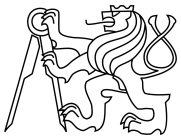
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Webová aplikace pro zasílání Push notifikací na mobilní zařízení
Student:	Jiří Fiala
Vedoucí:	Ing. Marek Žehra
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Navrhněte a implementujte webovou aplikaci k zasílání push notifikací na mobilní zařízení s operačními systémy iOS, Android a Windows Phone. Webová aplikace bude obsahovat aplikační rozhraní, přes které budou komunikovat open-source knihovny pro mobilní operační systémy, tyto knihovny nejsou součástí této práce. Aplikace bude podporovat odklad odesílání notifikací a odesílání notifikací skrze vícero mobilních aplikací v rámci jednoho projektu uživatele. Využijte webový framework ASP .NET a PostgreSQL databázi. Důležitou součástí práce je vytvořit intuitivní uživatelské rozhraní pro co nejjednodušší používání systému.

1. Pomocí metod SI přesně popište funkční požadavky na aplikaci.
2. Navrhněte architekturu aplikace.
3. Návrh implementujte, zdokumentujte a otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 11. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového Inženýrství

Bakalářská práce

Webová aplikace pro zasílaní push notifikací na mobilní zařízení

Jiří Fiala

Vedoucí práce: Ing. Marek Žehra

15. května 2016

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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Fiala, Jiří. *Webová aplikace pro zasílaní push notifikací na mobilní zařízení.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Práce popisuje webovou aplikaci k zasílání push notifikací na mobilní zařízení. Zabývá se návrhem a implementací vlastního aplikačního frameworku, postaveného na návrhovém vzoru MVC za použití platformy ASP.Net. Zaměřuje se na popis nejdůležitějších částí výsledné aplikace a přibližuje čtenáři chování systému, který komunikuje s aplikacemi na mobilních zařízeních. Aplikace shromažďuje zaslané údaje ze zařízení, jenž umožňují segmentaci a filtrování doručovaných notifikací z webového rozhraní.

Klíčová slova Push notifikace, Vývoj mobilních aplikací .NET Framework, MVC, Notifikační centrum, API

Abstract

The thesis describes the Web application to send push notifications to mobile devices. It deals with the design and implementation of custom application framework built on the MVC design pattern using ASP.Net platform. It is focused on the description of the most important parts of the resulting application and brings readers the behavior of the system, which communicates with other applications on mobile devices.The application collects data sent from devices that allow segmentation and filtering of push notifications delivered by the web interface.

Keywords Push Notifications, Mobile Development, .NET Framework, MVC, Notification Center, API

Obsah

Seznam obrázků

Úvod

Mnoho aplikací určených pouze pro obrazovky stolních počítačů se přepisuje, aby fungovaly na malých mobilních zařízeních. Průměrná doba strávená na telefonu, či jiném chytrém zařízení, za účelem vyřizování korespondence nebo prací na jednoduchých kancelářských rutinách, se stále zvyšuje. Tím se začíná měnit i způsob doručování obsahu uživatelům. Informace o novinkách se nativně zobrazují na ploše, či ve vyhraněné oznamovací oblasti operačního systému. Tento způsob, kterým vývojáři mobilních aplikací dnes komunikují s koncovým uživatelem se nazývá push notifikace.

Push notifikace je způsob komunikace po internetu, kdy komunikaci neiniciuje klient, ale server. Je opakem tzv. pull techniky, kdy žádost o komunikaci zahajuje klient. Například poštovní klient na počítači kontroluje každou minutu, jestli na serveru příchozí pošty není nová zpráva. Pokud tam je, stáhne obsah a ihned upozorní uživatele o její existenci.[1]

Možnost zasílat push notifikace uživateli prostřednictvím aplikace, kterou má nainstalovanou na svém telefonu je tedy velmi snadnou příležitostí, jak uživatele ihned přímo oslovit, upozornit, nebo mu něco nabídnout.

Práce se zaměřuje na vytvoření flexibilního aplikačního frameworku postaveného na platformě ASP.Net. Bude kladen důraz na vytvoření konvencí v celé aplikaci, aby byla jednoduše rozšiřitelná a upravitelná.

V dnešní době můžeme narazit na několik serverů sloužících k odesílání notifikací. Většina je ale placená, či natolik složitá, že odradí začátečníky od využívání funkcí které push notifikace nabízejí.

Webová aplikace nese název NOTIFIT.

Kapitola 1

Cíl práce

Potřeba zkontaktovat někoho, či něco sdělit, je v dnešní době nezbytností pro většinu obyvatel světa. Tato nutnost je vázána především na rychlost a formu doručované zprávy. Práce se bude daným problémem zabývat v oblasti push notifikací na mobilních zařízeních.

Push notifikace jsou krátkým sdělením, která uživatel dostává na upozornění od mobilní aplikace na aktualizaci nového stavu.

Kladu si za cíl vytvořit přívětivé webové rozhraní, umožňující programátorům mobilních aplikací využívat push notifikace v co nejjednodušší formě. Budu se zabývat strukturou těchto notifikací a navrhnu webové rozhraní umožňující odesílat tyto zprávy na systémy Windows, iOS, a Android. Doplním systém o možnost plánování času odeslání notifikace a nastavení periodicity pro opakování stejných zpráv. Uživatel bude mít možnost cílit notifikace určitým zařízením, pomocí selekce uložených parametrů identifikující jednotlivá zařízení a skládáním logických výrazů pro zacílení do specifických segmentů. Uživatel dostane možnost jednoduše posílat notifikace napříč aplikacemi seskupenými do projektů a možnost upravit oznámení pro každou aplikaci zvlášť.

Na aplikaci jsou kladeny tyto požadavky:

- Funkční

  Možnost odesílat notifikace na zařízení s operačním systémem iOS, Windows, Android.

  Segmentovat zařízení na které se push notifikace odešle.

  Zobrazit informace o odeslané notifikaci.

  Zakládání projektů a přiřazování aplikací k nim.

- Nefunkční

  Přívětivé uživatelské prostředí.

  Udržitelnost systému.

  Rozšiřitelnost a modifikovatelnost.

Účelem aplikace je usnadnit vývojářům mobilních aplikací implementování push notifikací do svých programů. Tím mohou zefektivnit interaktivitu s uživateli jejich aplikací, a držet je v obraze ohledně novinek a aktualizací. Systém, který jim toto umožní, se skládá z několika vrstev:

- Webové rozhraní aplikace
- Mobilní knihovny
- REST API serveru

1.1 Webové rozhraní aplikace

Jádrem celého systému NOTIFIT je webová aplikace. Skládá se z uživatelského rozhraní, ve kterém si vývojář mobilní aplikace spravuje své projekty a z démona[1], který obsluhuje odesílání push notifikací.

Ve webové aplikaci lze mimo spravování projektů, aplikací a vytváření notifikací také sledovat metriky o zařízeních registrovaných k aplikacím.

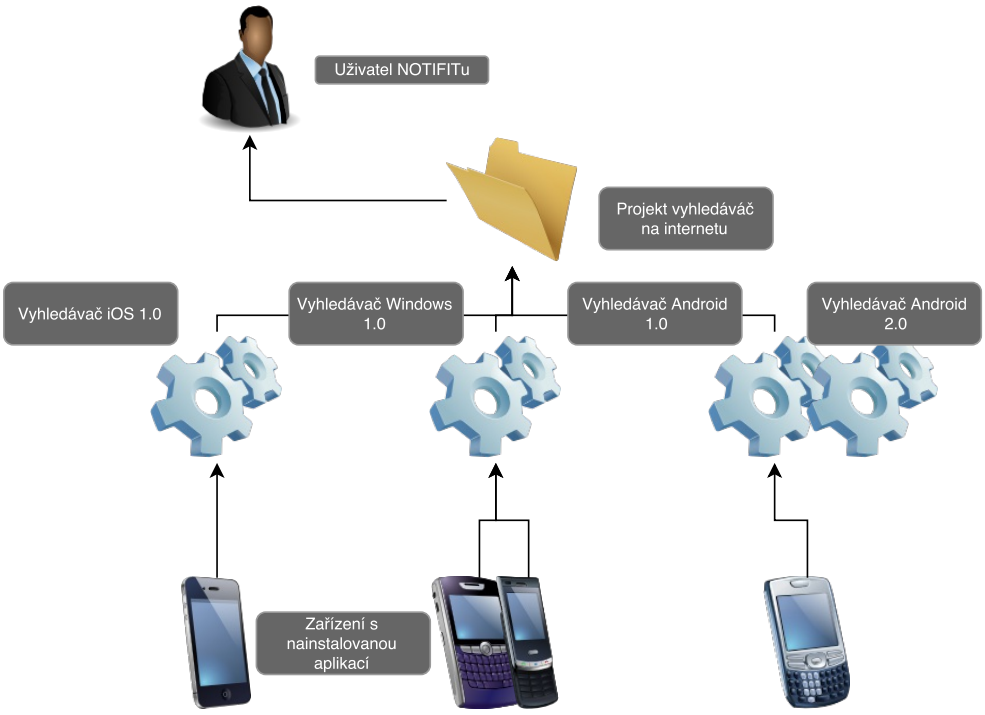
1.1.1 Rychlý start

NOTIFIT se snaží usnadnit mobilním vývojářům celý proces odesílání notifikací a klade si za cíl odstranit překážku v potřebě vlastního serveru pro správu tokenů zařízení[2]. Workflow registrace a zprovoznění služby klade důraz na to, aby bylo rychlé a hladké.

Nový uživatel se po registraci a přihlášení dostane na úvodní obrazovku, kde má k dispozici administrační menu a vidí zde přehled rozpracovaných, čekajících či dokončených notifikací. První nezbytná věc, je založení nového projektu, který funguje jako logický celek, zahrnující v sobě mobilní aplikace, které spolu souvisí např. verzemi, funkčností, nebo jsou stejné až na operační systém, na kterém běží, viz Obrázek 1.1. V dalším kroku přidá vývojář do projektu aplikace, které má hotové a jsou registrované v některém z internetových obchodů. V závislosti na operačním systému, jsou u aplikací vyžadovány další informace (Apple APNs certifikát, Windows aplikační tajný klíč, GCM aplikační klíč). Jde o nezbytné údaje sloužící k autentizaci NOTIFITu k webovým službám jednotlivých systémů. Po tomto kroku, je uživateli již dostupný Projektový a Aplikační token, umožňující inicializaci knihoven v mobilních aplikacích.

[1] Démon je označení programu, který se inicializuje při startu systému a vyčkává v nečinnosti na určitou událost, kterou na pozadí zpracuje bez interakce s uživatelem.

[2] Tokeny zařízení slouží k identifikaci zařízení, na které je odesílána notifikace. Přidělují je cloudové služby jednotlivých poskytovatelů notifikačních služeb.



Obrázek 1.1: Struktura projektů a aplikací NOTIFITu

1.2 Mobilní knihovny

Při inicializaci mobilní knihovny je získán tzv. notifikační token unikátní pro každé zařízení. Tyto tokeny poskytují jednotlivé webové služby, specifické pro různé operační systémy mobilních zařízení: jmenovitě Apple Push Notification Service, Google Cloud Messaging a Windows Push Notification Services. V současné době jsou v prvotní fázi zprovozněny knihovny pro operační systémy Windows 10, Apple iOS a Android.

1.2.1 Inicializace

Aby vše, začalo fungovat stačí do projektu mobilní aplikace vložit jediný řádek kódu, viz Kód 1.1 z knihovny pro Windows. Pokud uživatel mobilní aplikace schválí zasílání Push notifikací, knihovna zprostředkuje dotaz na tuto službu a pokusí se obdržet komunikační token. Knihovna je vždy svázána s projektovým a aplikačním tokenem, který si vývojář založil ve webové aplikaci. Při prvním spuštění aplikace se odešle na REST server NOTIFITu požadavek na registraci nového zařízení. V požadavku se nachází Notifikační token a několik dalších systémových proměnných. Jako odpověď ze serveru obdrží zařízení jedinečný identifikátor, který je označen jako komunikační token. Knihovna nadále už tedy nepoužívá projektový a aplikační token, ale posílá v hlavičkách

požadavků na NOTIFIT tento komunikační token.

Kód 1.1: Inicializace knihovny NOTIFIT

```
var NOTIFIT = new NOTIFIT("http://NOTIFIT.io/",
   "93c5752c-aaee-e511-942b-00155d000710",
   "932facc9-3cf0-e511-942b-00155d000710");
```

1.2.2 Uživatelské proměnné

Knihovna umožňuje vývojáři mobilních aplikací také definovat vlastní proměnné typu klíč - hodnota. Vývojář tedy může ukládat například hodnoty typu: uživatelské jméno, email, hodnocení a jiné. Tyto hodnoty se odesílají na REST API až tehdy, kdy stanoví programátor, zavoláním funkce NOTIFIT knihovny pro uložení hodnoty ke klíči. Pokud hodnota pro daný klíč v systému již existuje, vytvoří se nová hodnota s časovou značkou, kdy vznikla. Pro neexistující klíč je automaticky vytvořen nový a hodnota se k němu okamžitě připojí. Je možné tedy sestavovat metriky a sledovat změny v závislosti na čase.

1.2.3 Obvyklý běh knihovny

Po obdržení komunikačního tokenu jedinečného pro každé zařízení, posílá knihovna již jen PUT požadavek na REST API. Při každém spuštění se automaticky aktualizují všechny systémové hodnoty včetně Notifikačního tokenu. Tokeny mají totiž jen omezenou platnost v závislosti na operačním systému.

1.3 REST API

Webová služba NOTIFITu pro vnější komunikaci se zařízeními je rozdělená na dvě části:

- Služba pracující s aplikačními knihovnami
- Služba kopírující funkcionalitu webového rozhraní NOTIFITu

1.3.1 Služba pracující s aplikačními knihovnami

K propojení aplikačních knihoven z databází NOTIFITu je potřeba služba která zpracuje Notifikační tokeny, systémové proměnné a uživatelem definované klíče a hodnoty. Tato služba pracuje pouze s open source knihovnami NOTIFITu. Slouží k registraci zařízení a spárování s projekty a aplikacemi vývojářů.

1.3.2 Služba kopírující funkcionalitu webového rozhraní NOTIFITu

V závislosti na současný trend ovládat velkou část aplikací pouze pomocí mobilních zařízení, je nezbytné vystavit službu pro NOTIFIT aplikace, přizpůsobené pro mobilní systémy. Služba musí zajišťovat CRUD prostředky pro správu uživatelských projektů a aplikací. V neposlední řadě také musí umožnit uživatelům ve zjednodušené formě odesílat notifikace a zobrazit metrické údaje o uložených zařízeních.

Kapitola 2

Analýza a návrh

2.1 Možnosti řešení

V dnešní době je k dispozici několik nástrojů pro tvorbu dynamické webové aplikace. Základním kamenem aplikace je její platforma. Ta se v tomto případě skládá ze čtyř nezbytných částí:

- Operační systém
- Webový server
- Databázový systém
- Programovací jazyk

2.1.1 Operační systém

Operační systém je základní programové vybavení počítače, umožnující ovládat počítač ve stabilním aplikačním rozhraní. Pro hostování dynamických webů lze užít například některou z verzí Windows Server od firmy Microsoft. Tento operační systém není zdarma a musí být licencován. Jinou možností by bylo použít některou z distribucí Linuxu a web postavit na velmi známé platformě LAMP (Linux Apache, MySQL, Php). NOTIFIT je však psán v ASP.Net, který je schopen nativně fungovat jen na zařízeních s operačním systémem Windows.

2.1.2 Webový server

Další důležitou složkou platformy je webový server. Ten běží spuštěný jako démon na serverovém počítači, který zpracovává převážně požadavky pro protokol http, https a jako odpověď na tyto dotazy odesílá výsledné HTML, či obsah v jiné formě (JSON, XML) zpět klientovi. Pro Windows Server je součástí webový server IIS neboli Internet Information Sevices od firmy Microsoft.

Nejznámější webové servery jsou: Apache, IIS, Nginx, GWS. které obsluhují zhruba 90% všech webových stránek. [2].

2.1.3 Databázový systém

Je klíčovou komponentou pro manipulaci, ukládání a správu dat aplikace. Existuje několik databázových modelů. Nejužívanější bývají v dnešní době relační, objektové a objektově-relační databáze. Z hlediska stability vytvořené dlouholetým vývojem a známostí jazyka SQL dávám přednost relační databázi. Mezi nejznámější relační databáze patří MySQL, Oracle, MSSQL, PostgreSQL a jiné.

2.1.4 Programovací jazyk

Na tvorbu dynamického webu je možné užít nespočet jazyků. Patří sem takové jako jsou C#, Java, JavaScript, PHP, a další. Pro jaký jazyk se rozhodnout určuje především zvolený operační systém serveru.

2.2 Zvolené řešení

Zvolenou platformou pro mé řešení se stalo na základě zadání práce ASP.NET instalované na operačním systému Microsoft Windows Server 2012 R2, který mi umožní použít požadované technologie. Jako webový server je užitý IIS[3], dodávaný jako instalovatelná součást Windows serveru.

Pro tuto specifickou platformu je zapotřebí užít programovací jazyk C#. Ten díky částečné kompilaci zdrojového kódu na webovém serveru zaručuje vysokou rychlost vyřízení požadavků přijatých od klienta. Frontend aplikace je psán za pomoci HTML, Javascriptu, a CSS stylů. HTML je generováno z CSHTML souborů, kde je použit šablonovací značkovací jazyk Razor. Díky němu jsou i komplexnější pohledy přehlednější v době implementace a usnadňují práci s formátováním výsledného HTML.

.NET Framework také v poslední době velmi podporuje softwarovou architekturu MVC [4] a vytvořil pro ni webový a serverový framework, který užívám popořadě pro webovou front-end část a pro RESTful API server, komunikující s jednotlivými mobilními zařízeními.

Veškerá data se budou ukládat do relační databáze PostgreSQL, šířené pod licencí MIT[5] jako open source. Databáze je tedy zdarma a bez omezení na distribuci. Nicméně vzhledem k tomu, že NOTIFIT vytváří a následně přistupuje k databázi pomocí Entity Frameworku (dále jen EF), je potřeba zvolit vhodný konektor pro přístup k databázovému serveru PostgreSQL. EF totiž primárně pracuje pouze s MSSQL.

[3] Internet Information Services

[4] Architektura pro oddělení logiky domény, vykreslování, a logiky zpracování dat.[3]

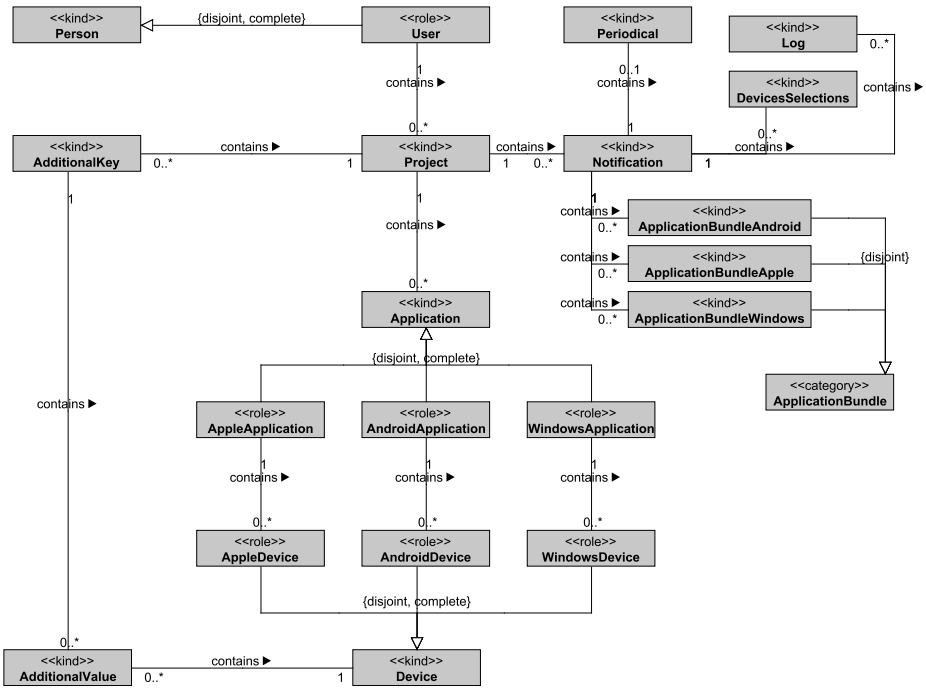
[5] Plné znění licence na `https://opensource.org/licenses/MIT`

Jako konektor s notifikačními centry bude využívána open source knihovna PushSharp. Knihovna umožňuje zasílat notifikace na zařízení s operačním systémem iOS, Android, Windows Phone, Windows 8 a další. Výhodou knihovny je, že odpovědi z center získané po odeslání notifikace nejsou vždy naprosto triviálního formátu, PushSharp však poskytuje jednoduché API, jak s těmito odpověďmi nakládat a řešit případné chyby.

Samotné GUI webové aplikace pak stojí na responsivním javascriptovém frameworku Twitter Bootstrap s použitím open source šablony AdminLte vytvářející přehledné uspořádání webu pro administrační internetové stránky.

2.3 Doménový model

Doména projektu by neměla být ze začátku nijak složitá. Do budoucna se ale počítá s přidáváním dalších funkcí. Je tedy potřebné navrhnout systém tak, aby na sobě nebyly jednotlivé části systému závislé, a byla podporovaná jednoduchá rozšířitelnost aplikace, viz Obrázek 2.1.



Obrázek 2.1: Objektový model aplikace NOTIFIT

Vzhledem k povaze aplikace, kdy uživatel ukládá přístupy ke svým aplikacím, musí být téměř všechny akce prováděny po přihlášení uživatele. Nejvyšší entitou aplikace je proto uživatel. Po uživateli budou požadovány základní

přihlašovací údaje jako je uživatelské jméno, emailová adresa a heslo. Pro personalizaci účtu je po uživateli vyžadováno jméno a příjmení. Časem by měla přibýt i služba pro autentizování pomocí Facebookového a Google účtu. U těchto uživatelů bude email shodný s uživatelským jménem.

Uživatel může po přihlášení zakládat projekty. Entitou projekt je myšlen logický celek, do kterého uživatel bude přidávat aplikace, které spolu souvisejí. Projekt je tedy jen jakýsi kontejner držící aplikace a obsahuje kolekci s názvy systémových a uživatelsky definovaných klíčů. Projekt musí mít název. Nepovinně mu může být zadán název firmy, pod kterou spadá a účel pro lepší orientaci mezi dalšími projekty.

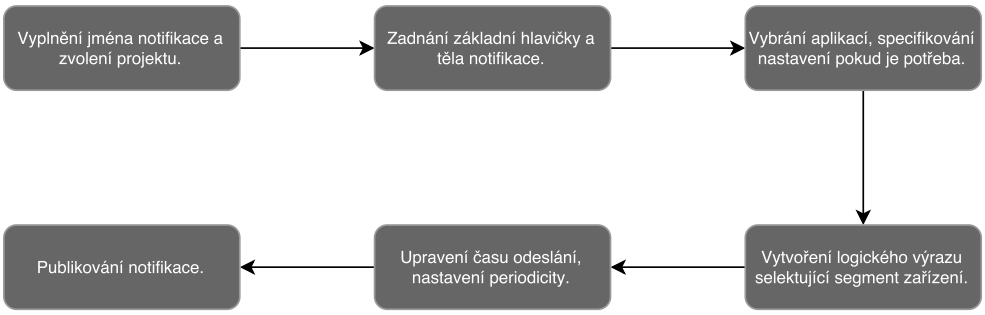
Aplikace mohou být tří typů podle druhu operačního systémů: Windows OS, Apple iOS, Google Android. Typ operačního systému aplikace by měl být držen v databázi jen jako číslo. Tím docílíme jednoduché udržitelnosti a rozšiřitelnosti, bez potřeby kopírovat kód a vyhneme se složitějšímu porovnávání řetězců při třídění. Entita aplikace nese potřebné informace k autentizaci služeb push notifikací a číslo svého typu, stanoveného enumerátorem aplikace. Dále bude obsahovat název a datum vytvoření. Ke každé aplikaci se váže n zařízení, které se registrují pomocí aplikačních knihoven přes webovou službu.

Zařízení již v sobě musí obsahovat notifikační token, který je nezbytný pro identifikaci zařízení u notifikačních služeb. Je také nutné, aby se dalo zařízení jasně identifikovat a nedocházelo k vytváření nových zařízení v databázi NOTIFITu. Nabízelo by se použít notifikační tokeny, nicméně ty mají jen omezenou životnost a často se mění v závislosti na systému zařízení. Proto bude k zařízení přiřazen jedinečný a neměnný komunikační token vytvořený NOTIFITem. Díky němu se může zařízení autentizovat v REST službě NOTIFITu a upravovat o sobě údaje. Aplikace by měla umět také monitorovat dobu běhu aplikací na každém zařízení. Proto se na zařízení bude vázat entita zaznamenávající různé stavy, do kterých aplikace na zařízení přešla a bude zaznamenán její čas pořízení. Zařízení k sobě váže několik přídavných uložených hodnot.

Přidané hodnoty se váží vždy na specifický klíč, registrovaný uvnitř projektu. Při změně hodnoty by se měl vytvořit nový záznam a uložit předchozí hodnotu do historie zařízení. Klíče i hodnoty se budou ukládat jako textová pole. Do budoucnosti by se mohly hodnoty rozšířit o datové typy, aby se mohla nabídnout lepší pružnost kontroly formátu ukládaných parametrů.

Samotné entity notifikací se váží na projekt. Jak již bylo zmíněno výše, projekt je logický celek, v rámci kterého se notifikace odesílají. Workflow vytváření notifikace znázorňuje obrázek 2.2. Notifikace v sobě musí nést časové informace o tom, kdy byla vytvořena, zařazena do fronty na zpracování, a kdy by se měla reálně zpracovat. Dále musí obsahovat základní zprávu, sestávající se z hlavičky a těla. Pro případ, že bude notifikace periodická, bude pro ni vytvořena speciální entita s parametry popisující kolikrát a po jakých intervalech se notifikace odešle. Uživatel by měl být schopný definovat také jednotlivé nastavení push notifikací pro každou aplikaci zvlášť. K tomu po-

slouží aplikační balíčky. Balíček bude vázán na konkrétní aplikaci a bude mít v sobě uložené specifické nastavení. Pro případ, že v tomto nastavení nebude přepsána hlavička a tělo zprávy, bude automaticky doplněna základními hodnotami vyplněnými v předchozím kroku. Nespecifikované nastavení pro ostatní aplikace bude rozšířeno při odesílání notifikace výchozím nastavením NOTIFITu. Uživatel bude mít možnost posílat notifikace jen konkrétnímu segmentu zařízení. Za tímto účelem může vytvořit logický výraz, definující, jak se odpovídající zařízení vyberou. Výraz je tvořen kolekcí objektů, popisující jednotlivé části výběrů. Tato kolekce se následně serializuje do XML řetězce a uloží se uvnitř notifikace. Notifikace budou moci být zveřejněné nebo nezveřejněné. Všechny zveřejněné notifikace budou zařazovány do fronty ke zpracování. Je nutné sledovat i stav, jestli je notifikace již zpracovaná nebo čekající. Samotná tvorba notifikace bude rozdělena do několika kroků. Při tomto procesu bude nutno rozpoznat konkrétní rozpracovanou notifikaci pomocí identifikátoru, který se bude ukládat v session a controllery ji pomocí něj budou moci vyhledat v databázi. Tento proces bude zároveň ukládat nejvyšší dosažený stav tvorby a dovolí po přerušení pokračovat od posledního rozpracovaného bodu. Nedokončené nebo rozpracované notifikace bude možné smazat. Odeslané notifikace bude uživatel moci smazat také, ale dojde jen ke zneviditelnění pro uživatele nastavením stavu smazáno v databázi. Takto se bude možno později dostat v backendu k důležitým metrickým datům, které by mohly pomoci dalšímu vývoji aplikace. Při zpracování notifikací bude vytvářen také log, vázaný ke konkrétní notifikaci,umožňující sledovat případné pády a stavy, a následně informovat uživatele o výsledcích zpracování.



Obrázek 2.2: Průběh vytváření notifikace

2.4 Notifikační služby

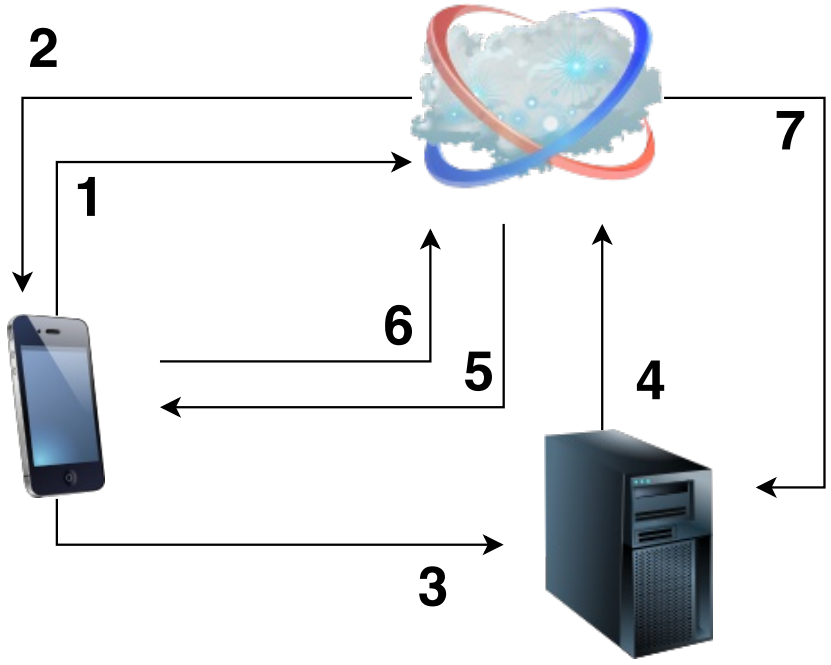
Od vytvoření zprávy, která má být zobrazena na zařízení, po doručení a získání odpovědi o úspěchu či neúspěchu, vede relativně dlouhá cesta. Každý operační systém, pro který bude NOTIFIT fungovat, zveřejňuje vlastní službu, které zařízení naslouchají a přijímají od nich čekající push notifikace.

- Windows Push Notification Services (dále jen WNS) je služba umožňující vývojářům třetí strany rozesílat toast, tile, badge a raw aktualizace z vlastní cloudové služby. Tato služba vytváří mechanismus k doručení těchto novinek výkonnou a efektivní cestou.

- Google Cloud Messaging (dále jen GCM) umožňuje odesílat data z vlastního serveru do uživatelských zařízení a získávat odpovědi nazpět na stejném připojení. GCM služba řeší všechny aspekty řazení zpráv a doručuje obsah do klientských aplikací běžících na cílovém zařízení.

- Apple Push Notification Sevice (dále jen APNs) je prostřední článek k odesílání vzdálených notifikací. Je to robustní a velmi efektivní služba k propagování informací na iOS (a nepřímo také na watchOS), tvOS a OS X zařízení. Každé zařízení uskutečňuje akreditované a šifrované IP spojení s APNs a obdržuje notifikace přes toto přetrvávající spojení. Jestliže notifikace dorazí právě tehdy když zrovna aplikace neběží, zařízení upozorní uživatele, že má aplikace zprávu čekající na zobrazení.

Všechny tyto služby fungují převážně na stejném principu a jeho workflow lze vysvětlit pomocí sedmi kroků, viz Obrázek 2.3.



Obrázek 2.3: Workflow komunikace s notifikačním centrem, zařízením a serverem třetí strany

1. V zařízení se vytvoří notifikační kanál. Zde se většinou jedná o zavolání systémové funkce. Pomocí ní se požádá notifikační služba o token, dle kterého bude zařízení a aplikace rozpoznána.

2. Pokud zařízení obdrželo notifikační token, naváže komunikaci se serverem třetí strany a pomocí smluvené metody[6] mu tento token odešle.

3. Server třetí strany obdrží token a zpracuje ho například uložením do databáze. Ve chvíli, kdy je zapotřebí vytvořit notifikaci, vytvoří server payload[7] notifikace.

4. V dalším kroku se autentizuje s notifikační službou, a po nově vytvořeném kanále odešle čerstvou zprávu.

5. Pokud došlo ke správné autentizaci k aplikaci notifikačního serveru, zpracuje notifikační server payload a pokusí se zprávu odeslat na zařízení s obdrženým tokenem.

6. Jestliže notifikace úspěšně dorazila na zařízení, odešle potvrzení o přijetí notifikačnímu serveru.

7. Odpověď je následně přeposlána serveru třetí strany, aby byl informován o úspěchu či nezdaru a jeho příčině. Například při neplatném tokenu může server třetí strany uloženou položku zařízení vymazat z databáze, protože nemá cenu pro toto zařízení dále cokoliv odesílat.

2.4.1 WNS

Nabízí k dispozici čtyři různé typy notifikací.

- Tiles - reprezentativní notifikace zobrazující aktuální informace v nabídce Start na dlaždicích.

- Badges - slouží k zobrazení ikony nebo čísla na tiles - dlaždicích.

- Toast - je zpráva s vyskakovacím UI prvkem nazývaným toast nebo také jinak banner. Zpráva může být zobrazena i když konkrétní aplikace není právě spuštěná.

- Raw - jsou krátké notifikace obecného účelu. Jsou striktně instruktážní a neobsahují žádný UI prvek.

[6] Odeslání dat na jiný server může být řešeno například přes protokol HTTP, kdy bude zařízení komunikovat s REST API serveru.

[7] Payload je objekt s uloženou strukturou notifikace. Dle služby může být ve formátu JSON, či XML.

NOTIFIT nabízí odesílání toast notifikací. Notifikace je odesílána ve formě XML na server WNS s tokenem zařízení, pro které se zpráva doručuje. Struktura této notifikace je popsána kódem 2.1 níže. Odesílání push notifikací vyžaduje přihlášení k WNS. Toto přihlášení je možné pouze pro uživatele, mající přístup do webové sekce Microsoft Developer Center. Autorizace pak vyžaduje tři přihlašovací hodnoty pro každou aplikaci. Jedná se o Package SID, Package Name a Client Secret.

Kód 2.1: Struktura toast notifikace

```
<toast launch? duration? activationType? scenario? >
    <visual version? lang? baseUri? addImageQuery? >
        <binding template? lang? baseUri?
           addImageQuery? >
            <text lang? >content</text>
            <text />
            <image src placement? alt? addImageQuery?
               hint-crop? />
        </binding>
    </visual>
    <audio src? loop? silent? />
    <actions>
    </actions>
</toast>
```

XML schéma notifikace, viz Kód 2.1 má kořenový element s názvem `toast`. Pro něj lze specifikovat několik nepovinných atributů, jejichž výčet následuje dále. `Launch` je řetězec, předávaný aplikaci po aktivaci notifikace. V závislosti na `activationType`, může obdržet aplikace tuto hodnotu buď na popředí, nebo na vlákně na pozadí, protokolem nebo systémem. `Duration` může nabývat hodnot *short* nebo *long* pro zobrazení notifikace buď na standardních sedm či dvacet pět sekund. Při specifikaci atributu `scenario` je možné vynutit dlouho přetrvávající notifikaci například pro příchozí hovor nebo upomínku, která nezmizí, než ji uživatel potvrdí nebo zavře.

Element `visual` má atribut `version`, který je nepovinný a je již zastaralý, proto není nutné ho vyplňovat. Atribut `lang` specifikuje jazykové preference těla XML pomocí jazykového tagu. Systémově je nastaven na jazyk uživatelova zařízení. `BaseUri` nese základní URI zdroje obrázků pro kombinování s relativními adresami použitých u obrázků. `AddImageQuery` je hodnota typu boolean říkající, zda server poskytující použité obrázky, umí zpracovávat dotazy na obrázky k docílení nastavení velikosti, kontrastu a jazyka.

Elementu `binding` můžeme nastavit atribut `template` pro získání požadovaného efektu zobrazené notifikace. Ostatní atributy jsou stejné jako u elementu `visual`. Potomky mohou být elementy `text` a `image`. Element `image` má povinný atribut `src`, kde je uložena adresa odkazu. Mimo jiné je možné nastavit způsob umístění obrázku. Pozice může být buď uvnitř notifikace nebo

místo zobrazovaného loga aplikace. Pro každé umístění může být specifikován nanejvýš jeden obrázek. `Hint-crop` atribut umožní oříznout obrázek do tvaru kruhu, například pro zobrazení uživatelů. Poslední atribut `alt` je pro zobrazení popisku pro uživatele se zapnutými asistenčními technologiemi.

Element `audio` umožňuje upravit přehrávaný zvuk při zobrazení notifikace. `Src` atribut je identifikátor zvuku systému Windows. Příchozím hovorům a upomínkám je možné nastavit atribut `loop` pro neustálé přehrávání zvuku. Atribut `silent` vypne zvuky notifikace. Elementy akcí slouží k vytvoření rychlých formulářů.[4]

2.4.2 GCM

Server GCM přijímá notifikace ve formě JSON balíčku. Autentizace probíhá na základě GCM aplikačního tokenu. Ten získá vývojář po zaregistrování aplikace v backendu Google Play Developer Console. Struktura payloadu notifikace je znázorněna níže, viz Kód 2.2.

Kód 2.2: Struktura GCM push notifikace

```
{
  "registration_ids" :
     ["bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
     ...]
  "priority" : "normal",
  "collapse_key" : "Update",
  "restricted_package_name" : "demo",
  "delay_while_idle" : true,
  "dry_run" : true,
  "time_to_live" : 3,
  "data" : {
    "message" : "This weeks edition is now available.",
    "title" : "NewsMagazine.com"
  },
}
```

Payload se skládá z nastavení a těla notifikace. Parametr nastavení `registration_ids` je pole naplněné všemi notifikačními tokeny, kterým je zpráva určena. `Priority` může být nastaveno na hodnotu *normal* nebo *high*. Při normální prioritě dává GCM služba pozor na stav baterie uživatelova zařízení. Pokud je tedy uživatelova baterie v kritickém stavu, může mu být zpráva doručena s nespecifikovaným zdržením.

Parametr `collapse_key` zajistí, aby nedošlo k hromadění zpráv stejného typu. Pokud tedy přijde několik zpráv se stejnou hodnotou `collapse_key`, zobrazí se pouze ta poslední. Parametr je zamýšlen k tomu, aby po připojení zařízení do sítě, kdy zařízení nebylo delší čas připojeno ke službě GCM, nedošlo

k zahlcení novinkami. To může nastat pokud byl nastaven parametr `delay_while_idle`.

Pokud je parametr nastavení `delay_while_idle` nastaven na *true*, indikuje to, že zprávy nemají být poslány dřív, než dojde k aktivaci zařízení. V základu je tato hodnota nastavena na *false*.

`Restricted_package_name` je název balíčku aplikace, který se musí shodovat s názvem aplikace, ve které je uložen registrační token, aby mohlo dojít k zobrazení notifikace.

`Dry_run` je určen k testování nastavení serveru. Umožňuje posílat notifikace na server GCM aniž by došlo k tomu, že by je GCM přeposílala na jednotlivá zařízení.

Parametr `time_to_live` specifikuje jak dlouho v sekundách má být zpráva držena v GCM úložišti, jestliže je zařízení offline. Maximum této hodnoty jsou čtyři týdny, ty jsou zároveň i základní nastavenou hodnotou.

V sekci `data` je možné specifikovat vlastní klíče a hodnoty, které budou přístupné v aplikaci po přijetí notifikace. Neměli by se ale odesílat žádné citlivé uživatelské informace.[5]

2.4.3 APNs

Přihlášení k serveru APNs probíhá přes šifrované připojení, pomocí certifikátu podepsaného certifikační autoritou společnosti Apple Inc.. Certifikát je nutné spárovat s Apple aplikací v portálu, který je určen vývojářům. Na serveru se pak uživatel přihlašuje pomocí soukromého klíče tohoto certifikátu. Aplikace má dva certifikáty. Jeden užívá při testování aplikace druhý v produkci. V závislosti na tom, v jakém stavu aplikace je, se server třetí strany přihlašuje ke dvěma odlišným serverům společnosti Apple Inc.. Tělo notifikace je tvořené JSON objektem, viz Kód 2.3. Velikost notifikace je omezena na 4096 bytů. Jestliže je větší, je na straně APNs odmítnuta.

Pro každý payload odeslaný na APNs je indentifikována notifikace parametrem `aps`. Ten může obsahovat jednu nebo více vlastností, které jsou popsány dále.

Klíč `alert` může být buď řetězec nebo slovník s dalšími parametry. Pokud zvolíme řetězec zobrazí se text zprávy se dvěma tlačítky *Zavřít* a *Zobrazit*. Jestliže specifikujeme zprávu podle slovníku, můžeme notifikaci definovat další nastavení, viz další odstavec.

Struktura `alert`, má několik vlastností. Atribut `title` je řetězec - titulek notifikace sdělující účel oznámení, `body` je tělo zprávy. `Title-loc-key` je typu string nebo null. Je to název klíče k řetězci titulku v lokalizačním slovníku, kde se nahradí podle aktuálního jazyka prostředí. `Title-loc-args` je pole parametrů, které se předávají lokalizovanému řetězci, který vytvoří formátovaný řetězec nahrazením substitučních znaků. Pokud je specifikován řetězec `action-loc-key`, je v něm uložen klíč na lokalizaci tlačítka umístěného na pravé straně namísto *View*. `Loc-key` je klíč k lokalizaci, určené pro zprávu

oznámení. Tento řetězec může být formátován pomocí substitučních znaků `%@` a `%n$@`. Parametry k nahrazení jsou předány v poli `loc-args`. Poslední atribut `launch-image` umožňuje nastavit cestu k obrázku, jenž se uživateli zobrazí při kliknutí na notifikaci. Pokud tato vlastnost není specifikována, použije se v případě existence poslední pořízená fotografie obrazovky, jinak se použije základní startovací obrazovka aplikace.

Kód 2.3: Struktura APNs push notifikace

```
{
    "aps" : {
        "alert" : {
            "title" : "Game Request",
            "body" : "Bob wants to play poker",
            "title-loc-key" : "PLAY",
            "title-loc-args" : ["Play", "Prehrat" ],
            "action-loc-key" : "PLAY",
            "loc-key" : "PLAY",
            "loc-args" : ["Play", "Prehrat" ],
            "launch-image" : "Default.png",
        },
        "badge" : 5,
        "sound" : "bingbong.aiff",
        "content-available" : 1,
        "category" : "Identifier",
    },
    "acme1" : "bar",
    "acme2" : [ "bang",  "whiz" ]
}
```

Parametr `badge` je číslo zobrazované v pravém horním rohu ikony aplikace. Upozorňuje uživatele například na počet zpráv ve schránce. Při vyplnění nuly se současný `badge` smaže.

`Sound` je string značící jméno zvukové stopy v balíčku aplikace nebo v knihovně zvuků v aplikačním datovém kontejneru. Zvuk se použije pro upozornění uživatele, že přišla notifikace. Pokud není zvuk specifikován nebo neexistuje, přehraje se systémově nastavená zvuková stopa.

Nastavení parametru `content-available` na hodnotu *1* indikuje, že je k dispozici nový obsah. Zahrnutí tohoto klíče znamená, že pokud je aplikace spuštěna na pozadí nebo obnovena, je zavolán speciální handler aplikace, který vykoná úlohu pro tento stav.

Poslední parametr `category` je řetězec. Při poskytnutí tohoto klíče s hodnotou reprezentující identifikátor vlastnosti je předán jako parametr specializované třídě, která má definovanou akci pro tuto hodnotu.

Je také možné notifikaci doplnit vlastními klíči s hodnotami, či objekty ve formátu JSON. Tyto vlastnosti jsou umístěny mimo objekt `aps`, a musí se

vejít i s tělem notifikace do maximální velikosti povolené pro notifikaci.[6]

Kapitola 3

Realizace

Řešení projektu je rozděleno na dvě aplikace. Každá z nich je spuštěná samostatně na serveru v IIS. Aplikace jsou uvnitř společného projektu ve Visual Studiu pojmenovány následovně:

- Web - je webová aplikace s GUI, kde uživatelé mohou spravovat své projekty.
- API - slouží jako rozhraní k registraci a obsluze mobilních zařízení.

Aplikace následně využívají několik podpůrných třídních knihoven, umístěných uvnitř projektu:

- Domain
- Infrastructure
- PushNotifications
- PushSharpCore
- PushSharpApple
- PushSharpGoogle
- PushSharpWindows

3.1 Třídní knihovna Domain

Tato knihovna obsahuje doménový model celé aplikace. Je základním kamenem a souborem entit pro Code First přístup, viz 3.1.4. NOTIFIT používá jako úložiště dat relační databázi PostgreSQL, viz 2.2. Za pomocí Entity Frameworku, viz 3.1.2 vytváří a migruje tabulková schémata v této databázi. Doménový model používá datové anotace, upřesňující mapování objektů do

relační databáze stanovující požadavky na parametry a jejich validaci. Pokud je po spuštění aplikace databáze prázdná, Entity Framework vytvoří celou strukturu databáze z doménového modelu. Ze tříd se generují tabulky a z třídních vlastností se vytvoří sloupce. Model je psán podle konvencí doporučených společností Microsoft. To má za následek, že jsou automaticky rozpoznány identifikátory a cizí klíče, bez vnějšího zásahu programátora.

Doménové modely nedokáží vždy pokrýt potřeby pro zobrazení dat uvnitř views. Zvlášť pokud přijde na scénu logika autorizace či zobrazení seznamů položek, je potřeba upravit model. K tomuto účelu slouží jmenný prostor `Domain.Models`, kde se nacházejí modely pro zobrazování logiky uvnitř views. Tyto modely se mapují z doménového modelu. K mapování slouží komponenta AutoMapper.

3.1.1 Mapování modelů

Modely nebo také třídy, které si jsou velmi podobné, je potřeba velmi často v MVC mapovat na sebe. Zvláště se tato nutnost objevuje mezi doménovými modely a view modely. Pokud je takovéto mapování potřeba, musíme vytvořit konfiguraci pro AutoMapper. V závislosti na složitosti mapování se dají použít dva principy.

1. Automatická konfigurace - když je zapotřebí vytvořit jednoduchou mapu, stačí implementovat k modelu generické rozhraní `IMapFrom` s typem modelu, ze kterého se má provést mapování. Při inicializaci programu se všechna tato rozhraní načtou a registruje se pro ně automaticky mapování. Jednoduché mapování spočívá v tom, že oba modely používají stejné názvy a datové typy vlastností, ty se podle shody automaticky namapují.

2. Specifická konfigurace - pro složitější a netriviální mapování. Třída musí implementovat rozhraní `IHaveCustomMapping`, které má povinnou metodu `CreateMappings`, kde se definuje, jak se na sebe objekty budou převádět. Při inicializaci programu se registruje tato specifická konfigurace do Automapperu.

V neposlední řadě se v knihovním projektu nachází seznam interface, enumerací a třídy které se serializují do XML (pro zjednodušení v databázi).

3.1.2 Entity Framework

Entity Framework je objektově-relační mapovač, který umožňuje .NET vývojářům pracovat s relačními daty. K tomu se využívá doménově-specifických objektů. Pokrývá většinu potřeb pro přístup k datům, které by musel jinak vývojář zdlouhavě psát, viz [7]. Databáze se následně dotazuje pomocí sady funkcí jazyka LINQ, viz 3.1.3.

3.1.3 LINQ

LINQ (Language-Integrated Query) je výkonná sada funkcí, rozšiřující možnosti dotazovací syntaxe jazyků C# a Visual Basic. LINQ představuje běžně užívané a snadno zapamatovatelné vzory pro dotazování či úpravu dat. Tuto technologii lze rozšířit o podporu prakticky jakéhokoli úložiště dat. Visual Studio obsahuje sestavení poskytovatele LINQ, která umožňují použití LINQ s kolekcemi rozhraní .NET Framework, databázemi serveru SQL Server, datovými sadami ADO.NET a dokumenty XML, viz [8]

3.1.4 Code First

Code First je přístup k vytváření tabulek relační databáze. Dovoluje vývojáři definovat objektový model za využití programovacího jazyka C# nebo VB.Net. Následná konfigurace je možná využitím atributů pro třídy a jejich vlastnosti nebo za použití fluent API[8]

3.2 Třídní knihovna Infrastructure

Knihovna Infrastructure definuje třídu `AppDb`. Třída dědí od třídy `IdentityDbContext<User>` patřící systému ASP.NET Identity, viz 3.2.1. V konstruktoru se rodičovské třídě `AppDb` předává parametr definující název Connection Stringu, uloženého v konfiguračním souboru aplikace, sloužícímu pro navázání spojení s databázovým systémem. Předávaný typ generickému rozhraní rozšiřuje základní typ uživatele definovaný v ASP.Net Identity o jméno a příjmení. Třída slouží jako databázový kontext. Entity Framework z každé vlastnosti typu `DbSet<T>` vytváří mapování s relační databází a umožňuje tak objektový přístup k repozitáři.

V závislosti na přístup Code First, viz 3.1.4 obsahuje projekt složku *Migrations*. Zde se ukládají všechny explicitní migrace nezbytné při změnách ve struktuře databáze. Třída `Configuration` pak nastavuje průběh migrací a obsahuje metodu `Seed`. V ní se nachází populační data pro vytvoření základních entit v databázi při migraci.

3.2.1 ASP.NET Identity

Po mnohaletém vývoji vydala společnost Microsoft řešení pro správu uživatelů napříč všemi ASP.NET frameworky jako jsou ASP.NET MVC, Web Forms, Web Pages, Web API a SignalIR. Umožňuje jednoduché připojení rozšiřujících uživatelských dat. Systém používá k uchovávání dat databázi populovanou pomocí Entity Frameworku s Code First přístupem. ASP.NET Identity zahrnuje

[8]Fluent API slouží k rozšířenému mapování objektů a pokrývá víc problémů než řeší datové anotace

uživatelské role, které umožňují aplikaci vymezovat přístupy do určitých částí jako je například Administrátorská sekce aj., viz [9]

3.3 Třídní knihovna PushNotifications

Jádrem celé aplikace je tato knihovna. Obsahuje řadu tříd a funkcí, které odesílají veškeré notifikace zařazené v databázi. Při inicializaci webové aplikace se spustí na samostatném vlákně získáním instance třídy `Processor` a zavoláním její metody `Start`.

3.3.1 Rozhraní IPush

NOTIFIT dokáže odesílat notifikace na tři různé operační systémy. Pro každý z nich je nutno implementovat rozhraní `IPush`, viz Kód 3.1. Toto rozhraní má dva generické parametry `T` a `U`.

Parametr typu `T` musí splňovat podmínku, že implementuje rozhraní `IDevice`. Tato podmínka je nutná kvůli možnosti cílit notifikace na určité typy zařízení, vytvořená uživatelovou selekcí ve webové aplikaci. Funkce `SendNotificationsForDevices` musí dostat jako parametr kolekci zařízení `IDevice`, kterým je tato zpráva určena.

Typ `U` má za podmínku implementovat rozhraní `INotificationBundle`. Toto rozhraní definuje objekt obsahující veškeré uživatelské nastavení, potřebné pro zformátování výsledné notifikace.

Kód 3.1: Rozhraní IPush

```
interface IPush<T, U>
        where T : IDevice
        where U : INotificationBundle
    {
        U SendNotificationsForDevices(IEnumerable<T>
           devices, U notificationBundle);
    }
```

Návratová hodnota `U` je typ implementující rozhraní `INotificationBundle`. Po odeslání notifikace v sobě nese metrické informace, kolika zařízením se zpráva doručila úspěšně a naopak. Obsahuje také výslednou zprávu odeslanou na notifikační službu daného operačního systému.

3.4 Třída Processor

Třída Processor se stará o frontu notifikací. Třída implementuje návrhový vzor Singleton, viz 3.4.1. To zaručuje jedinou instanci během běhu celé aplikace. Inicializuje se při spuštění webové aplikace na vlastním odděleném vlákně. Od té chvíle v určitých intervalech stahuje z databáze všechny notifikace, jejichž

čas zpracování po převodu do UTC je starší než současný čas serveru v UTC. Následně jsou všechny publikované notifikace seřazeny podle času, kdy byly do databáze uloženy od nejstaršího po nejnovější. Pokud je kolekce prázdná processor se uspí na jednu vteřinu. Jinak nastává zpracování.

Pro každou čekající notifikaci se postupně zavolá metoda `Send`. Nejprve se nastaví v notifikaci stav začátku zpracování, čas začátku zpracování a vytvoří se log o tomto stavu.

V notifikaci je možné při vytváření napsat výraz, který vybere jen specifický segment zařízení, vyhovující požadované selekci. Notifikace má v sobě uloženu tuto selekci v podobě serializované třídy `SelectionList` do formátu XML. Po zahájení odesílání dojde k deserializaci a následně k vyhodnocení kolekce výběrů za pomoci třídy `SelectionTreeParser`. Volání metody `EvaluateParser` vrátí kolekci entit typu `Device`. Ty se následně roztřídí do kolekcí dle operačního systému.

Další krok je expandování nastavení. Pokud při vytváření notifikace uživatel zvolil možnost odeslat všem aplikacím v projektu a nespecifikoval nastavení pro všechny aplikace zvlášť, vytvoří se u notifikace nové základní nastavení pro každou aplikaci bez uloženého nastavení. Tato nastavení obsahují pouze nezbytné parametry, aby mohla být notifikace odeslána na webové služby všech providerů.

Když je vše připraveno, vytvoří se tři nová vlákna zvlášť pro aplikace typu Android, Apple a Windows. Předají se jim kolekce zařízení a notifikační balíček, odkazující na aplikaci a nesoucí metrické údaje. Vzápětí se čeká na dokončení všech úloh, kdy dojde k vyhodnocení notifikace. Pokud byla nastavena periodicita vytvoří se kopie notifikace s novým časem posunutým o periodu a zařadí se do fronty. V tuto chvíli je notifikace úspěšně dokončena. Nastaví se jí požadovaný stav. Vše se uloží do databáze a začne se zpracovávat další notifikace.

3.4.1 Singleton

Singleton nebo česky jedináček je taková třída, která zabezpečí, že bude mít jedinou instanci a poskytne k ní globální přístupový bod. V programech se nutnost existence jediné instance třídy vyskytuje poměrně často. V aplikaci NOTIFIT je potřeba aby existovala pouze jediná instance třídy Processor, viz 3.4, která má na starosti frontu notifikací, které dostane z databáze a následně je zpracovává. [10, s.107-108]

3.4.2 Třída SelectionTreeParser

Jedinečnost NOTIFITu spočívá v možnosti posílat notifikace zařízením specifikovaným vlastním výběrem. Uživatel - vývojář mobilní aplikace má k dispozici v základu několik systémových hodnot, které knihovny samy dostanou ze zařízení. Jedná se například o systémové jméno, číslo modelu, jazyk zařízení,

časovou zónu a jiné. Mimo jiné si vývojář může specifikovat vlastní klíče a následně odesílat řetězce ze zařízení k těmto hodnotám na REST API NOTIFITu. Klíče jsou ukládány globálně pro celý projekt a lze pak podle nich vytvářet lepší selekce zařízení napříč aplikacemi. Jako příklad můžeme jmenovat klíče typu hodnocení aplikace, email a jiné. V každé notifikaci je tato selekce uložená jako řetězec v podobě serializované třídy `SelectionList`.

Jeden řádek selekce se skládá z několika částí:

- Prefix - Výraz může být součásti většího celku. Proto je možné před výrazem začít závorkami k vytvoření priority výrazu, nebo je potřeba vložit logické operátory AND či OR.
- Key - Klíč je hodnota vytvořená buď uživatelem nebo pomocí knihoven NOTIFITu. Značí vlastnost zařízení, ke které se ukládají pravidelně aktualizované hodnoty. Výraz bude vytvářet selekci na základě zvoleného klíče.
- Rule - Pravidla pro vyhodnocení výrazu jenž jsou dána systémem. Uživatel si může vybrat jednu z možností, jak bude selekce probíhat při vyhodnocování výrazu hodnota klíče. Poskytnuté možnosti jsou: začíná na, je rovna, končí na, nebo obsahuje.
- Value - Hodnota, které mají vybraná zařízení odpovídat.
- Postfix - Stejně jako u prefixu může být potřeba ukončit logický výraz nebo začít logickými operátory pro další řádek selekce zařízení.

Selekce následně probíhá ve dvou fázích. V konstruktoru třídy `SelectionTreeParser` se nastaví parametry nutné k inicializaci třídních proměnných. Je zde předána reference na třídu `SelectionList`, obsahující všechny řádky selekcí, které uživatel vytvořil. Na ni je okamžitě volána metoda `Parse`. Pokud nebyly vytvořeny žádné řádky metoda vrací `Null`. Naopak je potřeba převést selekce na logický výraz. Na třídu `SelectionList` se zavolá přetížená metoda `ToString()`, zjednodušující výraz tím způsobem, že postupně navštíví všechny selekce, a skládá za sebe prefix, vzápětí vytvoří číslo výrazu ve tvaru `{i}`, kde `i` odpovídá číslu zpracovávaného řádku. Nakonec vloží řetězec postfix výrazu. Po vrácení zjednodušeného výrazu je nutné převést infixový zápis do postfixového, aby mohlo dojít k jeho vyhodnocení.

S postfixovým zápisem je již možné vytvořit binární strom. V jeho listech se nacházejí čísla odkazující na řádek v kolekci selekcí. Uzly nesou informaci o tom, zda má být s jeho potomky provedeno logické sjednocení nebo součin.

Ve vhodné chvíli se volá metoda `EvaluateTree` jejíž návratová hodnota je již kolekce `IEnumerable<Device>`, která obsahuje zařízení vyhovující logickému výrazu.

K vyhodnocení dochází pomocí průchodu binárního stromu v pořadí postorder. Postupně se v každém listu nahlédne na řádek selekce a z databáze

se vyberou odpovídající zařízení, která mají hodnotu klíče splňující pravidlo zadané uživatelem. Při probublávání směrem ke kořeni se na kolekcích zařízení provádí buď logický součin nebo součet v závislosti na operátoru uvnitř uzlu. Kořen po vyhodnocení obsahuje kolekci zařízení splňující logický výraz uživatele.

Jestliže se stane, že uživatel nevytvořil žádnou selekci, metoda vrátí všechna zařízení, náležející danému projektu. V opačném případě vrátí vyhodnocený výraz.

3.5 Webová aplikace NOTIFIT

NOTIFIT aplikuje populární architektonický styl MVC - Model, View, Controller. Pro přehlednost se nachází všechny modely v projektu `Domain` rozdělené do jmenných prostorů podle využití, aby byly dostupné i jiným projektům, které s nimi potřebují pracovat. Aplikace využívá spoustu výhod frameworku ASP.NET MVC. NOTIFIT implementuje vlastní aplikační framework, který se snaží zaručit jednoduchost, dobrou udržovatelnost a škálovatelnost kódu. Tento aplikační framework se soustředí převážné ve jmenném prostoru `Web.Infrastructure`.

3.5.1 Webový framework Infrastructure

Webový framework implementuje návrhový vzor IoC, viz 3.5.2. K injekci konkrétních datových typů využívá NOTIFIT rozšíření `StructureMap`.

V souboru Global.asax, jinak známém jako ASP.Net aplikační soubor, je na nastaveno, aby se jako rozpoznávač závislostí, neboli `DependencyResolver`, použil vlastní resolver `StructureMapDependencyResolver`. Resolver musí implementovat rozhraní frameworku ASP.Net MVC `IDependecyResolever`, které vystavuje dvě metody `GetService` a `GetServices`. Obě metody přebírají parametr `Type` a jejich návratovou hodnotou je `object`. Pokud je vytvářen některý datový typ, který má dostat parametry některé konkrétní implementace rozhraní jako referenci, dojde k takzvané injekci a `StructureMapDependencyResolver` se pokusí pomocí IoC Containeru konkrétní typ vytvořit za pomocí vyřešení závislostí a vytvoření všech potřebných konkrétních datových typů.

Během startu webové aplikace je v tovární třídě `IoC` konfigurován objekt `Container`. Tento IoC Container má na starosti vyřešení závislostí tj. dependenci resolution. Jeho konfigurace probíhá v aplikačním souboru *Global.asax*. Nastavení se předávají registry, umožňující rozčlenit konfiguraci do menších segmentů.

První registr scanuje všechny typy dodržující konvence. To znamená, že pokud je potřebné v době běhu vytvořit některou třídu, které nejsou předány v konstruktoru konkrétní datové typy požadovaným rozhraním, StructureMap automaticky vytvoří instance konkrétních typů a předají se konstruktoru. Konvence je taková. že všechny rozhraní začínají na písmeno *I* a jejich konkrétní

datové typy mají stejný název bez úvodního I (např. rozhraní IObject a konkrétní typ Object).

Další je `ControllerConvention` registr. Ten zaručí, že životní cyklus všech instancí tříd uvnitř controllerů, bude vždy jen na jeden HTTP požadavek. Registr ve smyčce projde postupně všechny třídy, které nejsou abstraktní, a je možné je převést pomocí `cast` na třídu `Controller`. Tato selekce zařídí to, že i NOTIFIT controllery, dědící od `NotifitController` budou vybrány a registrovány frameworkem MVC. Pro vybrané třídy se nastaví životní cyklus `UniquePerRequestLifecycle`.

`MvcRegistry` zajistí možnost přistupovat k běžným abstrakcím MVC frameworku z míst, která by to běžně neumožnila. U tříd, které nejsou potomky třídy `Controller` můžeme požádat v konstruktoru například o identitu právě přihlášeného uživatele, nebo dostat aktuální instanci webového požadavku `HttpContextBase` a jiné běžně nedostupné objekty vně controlleru.

V aplikaci bývá potřeba spouštět úlohy v různých chvílích během HTTP požadavků. `TaskRegistry` zajistí, že všechny typy nacházející se v sestaveních začínajících řetězcem NOTIFIT a implementují alespoň jedno z rozhraní `IRunAtInit, IRunAtStartup, IRunOnEachRequest, IRunOnError, IRunAfterEachRequest` budou registrovány. V souboru *Global.asax* jsou definovány metody pro každou situaci, a při dané události se postupně provedou všechny registrované úkoly zavoláním metody `Execute()` - povinnou metodou pro všechna rozhraní.

Registr `ModelMetadataRegistry` začlení do IoC Containeru a registruje do MVC všechny typy implementující rozhraní `IModelMetadataFilter`. Jedná se především o filtry zpracovávající atributy a upravují typy dat putujících do modelu, kde jsou následně zobrazeny.

`ActionFilterRegistry` slouží k nastavení vlastních filtrů metod. V běhu aplikace se MVC ptá StructureMapu, zda má k dispozici vlastní implementaci rozhraní `IFilterProvider`; řekneme, aby byla užita naše vlastní implementace `StructureMapFilterProvider` namísto základního poskytovatele MVC `FilterAttributeFilterProvider`. Tento nový poskytovatel je rozšířený o možnost užívat IoC Container uvnitř filtrů. StructureMap v základu neposkytuje injekci setterů vlastností. Je tedy potřeba nastavit injekci vlastností uvnitř filtrů, ale jen u těch, které nejsou filtry MVC, kde by mohlo dojít k poškození stávajících filtrů MVC. Je tedy vytvořena konvence, říkající StructureMapu aby všechny vlastnosti, které mohou být převedeny na typ `ActionFilterAttribute` nacházející se v jmenném prostoru začínajícím řetězcem Web, a nejsou primitivní nebo string, byly injektovány. Registru `ActionFilterRegistry` je předána lambda výrazem anonymní metoda pro získání IoC Containeru.

Třída `CurrentUser` je implementací rozhraní `ICurrentUser` definující vlastnost `User` datového typu `User`. Dotaz na vlastnost automaticky vyhledá v databázi přihlášeného uživatele a vrátí jeho instanci. Pokud již k vyhledávání dříve došlo, vrátí se pouze reference na existující instanci, čímž se zabrání zbytečnému dotazování na databázi.

Základním typem controlleru je ve webové aplikaci třída `NotifitController`. V konstruktoru jsou předány reference na aktuální databázový kontext typu `AppDb` a implementaci rozhraní `ICurrentUser`, sloužící k získání přístupu k uživateli. Parametry jsou uloženy do třídních vlastností a jsou dostupné i potomkům třídy. `NotifitController` obsahuje několik pomocných funkcí pro své potomky. Přetížená metoda `IsOwner` s návratovou hodnotou typu `bool` pomáhá určit, zda je daný uživatel vlastníkem předané notifikace, projektu, či aplikace. Jako další implementuje metodu `RedirectToAction`. Ta pomáhá zbavit se za pomocí balíčku *Microsoft.AspNet.Mvc.Futures* takzvaných *magických stringů*. Umožňuje namísto standardního přesměrování od MVC frameworku, využívající stringů a anonymních objektů pro předání parametrů nadcházející akci, přesunout se k silně typovému zápisu, viz Kód 3.2. Díky tomu je zajištěna korektnost přesměrování již během překladu, a není ani možné zmýlit se v předávaných parametrech, protože se přesměrování chová jako volání jednoduché metody na rozdíl od běžného zápisu.

Kód 3.2: Vyhnutí se magickým stringům

```
// Standartni presmerovani v~MVC
return RedirectToAction("Details", "ProjectController",
    new {projectId = project.Id});
// Silne typove presmerovani diky NotifitControlleru
return RedirectToAction<ProjectController>(c =>
   c.Details(project.Id));
```

3.5.2 Inversion of Control

Inversion of Control - neboli obrácené řízení se snaží odpoutat nutnost používat třídy závislé na jiných pevně daných třídách. Tento problém nás pevně svazuje v testování a nelze měnit některé závislosti v době běhu. V obráceném řízení se datové typy konkrétních tříd nahrazují rozhraními. Třídě je pak v době běhu předán konkrétní datový typ implementující dané rozhraní a odpoutává se provázanost mezi konkrétními typy.

3.5.3 Filtry

Když putuje model z controlleru do view, není vždy formátován tak, jak bychom si přáli. Některé vlastnosti modelu nemusí být inicializovány, jiné v sobě nenesou správnou informaci o tom, jak se mají ve view zobrazit. Model je sada vlastností, které jsou zapotřebí ve view. Ať už jde o pouhé zobrazení informací uživateli, nebo o předvyplnění webových formulářů. Filtry se nacházejí ve složce Filters ve jmenném prostoru `Web.Infrastructure.Filters`. Každý z filtrů implementuje rozhraní `IModelMetadataFilter`. Při inicializaci aplikace jsou díky IoC automaticky registrovány a předány MVC.

První filtr `DateTimeFilter` slouží k inicializaci datového typu `DateTime`. Základní hodnota instance objektu `DateTime` je rovna *00:00:00.0000000 UTC, 1. ledna, 0001* v gregoriánském kalendáři. Pokud by došlo k uložení do databáze této hodnoty, bude vyvolána vyjímka. SQL server dokáže ukládat jenom datum větší rovno 1. ledna 1753. Je proto nezbytné aby všechny typy `DateTime` měli nastaveno jako nejmenší hodnotu toto datum. Filtr tedy zajistí, aby všechny data která se mají zobrazit uživateli, a jsou menší než minimální hodnota SQL serveru byla nastavena na aktuální čas.

V modelu jsou v jistých chvílích předávány i komplexní datové typy (třídy vytvořené uživatelem). Ve views je pak potřeba psát velkou řadu podmínek na tyto objekty, a kontrolovat zdali nejsou `null`. Většinou se na tyto podmínky při úpravách zapomene a kód se těmito kontrolami velmi znepřehledňuje. `ObjectNullInitializationFilter` tuto nutnost kontroly potlačuje. Pro každou třídu s hodnotou `null`, která není primitivního datového typu, typu `string` nebo `byte[]` zavolá systémovou třídu `Activator` a požádá ji o vytvoření instance. Ta se uloží do modelu místo hodnoty `null`.

Filtr `LabelConventionFilter` je velmi užitečný pro vylepšené zobrazování názvů proměnných ve views. Standardní konvence zobrazování názvů veřejných proměnných modelu je taková, že např. parametr s názvem `UserName` ve view používá k zobrazení jména proměnné helper `@HTML.LabelFor(model => model.UserName)`. Ten jednoduše zobrazí: *UserName*. Název je bohužel bez mezer a uživatel musí přidat vlastnosti modelu atribut `[DisplayName("User Name")]`. Tento způsob vede k překlepům a nesoudržnosti napříč aplikací. Proto `LabelConventionFilter` pomocí speciálního regulárního výrazu přiřadí každé vlastnosti, která nemá již nastaven atribut `DisplayName`, nové jméno rozdělené mezerami tam, kde jsou velká písmena.

`NotificationWorkflowFilter` funguje ke zjednodušení logiky při tvorbě nové notifikace. Aplikuje se na controllery, které se nacházejí v tomto notifikačním workflow. Postup tvorby je uložen v entitě notifikace jako číselná hodnota z enumu `WorkflowProgres`. Filtr má dva parametry `CurrentStage` a `MinimumRequiredStage`. Před zahájením vykonávání metody controlleru se filtr pokusí načíst ze sezení proměnnou `Notification`. Pokud se mu to podaří, požádá IoC Container o instanci databázového kontextu a získá entitu notifikace. V další fázi se porovná stav dosud nejvyššího dokončeného stádia workflow pro tuto notifikaci. Jestliže je současný stav vyšší než nejvyšší dokončený stav, je uživatel přesměrován na stav, který je pro uživatele nejvyšší možný. To zaručuje, aby nedošlo k přeskočení žádného z kroků v této workflow. V opačném případě je umožněno pokračovat. Po skončení metody, se filtr podívá na `HttpContext` jestli byla jeho metoda POST. Pokud ano, notifikaci se nastaví hodnota workflow, kterou měl controller nastavenu jako `CurrentStage` a uloží se do databáze.

Jestliže je potřeba vlastnost modelu pouze zobrazit a znemožnit její změnu aplikuje se filtr `ReadOnlyTemplateSelectorFilter`. Ten vyhledá všechny vlastnosti, které mají atribut `[ReadOnly]` a nastaví jejich metadata modelu vlast-

nost `DataTypeName` na *ReadOnly*. To v běhu zajistí výběr správné templaty zobrazující hodnotu v needitovatelném režimu.

`TextAreaByNameFilter` zajistí, že všechny vlastnosti modelu, které nemají nastavený atribut `DataTypeName`, a obsahují ve svém názvu některou z hodnot: *body*, *comments*, *log*, aby se jim nastavil `DataTypeName` na MultilineText. Ten jim přiřadí správnou zobrazovací template, která zobrazí textový vstup v několika řádkovém editoru.

Filtr `WatermarkConvetionFilter` nastaví všem vlastnostem modelu, které nemají nastavený atribut `Watermark` na hodnotu metadata `DisplayName` s připojenými třemi tečkami na konci. Watermark se zobrazuje uživateli ve všech vstupních polích, která neobsahují doposud žádný text.

3.5.4 Helpers

Helpery v MVC slouží ke zjednodušení psaní HTML kódu. V programu se vytvoří konfigurovatelná metoda, kterou je možné volat uvnitř views. Vytváření helperů a jejich užívání, vede k snadněji udržitelnému kódu a zpřehlednění. Veškeré helpery se nacházejí ve složce Helpers.

`BeginCollectionItem` je helper sloužící jako pomocník pro formuláře, kde je nutné pracovat s kolekcemi. Pokud potřebujeme odeslat do controlleru model s kolekcí objektů, je třeba mít na paměti, že tyto podformuláře musejí být ve správném tvaru a korektně pojmenovány, aby se uvnitř frameworku dobře namapovali na objekty, a vytvořili kolekci. Je potřebné stanovit unikátní názvy pro formuláře začínající na řetězec nesoucí jméno kolekce složené s unikátním identifikátorem známým jako struktura `GUID`.

Další třída webových helperů nese pojmenování `BootsrapHelpers`. Vzhledem k faktu, že celý web čerpá základní styly především z responsibilních CSS stylů Bootsrapu, je zapotřebí udržovat jednoduchý systém v tom, jak se budou styly aplikovat na jednotlivé HTML prvky. Je totiž možné, že s příchodem nové verze Bootsrapu, se mohou některé styly změnit nebo dokonce přejmenovat. To by vedlo k obrovskému refaktoringu, kdybychom aplikovali styly na HTML prvky implicitně v kódu. `BootstrapHelpers` tedy implementuje dva helpery rozšiřující obyčejné helpery `Label` a `LabelFor`. Ty vytvářejí popisky ve formulářích. Modifikovaný helper vytvoří základní typ a automaticky jim přiřadá požadované třídy CSS stylů. Při případných úpravách pak tedy stačí přepsat CSS třídu pouze na jednom místě a změny se okamžitě projeví napříč celou aplikací.

`ButtonHelper` slouží k zajištění použití stejných tlačítek v rámci celé aplikace. Jedná se o čtyři druhy tlačítek. První `ButtonAddNew` mluví samo za sebe a slouží k vytvoření tlačítka pro přidávání nových objektů. Stejně jako ostatní tlačítka přebírá několik parametrů specifikující text uvnitř tlačítka a cestu s parametry pro odkaz. Tlačítko zobrazí před zobrazeným textem ikonku plus zajišťující lepší vizuální přehled pro uživatele. Další je tlačítko `ButtonGoBack` se stejnou syntaxí jako mělo předchozí. Jediná výjimka je ikona šipky vlevo

pro upozornění uživatele, že bude přesměrován zpět. `ButtonSpecial` umožňuje specifikovat ikonu tlačítka a třídu pro zvolení barevnosti. Poslední tlačítko `ButtonSubmitForm`, určené pro odesílání formulářů, se mění podle toho, jestli se uživatel nachází ve stádiu vytváření nové entity, nebo určitou entitu edituje. Tato informace se předává pomocí dynamické kolekce `ViewBag`. Při vytváření je jméno nastaveno na *Create*, při editaci na *Save Changes*. Pokud by v kolekci nebyla informace o jakou situaci se právě jedná, zobrazí se pouze text *Submit*.

`NotificationWorkflow` je helper pro zobrazení lišty při vytváření notifikace. Notifikace je natolik komplexní objekt, že jsem se rozhodl proces tvorby rozdělit do několika kroků. Při vytváření je nutné, aby měl uživatel odemknuta na liště postupu pouze ta tlačítka, jejichž fázi již buď dokončil, nebo na kterých právě pracuje. Ostatní musí zůstat zamčená, a proto se zobrazují pouze jako text. Helper při inicializaci obdrží tyto informace o současném stavu jako parametry a podle toho rozhodne, jak danou položku notifikační lišty zobrazí.

3.5.5 Webová oznámení

Oznámení jsou výborným prostředkem, jak uživateli sdělit, že se například zdařilo uložit novou položku, nebo něco upravit smazat, či zobrazit informaci o jiné akci. NOTIFIT má implementováno rozšíření pro datový typ `ActionResult`. Tento typ je návratovou hodnotou většiny metod controllerů. Ve chvíli, kdy je potřeba uživateli cokoliv oznámit, stačí zavolat za návratovou hodnotou metody jedno z rozšíření ze statické třídy `AlertExtensions`, viz Kód 3.3, kde dojde ve view oznámení o uložení změn. Je možné vybrat z několika možných výsledků lišících se především ve výsledné barvě oznámení. Oznámení je možné i řetězit za sebe. Při vyhodnocení controlleru do view je zavolána metoda `ExecuteResult`. Všechná oznámení jsou přidána do dynamické kolekce `TempData` a uložena pod klíč Alerts. V layoutu[9] aplikace je vytvářeno parciální view _Alerts. To projede celou kolekcí oznámení uloženou v `TempData` a za použití javascriptového rozšíření toastr je vypíše uživateli.

Kód 3.3: Poslání oznámení do view

```
return RedirectToAction<ProjectController>(a =>
    a.MyProjects()).WithChangesSaved();
```

3.5.6 Šablony datových typů

V Mvc se pro zobrazování editorů dat uvnitř views používá HTML helper `HTML.Editor` případně `HTML.EditorFor` pro silné typování. Tyto editory je možné upravit pomocí tříd a atributů. Samozřejmě pokud takto ručně upravujeme editory v každém view, stává se to osudnou příčinou chyb a opakování

[9]Layout je základní HTML obalující téměř všechna views. Jsou zde definovány elementy `HTML, HEAD, BODY` a v těle renderuje aktuální views

se v kódu. Proto je výhodnou možností tyto editory upravit na jednom místě a používat něco jako jejich přetížené šablony. Všechny upravené předlohy musí být umístěny ve složce *EditorTemplates* uvnitř *Shared* složky *Views*. Šablony si framework vybírá podle zobrazovaného datového typu. Pokud je potřeba zobrazit speciální šablonu pro parametr modelu, je možné toho docílit pomocí atributu, viz Kód 3.4.

Kód 3.4: Atribut pro zobrazení vlastní šablony

```
[DataType("MyTemplate")]
public MyType MyTypeName { get; set; }
```

NOTIFIT používá přetížené editory hlavně kvůli možnosti nastavení správných CSS stylů a umístění watermark popisků. Jsou implementovány templaty pro základní datové typy jako je Int32, String. Speciální je například MultilineText, který zobrazí větší textové pole pro vlastnosti, u kterých se počítá s delším obsahem. Dále například šablona DateTime, které pro datum vytvoří hezké javasriptové UI pro volbu datumu a času pomocí kalendáře.

V určitých views je zapotřebí nahrávat soubory. NOTIFIT používá javascriptový uploader souborů s názvem Dropzone. Tento script nabízí přívětivé UI a funguje asynchronně, to znamená že soubory nejsou odeslány ve formuláři s ostatními. Šablona pro zobrazení upload políčka se skládá z pole, kam je umístěn doplněk Dropzone, ze skrytého formuláře pro URL obrázku a pole, kde se zobrazí čerstvě nahraný, nebo již existující obrázek. Šablona generuje jedinečné GUID identifikátory pro označení těchto polí. Následně jsou předány inicializační javascriptové funkci. Nejprve se vytvoří GET požadavek na server, aby vytvořil a poslal URL k miniatuře obrázku, který zaslal v parametru. Při úspěchu se obrázek umístí vedle doplňku Dropzone. Inicializuje se Dropzone plugin, kde se nastaví omezení nahrávání maximálně na jeden soubor pro každý prvek, a nastaví se cesty k API, řešící nahrávání a mazání souborů. Jako handler pro tuto manipulaci slouží controller `FilesController`. Při nahrávání je zjištěna přípona souboru a jméno je nahrazeno GUID identifikátorem. Následně se soubor uloží do serverové složky *tmp*. Pokud je soubor obrázek, je zmenšen automaticky tak, aby delší strana měla maximálně 1000px. Na konci se vytvoří relativní URL, odešle se pomocí JSONu zpět do view, kde je uložena jako hodnota skrytého vstupního pole. Při ukládání formuláře je nezbytné po vytvoření prvku v databázi, získat cestu k adresáři, kde by měl být obrázek správně uložen. Volání se provádí po uložení do databáze, kdy je vytvořen identifikátor entity nutný k získání nové cesty. Poté se pro parametr entity s uloženou dočasnou cestou zavolá metoda `ManageUpload`. Ta soubor přesune do nového adresáře, v případě že se aktuální URL nerovná nově zkonstruované cestě, a vrátí novou relativní adresu souboru, která entitu aktualizuje.

Pro zobrazení formuláře celého modelu poskytuje MVC šablonu `EditorForModel`. Ta v základní podobě popořadě zobrazí editory pro všechna pole

modelu. Pro nastavení hezkého vzhledu je ale potřebné upravit tuto šablonu ukryté pod názvem Object. Její model je typu `dynamic`., viz Kód 3.5

Kód 3.5: Šablona pro zobrazení modelu

```
@model dynamic

@foreach (var prop in ViewData.ModelMetadata.Properties)
{
    if (prop.TemplateHint == "HiddenInput")
    {
        @HTML.Hidden(prop.PropertyName, prop.Model)
    }
    else if (string.IsNullOrEmpty(prop.DataTypeName) ||
                !(prop.DataTypeName.Equals("DontShow")))
    {
        <div class="form-group">
            @HTML.BootstrapLabel(prop.PropertyName)
            <div class="col-md-10">
                @HTML.Editor(prop.PropertyName)
                @HTML
                  .ValidationMessage(prop.PropertyName,
                     "", new {@class = "text-danger"})
            </div>
        </div>
    }
}
```

Kód 3.5 vytvoří postupně pro všechny vlastnosti modelu editory konkrétních datových typů. Pro vlastnosti s atributem `[HiddenInput]` se umístí do HTML skrytá vstupní pole. Jedná se především o pole, ukrývající identifikační číslo upravované entity. Vlastnosti dekorované `[DataType("DontShow")]` atributem se pouze přeskočí. Zbytek vlastností je již obalen HTML tagy pro zkrášlení zobrazovaného formuláře. Nastaví se popisek vlastnosti, editor a připraví se validační zpráva v případě, že by došlo v chybě při vyplňování pole. Editor vlastnosti následně sáhne podle datového typu po správné šabloně, a vytvoří vstupní pole v HTML. Tento helper nám usnadní obrovskou práci s views a kód je v případě změny velmi snadno upravitelný v globálním měřítku.

3.6 API

Knihovny vyžadují kvůli registraci zařízení do NOTIFITu backendovou službu, která by jim to umožnila. NOTIFIT implementuje REST službu, viz 3.6.1. NOTIFIT API využívá framework ASP.NET Web API 2. Dokáže jednoduše

zpracovávat webové požadavky ve formátu JSON nebo XML. Odpovědi chodí zařízením v závislosti na hlavičce požadavku, kde hodnota parametru `Content-Type` zajistí správné naformátování výstupu požadované knihovnou.

3.6.1 REST

Pojem REST byl poprvé představen v disertační práci Roye Fieldinga v roce 2000. Vzhledem k tomu že Fielding byl jedním ze spoluautorů protokolu HTTP, mají oba pojmy spolu hodně co do činění. REST je architektura rozhraní využívaná pro distribuované prostředí. REST je orientován datově na rozdíl od SOAP či XML-RPC, které jsou procedurální. Webová služba definuje vzdálené procedury a protokol pro jejich zavolání. REST definuje, jak se přistupuje k datům, viz [11].

Rozhraní REST se používá pro přístup k zdrojům v podobě dat. Ke zdrojům se přistupuje pomocí adres URI, a jsou definovány čtyři základní metody pro přístup k nim:

- GET - je základní metoda pro získání zdroje. Zavoláním URI s HTTP metodou GET obdržíme data. Stejně jako například obyčejné zobrazení webové stránky. K identifikaci zdroje je možné v URI posílat parametry. Tím docílíme obdržení konkrétního zdroje.

- POST- Slouží k vytvoření zdroje. Vzhledem k tomu, že v této době není znám identifikátor tohoto zdroje, je nutné mít domluvený přístupový bod - endpoint (což není nic jiného než nějaká URI). Na službu se pošlou v těle požadavku data, naformátovaná do podoby, které API rozumí. Odpovědí po zdařilém vytvoření bude HTTP status kód hlásící úspěch a v těle nově vytvořená data.

- PUT - funguje obdobně jako POST. S rozdílem, že se přistupuje k již vytvořenému zdroji. Používá se jako metoda, která aktualizuje data a upraví existující zdroj. Po volání se vrací prázdná odpověď.

- DELETE - Je podobné jako volání GET. Výsledek je smazání zdroje. V odpovědi se většinou vrací smazaný zdroj s HTTP kódem úspěchu.

3.6.2 Registrace zařízení do databáze

Po stažení mobilní aplikace od vývojáře registrovaném v NOTIFITu, se při jejím prvním spuštění inicializuje knihovna NOTIFIT. Podle typu zařízení poskytuje NOTIFIT API tři podobné controllery pro vytvoření zařízení operačních systémů Windows, IoS a Android. Inicializace probíhá tak, že se pomocí HTTP POST požadavku odešle model obsahující systémové klíče a aplikační a projektový token. Podle tokenů se určí, kam zařízení spadá. Při nalezení aplikace se vytvoří entita zařízení, a namapují se hodnoty nutné pro uložení.

Následuje registrování systémových proměnných specifikovaných NOTIFITem a jejich přiřazení k zařízení. Po uložení do databáze je odeslán knihovně jedinečný komunikační token sloužící pro identifikaci zařízení.

V druhém kroku knihovna zasílá při každém novém spuštění HTTP požadavek PUT. S ním musí zaslat již notifikační token, obdržený od poskytovatele notifikačních tokenů daných operačním systémem. Je potřebné odesílat tato data při každém spuštění, protože notifikační tokeny se mohou časem měnit, a mohlo by docházet k neúspěšnému odesílání notifikací. Zároveň se odesílají i systémové hodnoty. Pokud se některá změní, je uložena do databáze se značkou, že se jedná o nejnovější. Předchozí hodnotě je tento stav odebrán. Odpověď na tento požadavek je jen HTTP kód 200 značící úspěch.

3.6.3 Uživatelské proměnné

Programátor mobilní aplikace může mít potřebu přidávat vlastní klíče k zařízením, které mu mohou posloužit k lepšímu určení uživatele nebo konkrétního zařízení. NOTIFIT disponuje možností přidávat tyto klíče. Ty jsou globální pro celý projekt. Pro uložení nové hodnoty slouží endpoint `KeyValues`. Čas volání této metody určuje programátor mobilní aplikace. Knihovna následně odešle v těle požadavku klíč a hodnotu. API provede kontrolu zda daný klíč již existuje a eventuálně ho registruje do databáze. Podívá se, zda již zařízení nemá u tohoto klíče nějaký záznam, a zaručí že se nový záznam označí jako nejnovější, u staré hodnoty se tato vlastnost odznačí.

3.7 Testy

NOTIFIT implementuje pro webovou sekci Systémové testy. Jedná se o různé scénáře, kterými může uživatel NOTIFITu ovládat aplikaci. Tyto testy pokrývají automatické vyplňování formulářů a jejich odesílání na server. Zjišťují výsledný stav aplikace a kontrolují, zda došlo ke správnému vyhodnocení požadavků. Tyto testy jsou plně automatizované. Probíhají za pomoci několika frameworků a connectorů pro webové prohlížeče. Samotné testování probíhá v prohlížeči Chrome. Oproti např. Internet Exploreru je totiž Chrome několikanásobně rychlejší[10] při spouštění a průběhu testů. To je velmi důležité, protože samotné systémové testy trvají oproti jednodušším testům až několik minut a rychlost je tedy důležitým faktorem. Systémové testy se snaží pokrýt co nejvíce scénářů, kterými by uživatel NOTIFITu mohl projít, ale ne vždy je samozřejmě možné předvídat uživatele v plném měřítku a nemůže nahradit testování člověkem.

Systémové testy se spouštějí těsně před publikováním nové verze aplikace na produkční server. Pokud by při vývoji došlo k poškození některé zásadní

[10] Srovnání internetových prohlížečů z `http://internet-browser-review.toptenreviews.com/`

funkčnosti webu, testy je objeví. Následuje opravení chyb a následně je možné znovu aplikaci otestovat. V případě, že všechny testy projdou je možné aplikaci publikovat.

Tyto testy jsou velmi složité. Musí totiž interagovat s prohlížečem, vyplňovat formuláře, nebo nalézat prvky a jednotlivé tagy v prohlížeči, či kontrolovat současnou URL. I přesto, jsou ale díky použitým frameworkům přehledné, a dá se v nich velmi rychle a jednoduše zorientovat. Pro testy je vyhrazen projekt *Web.Tests*.

3.7.1 Implementace testů

Jak již bylo zmíněno v sec. 3.7 při testech je použita řada frameworků.

Nejnižší vrstvou testů je testový framework NUnit, sloužící k tvorbě testů pro platformu .Net. Jde o převzetí známého frameworku JUnit využívaného v Javě. Slouží jako základ testového frameworku. Definuje testové atributy pro metody a třídy, které jsou rozpoznány testovým runnerem, který testy vyhodnocuje.

Pro implementaci testů, které dokáží proklikávat webový prohlížeč, je nutné použití dalšího softwaru. V mém případě se jedná o populární Selenium WebDriver. Tento nástroj dokáže automatizovat uživatelské úlohy na internetu. Zvládne ovládat webové prvky, či automatizovat definované úlohy ve webovém prohlížeči. WebDriver je objektově orientované API umožňující psát tyto úlohy a testy.

Moq framework je mockovací knihovna umožňující testování jednotek, aniž by programátor musel používat konkrétní typy pro závislosti metod a tříd. Moq se stará o vytvoření a dosazení těchto tříd za nás. Pokud očekáváme od určité závislosti nějaké chování, můžeme Moqu specifikovat, jak přesně očekáváme, aby se daná závislost chovala pro naši situaci. Například daný controller dostává v konstruktoru některou implementaci služby, která umí odesílat emaily a vrací buď *true* pro úspěch, nebo *false* pro nezdar. Při testech jednotek nás nezajímá chování této třídy, zajímá nás pouze, jak se projeví výsledky jejího chování na určitý požadavek. Řekneme tedy Moqu, aby vytvořil třídu, která implementuje rozhraní této služby pro konkrétní metodu, o které víme, že bude zavolána, a sdělíme pouze jakou hodnotu má vrátit. Tím zvýšíme abstrakci našeho kódu a zjednodušíme testovatelnost bez omezení na používání specifických závislostí.

Testy NOTIFITu používají vysoce abstraktní framework SpecsFor a SpecForMvc napsány Mattem Honeycuttem. Tento balíček přidává do projektu všechny výše uvedené frameworky plus některé další. Jedná se například o balíček Should, který zpřehledňuje testy svojí syntaxí a názvy metod používajících lambda výrazy. SpecsFor má řadu výhod, které přijdou programátorovi rychle pod ruku. Automaticky moquje objekty pro závislosti tříd, ale zároveň je umožňuje kdykoliv nastavit, a upravit jejich chování, pokud je to třeba. Posky-

tuje řadu pomocných helperů. Například metodu `ShouldLookLike` umožňující porovnávat dva objekty na základě poskytnutých veřejných vlastností.

SpecsFor následuje vzor testování *Given When Then*. Tedy v klauzule When je dán kontext testu. Jedná se o nastavení Moqu a jiné předpoklady k začátku testu. Ve fázi When se vykoná akce, kterou systém podléhající testování prochází. Then, neboli potom je stav kdy se kontrolují - testují výsledky provedené akce.

SUT neboli systém podléhající testu je třída, která je vystavena testování. SpecsFor automaticky vytváří instanci této třídy. Je přístupná jako objekt od fáze inicializace po celý životní cyklus SpecsFor testu. SUT třída je specifikovaná v definici třídy testu. Dědí od `SpecsFor<T>` kde se generický typ `T` stává třídou podléhající testu (SUT).

Konfigurační třída je třída s atributem NUnit frameworku `[SetUpFixture]` s názvem `SpecsConfiguratuion`. V konstruktoru jsou specifikovány konvence testování. NOTIFIT využívá dvě, viz Kód 3.6.

Kód 3.6: Konstruktor SpecsConfiguration

```
public SpecsConfiguration()
  {
     WhenTesting<INeedDatabase>()
                .EnrichWith<EFContextFactory>();
     WhenTesting<SpecsFor<MvcWebApp>>()
                .EnrichWith<DatabaseCreator>();
  }
```

Nastavení zařídí, že pokud je při testování implementováno testovou třídou rozhraní `INeedDatabase`, bude k dispozici při testování instance objektu přístup k databázovému contextu. To zařídí třída `EFContextFactory`. Při inicializaci se řekne moqovacímu kontejneru, aby použil konkrétní typ `AppDb` a při skončení testů ji náležitě uvolnil.

Druhé nastavení zajistí, aby při testování, kde je kontextem webová aplikace byla vytvořena nová databáze. Jestliže již existuje, použije se stávající. Tím se zaručí, že se vytvoří jen při prvním testu a ne pro každý zvlášť. Třída `DatabaseCreator` v tomto případě zjistí přítomnost staré databáze, smaže ji, a provede seedovací metodu, která naplní databázi daty potřebnými k testování.

V konfigurační metodě `AfterConfigurationApplied` se vytvoří nastavení pro SpecsForMvc. Sdělí se, že je potřeba použít IIS Express webový server, jméno testovaného projektu, cesta k MS Build nástroji pro umožnění deploye na server a použití trasformace konfigurace *web.config*, kde je specifikován speciální connection string pro testovací databázi. Sdělí se, jaká routovací tabulka má být použita. Velmi důležitá poznámka je, že na téměř všechny akce musí být uživatel NOTIFITu přihlášen. Proto je v konfiguraci přítomná generická metoda `AuthenticateBeforeEachTestUsing` přebírající typ `CredentialsForTester`. `CredentialsForTester` implementuje rozhraní frameworku

SpecsForMvc `IHandleAuthentication`. Zde jsou v povinné metodě `Authenticate` definované akce za pomoci Selenia nutné k přihlášení a autentizaci uživatele. Vzhledem k tomu, že se přihlašovací údaje ukládají do cookies, je přihlašování provedeno jen při prvním spuštěném testu. Po konfiguraci je vzápětí aplikace nastartována a začínají probíhat samotné testy.

Všechny testy webové aplikace mají definovanou jednotku SUT jako třídu `MvcWebApp`, náležející frameworku SpecsForMvc. Ta umožní jednoduše procházet a přistupovat k webové aplikací.

Závěr

NOTIFIT by se časem mohl stát užitečnou pomůckou vývojářů, vytvářejících software pro mobilní zařízení. Notifikace jsou v dnešní době velmi populární už jen kvůli tomu, že dokážou aplikaci udržet při životě déle, než by si je uživatel bez připomenutí nechal ve svém zařízení. Myšlenka stmelit a zjednodušit tento krok má potenciál především v nejednotném rozhraní, které jednotliví výrobci nabízejí. Vývojář by se již nemusel obávat tolik přechodu na další platformy, a mohl by jednodušeji rozšířit své portfolio.

Do budoucna by se měl v aplikaci objevit jednotný model či struktura pro definování notifikace. Na pozadí by je pak NOTIFIT měl převádět do konkrétních payloadů stanovených jednotlivými notifikačními službami. Vývojářům by se tak zpřístupnilo webové API, které by mohli volat jednoduše ze svého serveru v jednotné podobě. Nemuseli by se starat o autorizace, a vytvořila by se tak účinná vrstva abstrakce.

Je také plánováno rozšíření, kdy uživatelé budou moci tvořit týmy a členům přidělovat určitá oprávnění. Tak by mohlo vzniknout lepší rozdělování práce, kdy jeden člověk notifikace vytváří a jiný například analyzuje posbíraná data.

V další fázi využije NOTIFIT prvku gamifikace. S předpokladem, že mnoho vývojářů má rádo žánr RPG her, bude pro udržení a motivaci vývojářů u NOTIFITu, zaveden systém odměn a získávání zkušeností či odznaků. Odměňování bude fungovat v závislosti na tom, kolik notifikací poslali a podle statistik které budou nést informace, jak se jeho aplikacím daří v porovnání s ostatními.

Literatura

[1] Minář, P.: Push Notifikace [online]. [Cit. 2015-12-01]. Dostupné z: www.jaknait.cz/co-je/push-notifikace/

[2] Hlavenka, J.: Webové servery: jeden nebo druhý [online]. červenec 1998, [Cit. 2015-12-01]. Dostupné z: http://www.zive.cz/clanky/webove-servery-jeden-nebo-druhy/sc-3-a-3548/default.aspx

[3] Bernard, B.: Úvod do architektury MVC [online]. červen 2009, [Cit. 2015-12-01]. Dostupné z: https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[4] Adaptive and interactive toast notifications for Windows 10 [online]. Dostupné z: https://blogs.msdn.microsoft.com/tiles_and_toasts/2015/07/02/adaptive-and-interactive-toast-notifications-for-windows-10/

[5] HTTP Connection Server Reference [online]. Dostupné z: https://developers.google.com/cloud-messaging/http-server-ref

[6] The Remote Notification Payload [online]. Dostupné z: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/TheNotificationPayload.html

[7] Entity Framework [online]. Dostupné z: https://msdn.microsoft.com/en-us/data/ef.aspx

[8] LINQ (Language-Integrated Query) [online]. Dostupné z: https://msdn.microsoft.com/en-us/library/bb397926.aspx

[9] Pranav Rastogi, T. D., Rick Anderson; Galloway, J.: Introduction to ASP.NET Identity [online]. říjen 2013, [Cit. 2016-04-11]. Dostupné z: http://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity

[10] Pecinovský, R.: *Návrhové vzory*. Computer Press, první vydání, 2007.

[11] Malý, M.: REST: architektura pro webové API[online]. březen 2009, [Cit. 2016-04-18]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

Příloha **A**

Seznam použitých zkratek

LINQ Language-Integrated Query

UTC Coordinated Universal Time

XML Extensible markup language

REST Representational State Transfer

URI Uniform Resource Identifier

EF Entity Framework

CRUD Create, Read, Update, Delete

GUID Globally Unique Identifier

GCM Google Cloud Messaging

APNs Apple Push Notification Service

WNS Windows Push Notification Services

API Application Programming Interface

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS