím požadavku děje.

Ukázka definice rout

```
//api.js
app.use('/api/', router); // import router.js

//router.js
const router = express.Router();
router.use(devicesRouterUnsecured);

// Our easy ACL
router.use(checkPermissions); // Middleware function

router.use(devicesRouter);
router.use(appsRouter); // import apps/index.jd

//apps/index.js
const router = express.Router();

router.route('/applications')
  .get(actions.getApps)
  .post(actions.postApps);

router.route("/application/:id")
  .get(actions.getApp)
  .delete(actions.deleteApp)
  .put(actions.updateApp);

export default router;
```

Z ukázky kódu je patrné, že ve vstupním souboru serveru pouze využíváme importovaný router z souboru *routes.js*. V tomto souboru aplikujeme náš middleware a slučujeme routy jednotlivých modulů. Ukázkový modul *apps/* má definované jednotlivé routy, jejich povolené metody a obslužné akce. Vše je naprosto přehledné, snadno obměnitelné a rozšiřitelné. Dále bychom mohli k jednotlivým modulům například přidávat další specifický middleware pokud by byl třeba.

5.5 Webový klient

Webový klient je vytvořen pomocí reactu [8] a reduxu [11]. Kód je vykonáván prohlížečem uživatele. Takto vytvořená aplikace si spuštění načte výchozí stav a poté na každou interakci s uživatelem obnovuje své dílčí komponenty, kterých se změna stavu týká. Stav je ukládán to *redux storu*. Tento stav mohou měnit pouze *reducery*, což jsou funkce, které dle typu akce provedou změnu vstupního stavu a vracejí nový stav aplikace. Tohoto faktu se dá využít například při uchovávání historie změn apod. — list stavů, které se dají libovolně zaměňovat. Pro persistenci změn je nutné data uložit na server. Aplikace si uchovává stav pouze po dobu svého běhu — tedy do refreshe prohlížeče, zavření okna apod. Každá instance v prohlížeči má svůj vlastní stav, znamená to tedy, že změny provedené jednom okně se neaplikují na okno druhé. Je zde nutná interakce se serverovou API pro uchování a získání nových dat.

Klientský kód je pro člověka neznalého dané problematiky poněkud složitý, chtěli bychom se tedy vyvarovat ukázek kódu, které by mohli být nic neříkající. Pro podrobnosti o zdrojovém kódu klientské aplikace navštivte repozitář na Githubu naší aplikace `https://github.com/Degee/pushnotif`.

5.6 Nestandardní části aplikace

Cílem této práce bylo vytvořit službu pro zasílání push notifikací. Kromě serveru pro správu a webového klienta pro pohodlné ovládání systému bylo nutné vytvořit vnitřní implementaci zasílání push notifikací. Jelikož může být odesílání notifikací naplánováno dopředu, je nutné periodicky kontrolovat, zda se v systému nenachází nějaká notifikace, která by měla být odeslána. Za tímto účelem jsme využili knihovnu *node-cron*, která umožňuje periodicky volat libovolnou funkci. Implementovali jsme v minutovém intervalu dotazování na databázi, zda nemají být nějaké notifikace rozeslány. Dotazujeme se v minutovém intervalu, avšak v databázi vyhledáváme notifikace, které nemají nastavený příznak „odesláno“ a datum a čas jejich odeslání je nastaven na *nyní - 10 minut*. Tímto bychom se měli vyhnout problému, že se nějaká naplánovaná notifikace neodešle.

Ukázka kódu — automatické rozesílání notifikací

```
//api.js
import runNotificationsChecker from './models/Cron';
runNotificationsChecker();

//models/Cron.js
const runNotificationsChekcer = () => {
  console.log("Running cron for notifications check!");
  cron.schedule(' * * * * *', () => {
```

```
    NotificationsModel.find({
      datetime: {
        $gte: moment().subtract(10, 'minutes').toDate(),
        $lt: moment().toDate()
      },
      isSent: false
    })
    .populate('app devices')
    .exec((err, notifications) => {
      notifications.map(item => {
        if (!item.devices.length) {
          Object.assign(item, {isSent: true}).save();
          return;
        }
        getSettingsByApp(item.app, (err, settings) => {
          const serverKey = settings.fcmToken;
          sendNotification(
            serverKey,
            item.devices.map(d => d.registeredId),
            {
            title: item.title,
            body: item.text
            }, (err) => {
                if (!err)
                    Object.assign(item, {isSent: true})
                        .save();
          })
        });
      });
    });
  });
};
```

Tento kód nejprve nastaví opakování vnitřní funkce pomocí plánovače úloh (*cron.schedule*). Vnitřní funkce se tak provádí každou minutu a jejím úkolem je v první řadě nalézt všechny doposud neodeslané notifikace, jejichž odeslání bylo naplánováno na aktuální čas. Vyhledáváme však veškeré neodeslané notifikace za posledních deset minut — tímto se vyhneme neobsloužení nějaké notifikace z důvodu výpadku serveru, zahlcení notifikacemi apod. K dotazu pro získání notifikací jsou pomocí funkce *populate* doplněny entity, na které má notifikace referenci — notifikace je přiřazená k aplikaci a má k sobě přiřazená cílová zařízení. Dále kontrolujeme, zda notifikace má nějaká cílová zařízení. Může se stát, že naplánujeme zaslání notifikace na zařízení, které se však odhlásí z odběru, tudíž není kam zaslat notifikace a můžeme ji označit za ode-

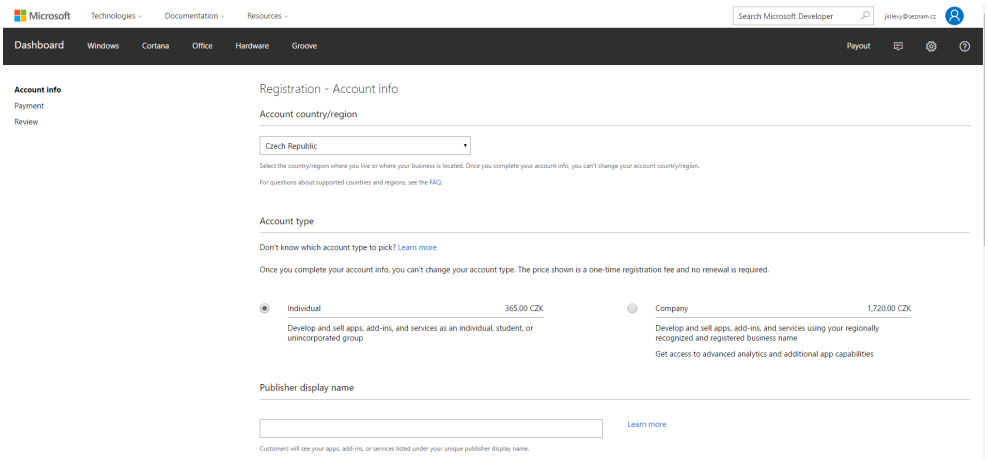
slanou a pokračovat dalším nalezeným záznamem. Pokud máme nějaká cílová zařízení, nalezneme si k dané notifikaci potřebný klíč pro komunikaci s FCM a odešleme notifikaci. V případě úspěchu ji označíme jako odeslanou.

V tomto fragmentu kódu se nám nabízí možnost zachytávat chybové hlášky, ovšem pro naše účely to není podstatné. Tímto způsobem lze odeslat notifikace až na 1000 zařízení. Pokud bychom chtěli zasílat na více cílových zařízení danou notifikaci, museli bychom kontrolovat počet zařízení dané notifikace a rozdělit její odeslání na dávky po 1000 zařízeních, čímž bychom obsloužili na více požadavků všechna požadovaná zařízení. Jsme si této slabiny vědomi, ovšem zpracování námi navrhovaného řešení necháme na budoucí rozšiřování projektu.

5.7 Problémy a nedostatky

V rámci realizace projektu nastal závažný problém. Pro implementaci zasílání push notifikací na platformu Windows Phone jsme také vyhledali knihovnu pro Node,js, která umožňuje komunikovat s Windowd Notification Service. Jedná se o knihovnu *wns* (`https://github.com/tjanczuk/wns`), jenž umožňuje zaslat push notifikace přihlášeným zařízením platformy Windows. Stačí pouze zadat údaje pro přístup k API — *client_id* a *client_secret*. Tyto údaje jsou vždy spjaty s konkrétní aplikací, tudíž jsme chtěli rozšířit naší entitu *application* o přístupové údaje k WNS API. Nejprve jsme si však chtěli ověřit funkcionalitu. Postupovali jsme podle návodu v oficiální dokumentaci Microsoftu pro WNS [16]. Při vytváření developerského účtu jsme však narazili na problém — vývojářský účet Microsoftu je placený (obrázek 5.3). Provedli jsme tedy ještě nějaký výzkum alternativ, avšak nenalezli jsme žádnou službu zdarma, která umožňuje zasílat push notifikace pro mobilní platformu Windows. Rozhodli jsme se i přes nesplnění zadání neimplementovat zasílání push notifikací pro platformu Windows Phone. Odradilo nás kromě placeného vývojářského účtu také množství potřebných nástrojů pro implementaci a otestování korektní funkcionality push notifikací. Museli bychom totiž implementovat navíc aplikaci pro klientské zařízení platformy Windows Phone. S vývojem pro tuto platformu nemáme žádné zkušenosti a samotná realizace by se časově protáhla na neúnosnou úroveň.

Naše řešení má samozřejmě další drobné chyby, o kterých sice víme, ale z časových důvodů jsme se rozhodli nepřikládat jim velkou váhu. Jedná se hlavně o požadavky na API, kde parametry u metod PUT jsou zpracovávány přímo z požadavku. Zde by měl být nějaký seznam povolených atributů, které lze modifikovat. Zasláním upraveného požadavku totiž lze modifikovat i atributy, které by za normálních okolností neměly být přístupné. Avšak implementovali jsme ke každé entitě kontrolu na uživatele, zda se jedná o jím vytvořený záznam a má tedy přístup pouze k údajům, které sám vytvořil. Nepředpokládáme, že by si uživatel chtěl z nějakého důvodu podvrhnutím požadavku



Obrázek 5.3: Vývojářský účet Microsoftu

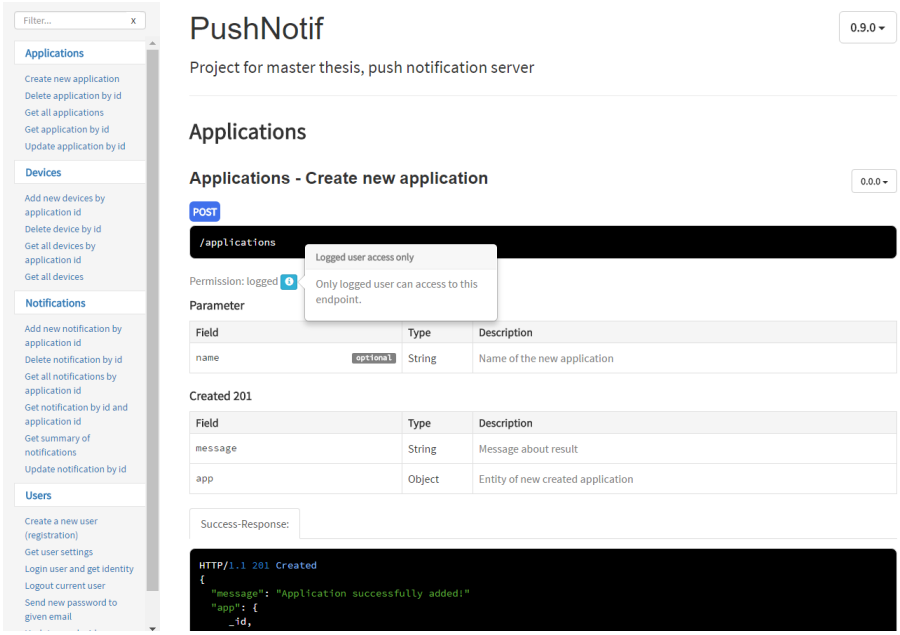
měnit své vlastní údaje. To je důvod, proč jsme upustili od kontroly těchto parametrů. Správně by však tyto *whitelisty* měly být implementovány.

Při vytváření notifikace a plánování její odesílky jsme vypustili možnost opakovaného zasílání. Uvědomili jsme si, že se jedná o poměrně náročnou funkcionalitu z hlediska databáze. buď bychom museli duplikovat záznamy vždy tak, aby byly vyplněné na určitý počet opakování dopředu a postupem času přidávat další a další záznamy, nebo bychom museli u všech notifikací dopočítávat, zda nemají být nyní odeslány. Další problém by nastal u mazání notifikací, potřebovali bychom rozhodnout, zda smažeme pouze jeden záznam nebo všechny. Stejný případ nastává při editaci notifikace či přidávání a odebírání zařízení. Rozhodli jsme se tedy z důvodu velkého množství práce poměrně malou funkcionalitu neimplementovat tento případ užití.

Jak již bylo řečeno, nedostatkem je také možnost zasílání notifikace pouze na 1000 zařízení. Jedná se o limit jednoho požadavku, lze tedy jednoduchým způsobem docílit toho, aby byla obsloužena všechna cílová zařízení nad počet 1000. Stačí rozložit požadavek na zaslání notifikace do více požadavků a zařízení obsluhovat v dávkách. Dobře víme, jak tento nedostatek odstranit, nepřišlo nám však reálné dosáhnout takového počtu přihlášených zařízení v nejbližší době. Plánujeme pokračovat s vývojem i mimo tuto diplomovou práci a určitě jedna z prvních změn bude odstranění tohoto nedostatku.

Dalším nedostatkem, který jsme zaznamenali, jsou statistiky. Představovali jsme si implementovat nejrůznější grafy pro znázornění přehledů a statistik. Z důvodu nedostatku času jsme ovšem tyto statistiky vypustili a rozhodli se je implementovat do budoucna v rámci rozšiřování projektu mimo tuto diplomovou práci. uživatel má nyní k dispozici pouze základní přehledy, ze kterých nelze vyčíst mnoho užitečných informací.

Rozhodně bychom také měli zabezpečit formulář pro registraci a obno-



Obrázek 5.4: Ukázka dokumentace API

vení zapomenutého hesla — například pomocí CAPTCHA testování (opisování kontrolních kódů) . Jelikož po registraci je automaticky uživatel přihlášen bez nutnosti ověření emailu, mohl by nám nějaký robot zahltit systém neaktivními uživateli, případně automaticky resetovat hesla uživatelům, pokud by věděl, jaké uživatele v systému máme. Tímto bychom se ovšem zabývali až v reálném provozu, jen nám přišlo důležité podotknout tuto zranitelnost.

5.8 Dokumentace API

Součástí projektu je dokumentace námi vytvořené API. Za tímto účelem jsme využili nástroj *apiDoc* [15], který automaticky generuje z anotací v komentářích zdrojového kódu dokumentaci. K našim souborům s definicí koncových bodů jsme přidali komentáře s anotacemi, které popisují dané routy, čímž jsme byli schopní vygenerovat kompletní dokumentaci ke každému koncovému bodu API serverové části našeho projektu. Tato dokumentace je napsaná v anglickém jazyce, aby mohla být využitá případnými zájemci o náš projekt. Věříme, že API serverové části, která je otestovaná a řádně zdokumentovaná má největší přínos. Kompletní dokumentace API se nachází ve složce */apidoc* u zdrojových souborů projektu. Do budoucna plánujeme vystavit tuto dokumentaci na samostatnou stránku, aby byla dostupná online. Ukázka dokumentace je na obrázku 5.4.

Závěr

Úspěšně jsme implementovali službu, která umožňuje zasílat push notifikace na Android a iOS zařízení. Implementace služby pro zasílání push notifikací na mobilní platformy byla však realizována pouze částečně. Při realizaci jsme narazili na několik problémů, které byly natolik zásadní, že jsme se rozhodli navzdory zadání neimplentovat zasílání push notifikací na mobilní platformu Windows Phone. Náš systém je schopný automaticky rozesílat vytvořené notifikace s nastavitelným titulkem a textem. Notifikaci lze nastavit datum a čas odeslání, čímž naplánovat odeslání na budoucí dobu. Původně jsme chtěli přidat možnost opakovaného zasílání, jedná se ovšem o natolik složitou funkcionalitu, že jsme se rozhodli její implementaci odložit do budoucna. Máme totiž v plánu nadále pokračovat s vývojem projektu i nad rámec této diplomové práce, rozšířit funkcionality, přidat další podporované platformy pro zasílání push notifikací a sami službu využívat.

Výstup analýzy konkurenčních řešení nám poskytl přehled o možnostech, které push notifikace mají. Později jsme však zjistili, že pro každou přidanou funkcionalitu je nutné upravit zdrojový kód aplikace přidružené k push notifikacím. Rozhodli jsme se tedy pro vytvoření systému, který bude spíše obecně použitelný na úkor funkcionalit. Tato služba je tedy základem pro další rozšiřování specifické pro konkrétní potřeby koncových uživatelů. Práce je zveřejněna na portálu Github jako open-source (`https://github.com/Degee/pushnotif`). Každý si tedy může podle svých potřeb upravit kód a doplnit požadované funkcionality. Služba také poskytuje API, které je kompletně zdokumentované v anglickém jazyce.

Obecně hodnotíme práci jako úspěšnou s menšími výhradami. Veškeré výhrady jsme si již popsali v sekci problémů a nedostatků 5.7. Z hlediska problematiky byla práce celkem složitá, jedná se o propojení různých technologií a bez důkladné analýzy a návrhu bychom nebyli schopni práci dokončit.

Literatura

[1] Pushwoosh Inc.: *Pushwoosh - Push Notifications Service | Your Apps Can Talk! [online]*. 2016. Dostupné z: https://www.pushwoosh.com/

[2] Lilomi, Inc.: *OneSignal - Multi-platform Push Notification Service*. 2016. Dostupné z: https://www.onesignal.com/

[3] Superblock, LLC.: *Pushover: Simple Notifications for Android, iOS, and Desktop*. 2016. Dostupné z: https://pushover.net/

[4] Carnival, Inc.: *Carnival Mobile | Mobile Marketing and Analytics for brands*. 2016. Dostupné z: http://carnival.io/

[5] Microsoft: *Push Notifications (Windows Phone)*. 2017. Dostupné z: https://msdn.microsoft.com/en-us/library/hh221549.aspx

[6] Google Inc.: *Firebase Cloud Messaging*. 2017. Dostupné z: https://firebase.google.com/docs/cloud-messaging/

[7] Joyent, Inc.: *Node.js*. 2017. Dostupné z: https://nodejs.org/en/

[8] Facebook Inc.: *A JavaScript library for building user interfaces - React*. 2017. Dostupné z: https://facebook.github.io/react/

[9] MongoDB, Inc: *The MongoDB 3.4 Manual*. 2017. Dostupné z: https://docs.mongodb.com/manual/

[10] Rasmussen, E.: *React Redux Universal Hot Example*. 2017. Dostupné z: https://github.com/erikras/react-redux-universal-hot-example

[11] Abramov, D.: *Read Me - Redux*. 2017. Dostupné z: http://redux.js.org/

[12] Reinman, A.: *Nodemailer*. 2017. Dostupné z: https://nodemailer.com/about/

[13] Chernev, I. I.: *Moment.js | Docs*. 2017. Dostupné z: https://momentjs.com/

[14] Khan, B.: *Firebase Cloud Messaging Tutorial for Android*. 2016. Dostupné z: https://www.simplifiedcoding.net/firebase-cloud-messaging-tutorial-android/

[15] inveris OHG: *apiDoc - Inline Documentation for RESTful web APIs*. 2017. Dostupné z: http://apidocjs.com/

[16] Microsoft: *Windows Push Notification Services (WNS)*. 2017. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/apps/hh913756.aspx

Příloha **A**

Seznam použitých zkratek

GUI Graphical User Interface

SDK Software Development Kit

URL Uniform Resource Locator

API Application Programming Interface

REST Representational State Transfer

NPM Node Package Manager

HTTP Hypertext Transfer Protocol

URI Uniform Resource Identifier

JSON JavaScript Object Notation

XMPP Extensible Messaging and Presence Protocol

TDD Test-driven development

XHR XMLHttpRequest

CRUD Create, Read, Update, Delete

ORM Object Relational Mapping

FCM Firebase Cloud Messaging

ACL Access Control List

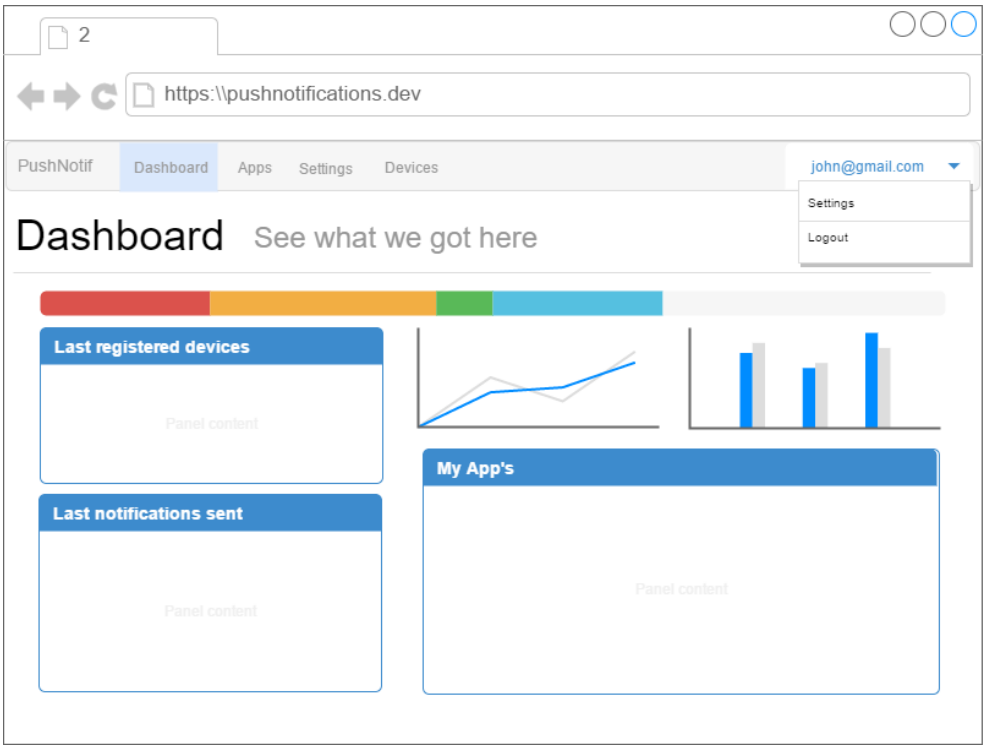
npm node package manager

WNS Windows Notification Service

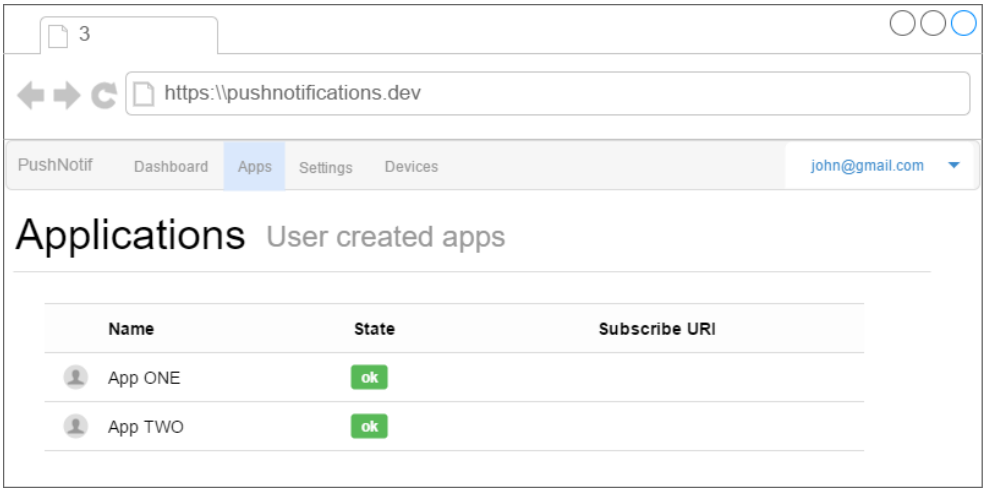
Příloha B

Drátěný model

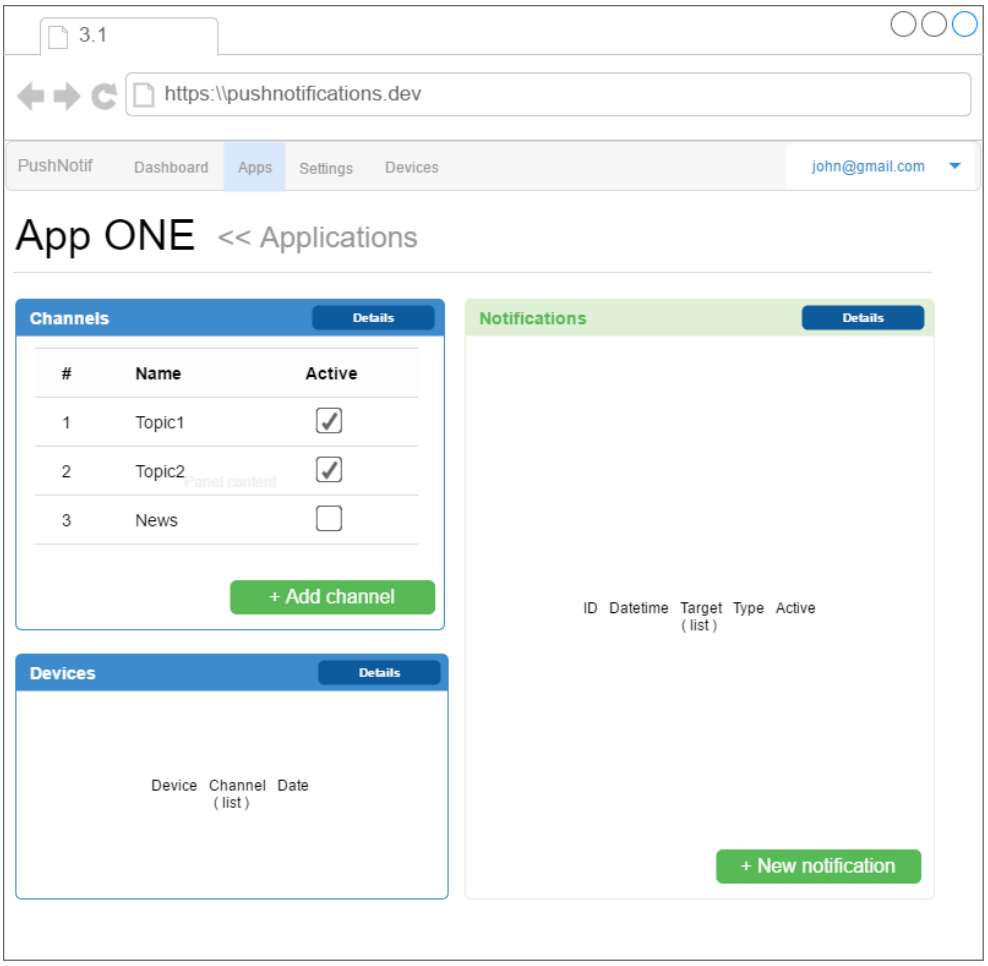
Obrázek B.1: Registrace a přihlášení uživatele



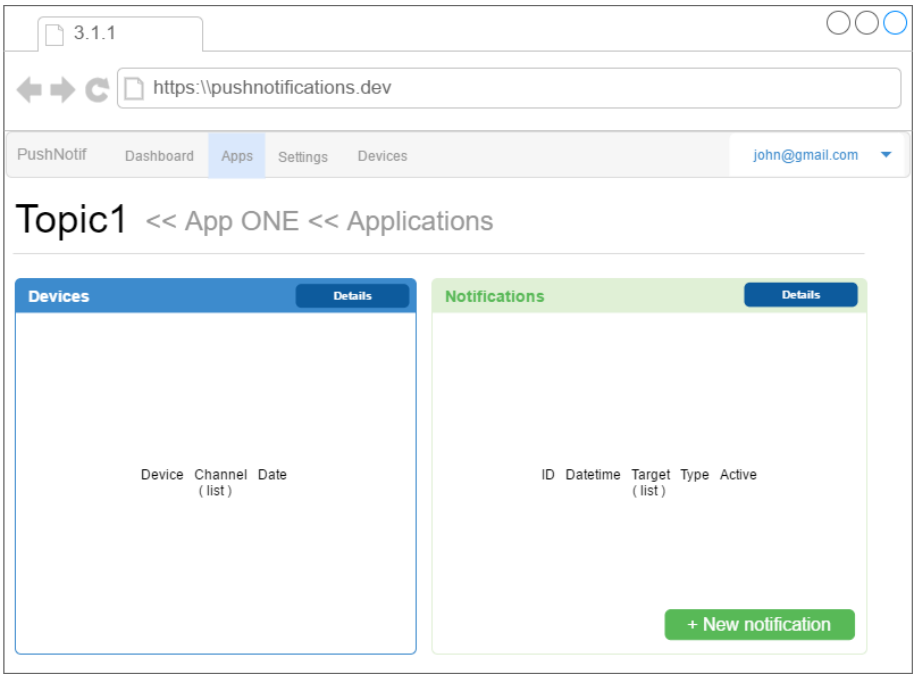
Obrázek B.2: Dashboard se zobrazením statistik



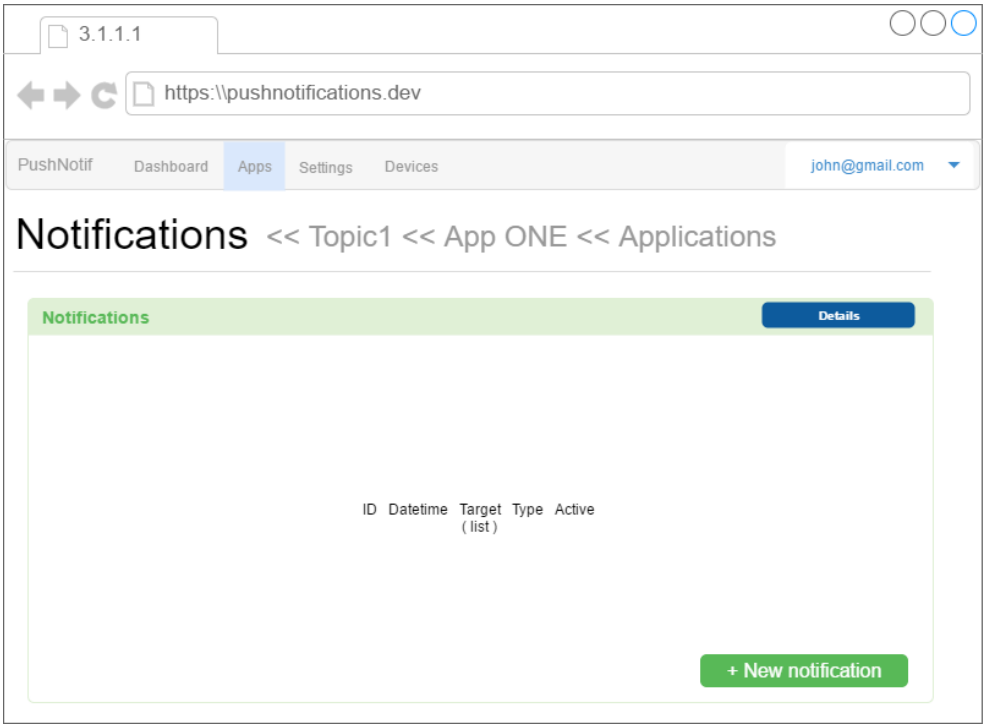
Obrázek B.3: Výpis aplikací



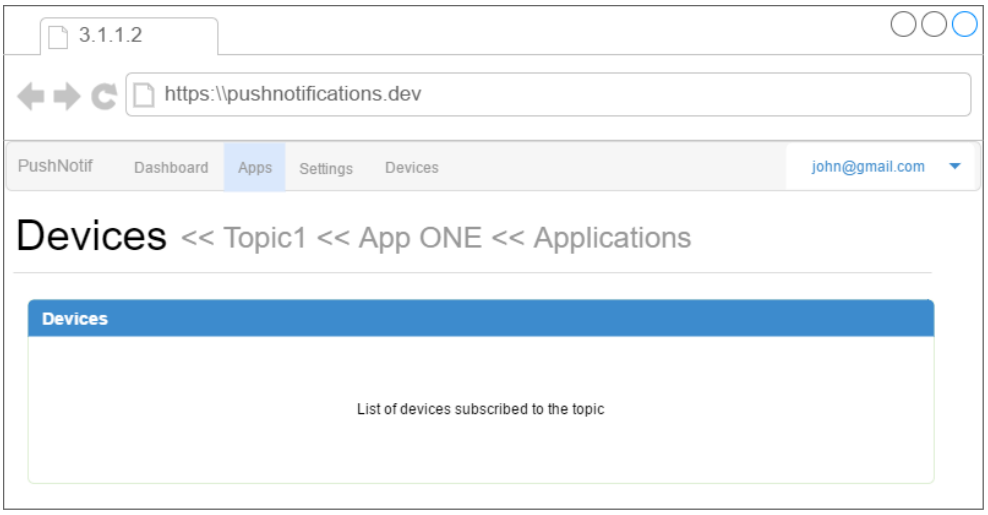
Obrázek B.4: Přehled vybrané aplikace



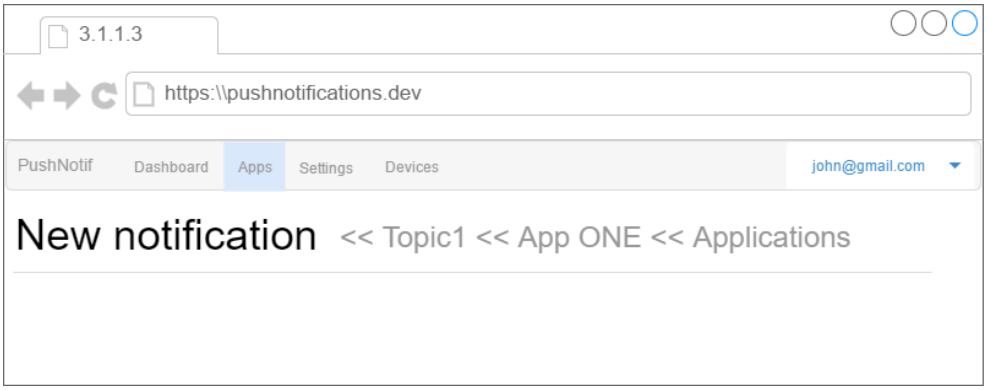
Obrázek B.5: Přehled vybraného kanálu dané aplikace



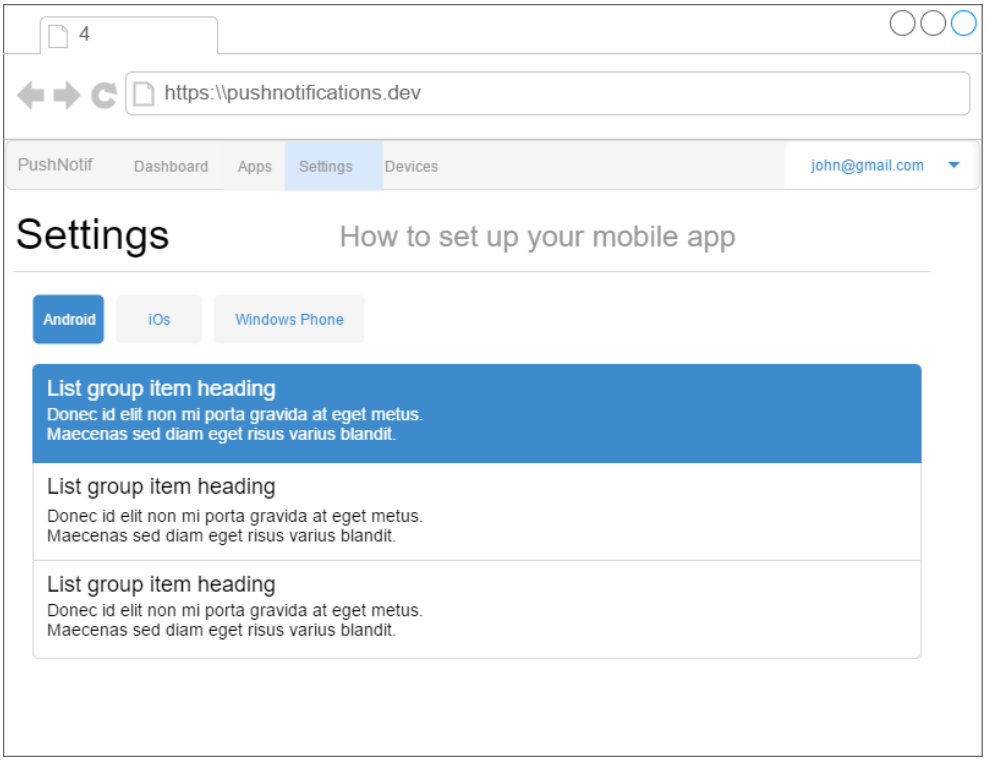
Obrázek B.6: Přehled notifikací daného kanálu dané aplikace



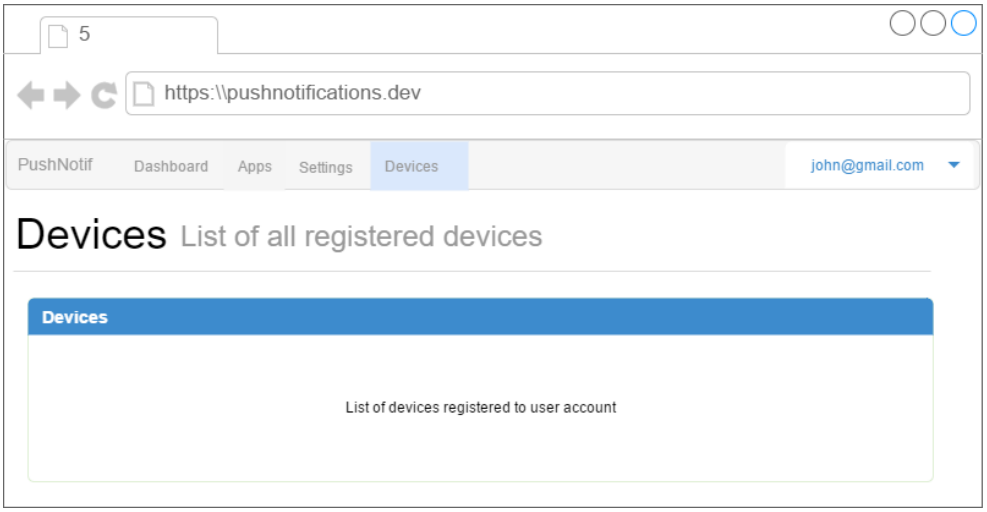
Obrázek B.7: Přehled přihlášených zařízení daného kanálu dané aplikace



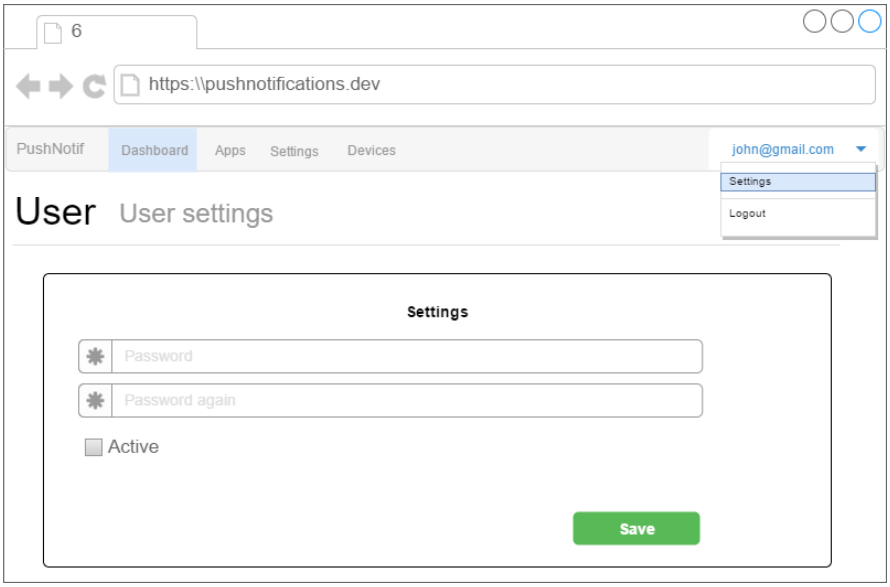
Obrázek B.8: Vytvoření notifikace daného kanálu dané aplikace



Obrázek B.9: Nastavení údajů pro API třetích stran



Obrázek B.10: Výpis všech zařízení přihlášených k našemu uživatelskému účtu



Obrázek B.11: Nastavení uživatelského účtu

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `readme.md` ..... popis projektu a instalační příručka
- `android` ..... spustitelná aplikace pro testování
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF