Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Sběr fingerprintů s využitím QR kódů a principů gamifikace

Bakalářská práce

Autor: Lukáš Chvátal
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing., Filip Malý, Ph.D.

Hradec Králové duben 2018

Prohlášení:

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

V Hradci Králové dne 22.4.2018 Lukáš Chvátal

Poděkování:

Děkuji vedoucímu bakalářské práce panu doc. Ing. Filipovi Malému, Ph.D. za přínosné konzultace, rady a připomínky během zpracování.

Anotace

Tato bakalářská práce se věnuje problematice sběru fingerprintů bezdrátových sítí s využitím QR kódů a principů gamifikace. Cílem je návrh a analýza systému tohoto záměru. Nejdříve je objasněn samotný pojem fingerprint, poté následuje definování a nastínění řešení problémů spojených s touto problematikou.

Teoretická část se zabývá detailnějším pohledem na jednotlivé komponenty, jež jsou stavebními kameny systému. Jsou zde sbírány poznatky pro praktickou část, ve které je následně sestavena analýza požadavků. Na základě stanovených požadavků je poté proveden návrh systému složený z několika komponent a všemi náležitostmi. Krom požadavků se praktická část věnuje i existujícím projektům podobného zaměření pro možné srovnání.

Annotation

Title: Collecting fingerprints using QR codes and gamification principles

This bachelor thesis deals with the problem of collecting wireless network fingerprints using QR codes and principles of gamification. The main purpose is to analyze and design the system of this intention. First, the term fingerprint itself is clarified. Following is the definition and outline of the problems associated with the matter of fingerprint collection.

The theoretical part deals with a more detailed view of individual components, which are building blocks of the system. With purpose to gain knowledge for the practical part, in which the requirements analysis is proposed. Based on the requirements a design of the system consisting of several components and all the requisites is made. Besides the requirements, the practical part explores existing projects of similar focus for a possible comparison.

Obsah

Seznam obrázků

Seznam tabulek

Seznam kódů

1 Úvod

Pod pojmem fingerprint (dále otisk) si člověk představí fyzický otisk prstu, ten ale naším zájmem není, předmětem práce je digitální otisk bezdrátových sítí. Hlavním cílem práce je analýza a návrh systému pro sběr těchto otisků v Česku. Účel sběru může představovat vizualizace pokrytí určité oblasti bezdrátovými sítěmi nebo sledování změn v konektivitě na časové ose.

Při návrhu takovéhoto systému nastává několik problémů, které je nutné adresovat. Efektivita jejich řešení je klíčová pro celkový úspěch systému. Nyní si jednotlivé problémy uvedeme a v následujících kapitolách se jim budeme věnovat.

Prvním problémem je zvolení zdroje otisků. Potřebujeme zařízení s dostatečnými možnostmi bezdrátové konektivity pro co možná nejobsáhlejší otisky, které je zároveň dostupné masám. Ideálním kandidátem jsou mobilní zařízení splňující obě kritéria. Zaměříme se pouze na mobilní zařízení s platformou Android, pokryjeme tak největší uživatelskou základu a zároveň nemusíme navrhovat multiplatformní aplikaci.

Druhý problém je návrh služby na uchování a možné zpracování otisků. Měla by být centralizovaná a schopná příjmu velkého objemu informací s minimální odezvou. Adekvátní je použití webového API. O většinu výpočetních operací se tak postará server, což ve spojení s bezestavovou komunikací představuje nízké výkonnostní nároky na mobilní zařízení.

Třetí a největší problém je, jak motivovat vlastníky mobilních zařízení, aby nám poskytovali otisky. Hrstka samaritánů by se našla, to však není dostatečné pro smysluplný výsledek. Zde nastupuje na scénu technika zvaná gamifikace, která není snadná na implementaci, ale dokáže být mocným motivačním nástrojem. Podporou gamifikace jsou pak QR kódy sloužící jako prostředek pro směrování a přidání hodnoty uživateli. Požadovaným cílem vlastníka zařízení je skenování QR kódů v patřičné reprezentaci, při kterém zároveň generuje otisky pro naše účely.

2 Otisk

Jak již bylo řečeno, otiskem v kontextu této práce není myšlen doslovný otisk prstu, nýbrž digitální otisk bezdrátových sítí. Může mít různorodou strukturu záležící především na zdrojovém zařízení. Platforma Android je pestrá, a ne všechna zařízení nabízí stejné prostředky pro získání informací. Základní údaje jako dostupné Wi-Fi a mobilní sítě by však měly být dostupné globálně.

Velká část mobilních zařízení podporuje i Bluetooth konektivitu. Nicméně se zde vyskytuje otázka, zda je vhodné takové informace sbírat. Problémem je mobilita těchto zařízení, v případě sběru je žádoucí jejich filtrace podle typu. Vynechat mobilní zařízení, jako bezdrátová sluchátka, která jsou neustále v pohybu, a zaměřit se pouze na stacionární.

Užitečné pro možné zpracování je zahrnutí údajů o zařízení. Kupříkladu výrobce zařízení, označení modelu a ostatní neosobní údaje, které jsou žádoucí pro kategorizaci otisku a zároveň nám téměř nic neprozradí o majiteli zařízení.

Dalšími nezbytnými údaji je lokace zařízení a časová značka v okamžiku sběru otisku, což nám pomůže ve dvou případech. Prvním je opět kategorizace a druhým, více důležitým, je validace otisku. Můžeme tak snadno ověřit, že otisk pochází z místa a času, ze kterého opravdu má.

Přesná struktura bude určena až v druhé části práce, kde se hlouběji podíváme na možnosti zařízení.

2.1 QR kód

Neboli **Quick Response** kód je maticový čárový kód určený k počítačovému zpracování. Standardně může obsahovat čísla, písmena, binární data a znaky kanji. Umožní nám tak vložení vlastního obsahu a identifikátoru. QR kód se stal velice oblíbeným pro jeho rychlou čitelnost i ve špatných podmínkách a velkou kapacitu oproti alternativám. Obvykle je složen z černých čtverců uspořádaných ve čtvercové mřížce na bílém pozadí.[1] Významy jednotlivých prvků QR kódu jsou stručně vysvětlené na obrázku 1 (Struktura QR kódu).

Využití

Původním účelem v roce 1994 po vynalezení japonskou společností **Danso Wave** bylo sledování postupu na výrobní lince v automobilovém průmyslu. Dnes se QR kódy využívají ve více odvětvích. Sledování postupů a životního cyklu produktu však zůstalo jedním z hlavních uplatnění. Nové oblasti představují především identifikaci, správu dokumentů a marketing. Možnými příklady použití z každodenního života jsou:

- **Přechod na URL** – Billboardy a ostatní marketingové struktury často obsahují webové adresy uvedeného produktu či služby. Pokus o zapamatování si této adresy nebo její manuální zadání do prohlížeče některé odradí. QR kód nabízí efektivnější způsob v podobě vnořené URL. Většina skenovacích aplikací dokáže automaticky uživatele přesměrovat na dané URL. Tato vlastnost může být prospěšná i pro náš návrh.
- **Platby** – Od roku 2012 je v Česku možné použití formátu **Short Payment Descriptor** pro pohodlné sdílení platebních informací. Uživatel bankovní aplikace vytvoří požadavek na platbu, ten následně odešle ve formě QR kódu. Po jeho oskenování je automaticky vytvořen příkaz k platbě pouze s nutností autorizace.[1]

Kapacita

Je jedna z hlavních předností QR kódu a odvíjí se od použité verze, typu obsahu a úrovně korekce chyb. Verzí zde není myšleno klasické verzování jako vývoj v čase, ale velikostní verze v podobě rozměrů kódu. Ta může nabývat hodnot od 1 do 40. Čím větší verze, tím větší rozměr QR kódu a jeho maximální kapacita.[1]

Tabulka 1: Maximální počty znaků při úrovni korekce L

Zdroj: [1]

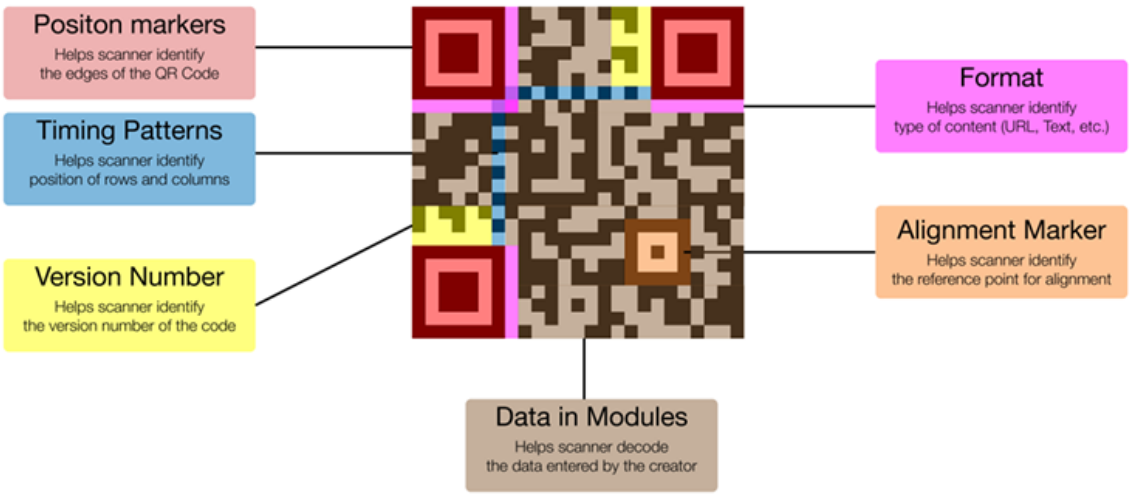
Typ obsahu	Počet znaků	Možné znaky
Číselný	7 089	0–9
Alfanumerický	4 296	0–9, A–Z, mezera, $, %, *, +, -, ., /, :
Binární	2 953	Sada ISO 8859-1
Kanji	1 817	Sada JIS X 0208

Korekce chyb

QR kód je robustní díky možnosti odstranění chyb skupinou algoritmů Reed–Solomon pro korekci kódů. Při generování je zvolena úroveň korekce, která má přímý dopad na výslednou kapacitu. Nižší úroveň znamená větší kapacitu, ale menší schopnost korekce. Jsou definovány čtyři úrovně:

- **Level L (Low)** – Korekce až 7 % QR kódu.
- **Level M (Medium)** – Korekce až 15 % QR kódu.
- **Level Q (Quartile)** – Korekce až 25 % QR kódu.
- **Level H (High)** – Korekce až 30 % QR kódu.

Vysoká schopnost korekce umožňuje zanesení úmyslných chyb, což může představovat doplnění QR kódu vlastním logem. Pro stroj stále čitelný a pro člověka přívětivější.[1]



Obrázek 1: Struktura QR kódu
Zdroj: https://scanova.zendesk.com

3 Android

Je open-source mobilní operační systém vyvíjený společností **Google** postavený na upraveném Linuxovém jádře. Dříve byl primárně určen pro mobilní telefony, dnes pohání širší spektrum zařízení. V jeho popředí stojí velice aktivní komunita, díky které si neustále drží náskok na trhu.[2] Téměř 75 % všech mobilních telefonů běží právě na Androidu.[3] Poslední verzí je Android **Oreo**, již osmá v pořadí, s API levelem[1] 27 a vydaná v druhé polovině roku 2017.[4]

Android platforma je složena ze čtyř hlavních úrovní:[2]

- **Jádro** – Představuje rozhraní mezi hardwarem a softwarem. Provádí základní funkcionality jako správu paměti, procesů, napájení a konektivity.[2]
- **Nativní knihovny** – Jsou knihovny napsané v jazyce C nebo C++ nabízející různé služby, příkladem může být WebKit pro prohlížení webu. Velká část pochází z open-source komunity.[2] Jejich použití pro základní účely není nutné.[5]
- **Android Runtime** – Nebo také **ART**, je prostředí, ve kterém běží aplikace. Řídí se konceptem AOT (Ahead-of-time), aplikace je kompilována při instalaci, což přináší výhody ve formě rychlosti, lepší alokace paměti, nové profilovací a debugovací techniky atd., oproti předchůdci **Dalvik**, který užíval JIT (Just-in-time) a kompilaci při běhu.[6]
- **Aplikační framework** – Je sada knihoven pro vývoj aplikací. Umožnují budování uživatelských prostředí, přístup k službám a ovládání ostatních funkcionalit zařízení. Například `AcitvityManager` spravující životní cyklus aktivit nebo `LocationManager` poskytující přístup k GPS službám.[2]
- **Aplikační vrstva** – Je tvořena samotnými aplikacemi, předinstalovanými i třetích stran.[2]

Distribuce a instalace aplikací na platformě Android je řešena pomocí archivního formátu **APK** (Android Application Package). APK je vytvořeno při kompilaci a

[1] API level představuje číselné označení sady dostupných funkcionalit pro vývojáře.

[2] Nativní knihovny a Android Runtime sdílí úroveň.

obsahuje všechny potřebné prvky pro spuštění aplikace, tj. zdrojový kód, zdroje, manifest a nativní knihovny.

Každá Android aplikace se skládá z několika hlavních stavebních prvků, které jsou její nedílnou součástí. Jde o aktivitu, fragment, pohled, manifest, zdroje a konstanty.

K dispozici máme však daleko více prvků. Jedná se ale spíše o prvky bez nutnosti použití zajišťující především operace aplikace na pozadí. Podíváme se jen na některé vybrané, konkrétně služby, content providery a broadcast receivery.

3.1 Aktivita

Reprezentuje jednu obrazovku aplikace s uživatelským rozhraním. Většina aktivit provádí interakce s uživatelem a je v režimu celé obrazovky. Aplikace je složena z více aktivit, které jsou uchovávány v zásobníku.[2]

Životní cyklus aktivity

Každá aktivita má svůj životní cyklus, který začíná jejím vytvořením a přidáním na vrchol zásobníku a končí odebráním. Sledování změn v životním cyklu je možné pomocí callbacků vyvolávaných při přechodech mezi stavy.[7]

Callbacky aktivity

Jsou metody s různými účely. Můžeme v nich řídit chování aktivity v jednotlivých stádiích životního cyklu. Jejich posloupnost je znázorněna na obrázku 2 (Diagram životního cyklu aktivity (vlevo) a fragmentu).

- **OnCreate** – Je první a typicky se zde vytváří pohled a inicializují prostředky.
- **OnStart** – Těsně před zobrazením pohledu uživateli. Aktivita přejde do stavu `started` a provádí se zde kód spravující pohled.
- **OnResume** – Aktivita je na popředí a připravena na interakci s uživatelem. Její stav se změní na `resumed` a zůstává v něm, dokud neztratí focus. Tento callback je volán při každém přechodu do popředí.
- **OnPause** – Aktivita ztratila focus, počítáme však s jeho znovuzískáním. Stav je `paused` a používá se většinou pro uvolňování systémových zdrojů a přístupů k senzorům, které je po pokračování opět nutné alokovat.

- **OnStop** – `Stopped` stav, aktivita již není na popředí, je překryta jinou nebo doběhla. Provádí se zde výpočetně náročné ukončující operace a uvolnění zdrojů nepotřebných na pozadí. Stav pohledu je uchován, tj. uživatel nepřichází o informaci napsanou v textovém poli.
- **OnRestart** – Když se aktivita vrací do popředí, třeba po navigaci zpět.
- **OnDestroy** – Poslední callback aktivity před úplným odstraněním ze zásobníku. Uvolňují se zde zbylé prostředky. Může být vyvolán i při změně orientace obrazovky, v tom případě se automaticky vytvoří nová aktivita.[7]

Vytvoření aktivity

Startovací aktivita uvedená v manifestu je automaticky vytvořena při spuštění. Vytvoření ostatních je již na nás a provádí se z aktivity nebo fragmentu.

Máme na výběr ze dvou metod. První je `StartActivity`, která se používá pro aktivity bez návratové hodnoty. Druhá je `StartAcitivtyForResult` s návratovou hodnotou určovanou uvnitř aktivity metodou `SetResult`. Má Integer atribut `requestCode` na identifikaci volání. Návratovou hodnotu získáme callbackem `OnActivityResult`, kde dostaneme `requestCode`, a můžeme tak ověřit, že výsledek je opravdu pro nás.

Obě metody mají parametr **Intent**, což je abstraktní popis požadované operace. Umožňuje předávání parametrů mezi komponentami. Vlastní parametry přidáme metodou `PutExtra` a následně získáme metodou `GetExtra`.[7]

3.2 Fragment

Je část obrazovky uvnitř aktivity. Fragmenty umožnují znovupoužitelnost a mohou být kombinovány. Jsou to v podstatě moduly, které mají vlastní životní cyklus s možností přidání a odebrání v aktivitě za běhu.[2]

Životní cyklus fragmentu

Stejně jako aktivita má fragment svůj životní cyklus, ten je ovšem závislý na jeho hostovi. Pokud aktivita obdrží `OnStop`, tak všechny její vnořené fragmenty také.[2]

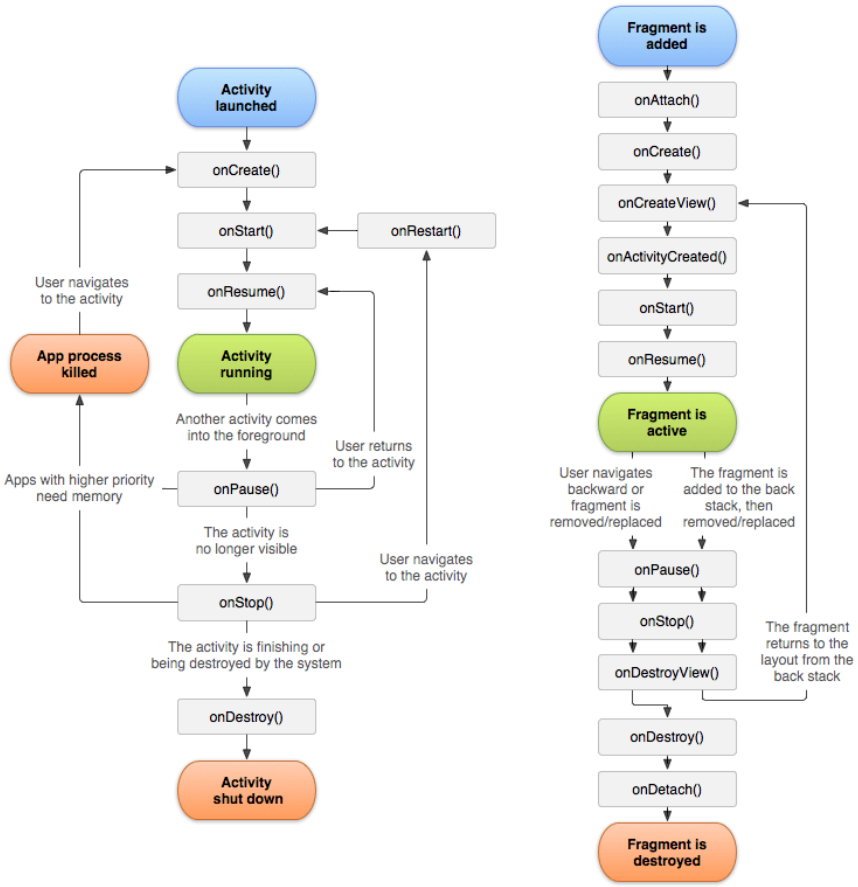
Callbacky fragmentu

Část callbacků vynecháme, jelikož je podobná těm z aktivity. Jejich posloupnost je znázorněna na obrázku 2 (Diagram životního cyklu aktivity (vlevo) a fragmentu).

- **OnAttach** – Callback po asociaci fragmentu s aktivitou.
- **OnCreateView** – Před prvním vykreslením pohledu, který se zde vytváří a následně vrací. V případě fragmentu bez uživatelského rozhraní vracíme null.
- **OnDestroyView** – Vyvolán po odebrání pohledu vytvořeného v `OnCreateView` z fragmentu. Provede se i v případě, když původní pohled byl null.
- **OnDetach** – Poslední callback po odejmutí fragmentu z aktivity.[8]

Správa fragmentů

Provádí se v aktivitě skrze transakce `FragmentManageru`. Fragmenty jsou uchovávány v zásobníku, který modifikujeme metodami `Add`, `Remove` a `Replace`. Pro dokončení transakce zavoláme metodu `Commit`.[8]



Obrázek 2: Diagram životního cyklu aktivity (vlevo) a fragmentu

Zdroj: [7] (vlevo) a [8]

3.3 Pohled

Představuje všechny prvky uživatelského prostředí. Tyto pohledy jsou organizované ve `ViewGroup`, což je neviditelný kontejner držící pohledy a ostatní skupiny pohledů. Pohledy a jejich skupiny je možné vytvářet deklarací v XML nebo programaticky v kódu. Obecně je preferována první varianta, oddělí se tak návrh uživatelského pohledu a aplikační logika, v praxi je ale často potřeba obou.[2] Android v základu poskytuje spoustu pohledů, není tak nutné znovu vymýšlet kolo a můžeme rovnou vytvořit textové pole pomocí `TextView`.

Pohledy jsou umístěny v hierarchii nazývající se **layout**. Jde o potomka `ViewGroup` pomáhajícího se strukturováním uživatelského prostředí. Existuje více druhů layoutů a každý nabízí jiné možnosti pozicování vnořených pohledů. Každý pohled v layoutu může mít id, přes které se dá následně referencovat. Pro kolekce dat se používají takzvané **adapter layouty**. Stejně jako pohledy mohou být vytvářeny deklarativně v XML nebo programaticky.[2]

3.4 Manifest

Je XML konfigurační soubor, který je součástí každé aplikace a obsahuje informace potřebné pro její kompilaci a spuštění. Povinně obsahuje následující prvky:

- **Jméno balíčku** – Většinou jde o jmenný prostor z kódové báze. Využívá se pro lokalizování prvků při kompilaci. Po kompilaci je vyměněno za unikátní id sloužící pro identifikaci aplikace v systému a obchodu Google Play.
- **Komponenty** – Neboli seznam všech aktivit, služeb, broadcast receiverů a content providerů. Každá komponenta nese svůj název společně s jmenným prostorem, ve kterém se nachází, dále může mít různé konfigurační parametry.
- **Oprávnění** – Jde o seznam všech oprávnění potřebných pro přístup k chráněným částem Android platformy, které aplikace vyžaduje.
- **Kompatibilita** – Je hardwarová a softwarová specifikace požadovaných funkčností cílového zařízení. Uvádí se zde například minimální API level.[9]

3.5 Zdroje a konstanty

Při vývoji aplikace je zapotřebí i multimediálních zdrojů jako obrázky nebo audio soubory. Ty jsou uloženy ve vyhrazených složkách a přistupuje se k nim přes id generované při kompilaci.[2] Máme k dispozici i vytvoření globálních konstant v XML souborech, které se využívají na deklaraci barev, textů, konfigurací apod.

3.6 Služby

Jsou komponenty umožňující provádět operace, které běží po delší dobu a bez vazby na uživatelské rozhraní. První možností je služba **na popředí**, která je postřehnutelná uživatelem a musí zobrazovat notifikaci. Třeba přehrávání hudby pokračující i po tom, co uživatel přeruší interakci s aplikací. Druhou možností je služba **na pozadí**, o které uživatel nemusí vědět, že probíhá, například zápis do úložiště. Služby se dále dělí podle způsobu spuštění a průběhu:

- **Started** – Služba běžící nezávisle na komponentu, která ji spustila. Může tak běžet neomezeně dlouho, i po zničení jejího spouštěče. Po dokončení svého procesu by se měla sama zničit, může být však zastavena jakoukoliv komponentou. Started služby nemají návratové hodnoty.
- **Bound** – Služba závislá na komponentě, ke které je vázaná. Komponenty s ní tak mohou interagovat, dokonce i mezi více procesy. Na rozdíl od started umožnuje návratové hodnoty. Bound služba běží pouze dokud má nějakou vazbu, po přerušení všech vazeb je zničena.[10]

3.7 Content providery

Zpřístupňují centrální Android úložiště aplikacím, jde tedy o způsob sdílení dat mezi aplikacemi. Omezení rozsahu sdílených dat lze definovat v manifestu. Pro získání dat je nutné mít požadované oprávnění a znát identifikátor provideru.[2]

Příkladem vestavěného content provideru je `UserDictionary` sloužící pro ukládání a přístup k textovým predikcím. Nabízí tak možnost sdílení těchto informací mezi různými klávesnicemi. Krom samotného slova se uchovává informace o frekvenci jeho použití a příznak jazyku.[11]

3.8 Broadcast receivery

Jsou komponenty provádějící různé akce na základě událostí. Většinou jde o systémové události jako slabá baterie, zapojení do elektrické sítě nebo zapnutí Wi-Fi. Události mohou být vyvolány i aplikacemi, například při úpravě kontaktu pro informování všech aplikací mající broadcast receivery na kontakty o této změně.[2]

3.9 Xamarin

Je sada vývojářských nástrojů postavena na frameworku **Mono** pro tvorbu mobilních i desktopových aplikací. Mono je open-source multiplatformní implementace .NET frameworku zahrnující C# kompilátor a CLR (Common Language Runtime) prostředí, ve kterém aplikace nativně běží a mají tak k dispozici téměř všechny Android funkcionality.[2]

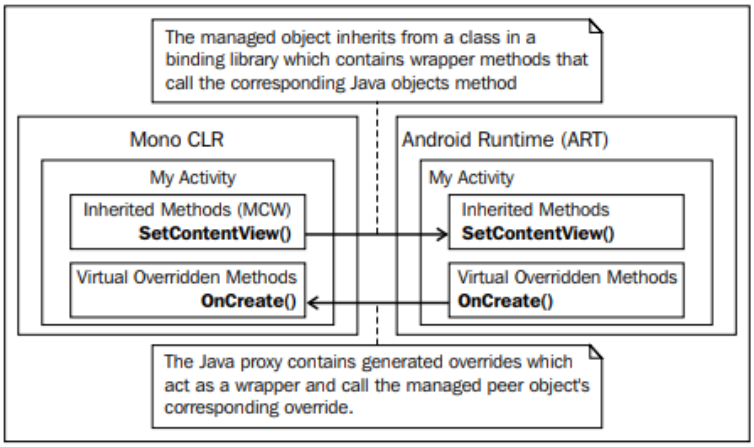
Od roku 2016 je Xamarin vlastněn **Microsoftem** a k distribuován zadarmo.[12] Využívá .NET framework, programovací jazyk je tedy C# a máme k dispozici většinu jeho funkcionalit. Podporované platformy pro vývoj jsou Android, iOS, macOS a Windows.

Xamarin Forms dále umožnuje do určité míry psaní multiplatformních aplikací. Obecné části jako uživatelské rozhraní stačí napsat jednou, poté jen vykompilovat pro Android, iOS nebo Windows Phone.[2]

Na výstupu je vytvořeno klasické APK obsahující assemblies z C# kódu pro CLR a všechny potřebné Mono runtime a Android nativní knihovny. To se negativně projeví na velikosti výsledné APK a může být viděno jako drobná nevýhoda.[13]

ART a CLR

V úvodu kapitoly je uvedeno, že aplikace běží v ART a nyní uvádíme CLR, jak to tedy je? Velice snadně, běží souběžně v obou. Pro jejich spolupráci je zapotřebí **JNI**, což je framework umožňující vzájemné volání Java kódu (ART) a nativního kódu (CLR). K tomu je nutné mít takzvané binding knihovny, ve kterých jsou C# třídy mapované na Java třídy. Při vytvoření C# třídy dědící z bindované třídy je vygenerována proxy Java třída s úkolem vyvolávat přetížené metody v C# třídě.[2]



Obrázek 3: Diagram spolupráce CLR a ART
Zdroj: [2]

API design

Snahou Xamarinu je vytvořit vývojové prostředí bindované na Android API, které žádnému .NET vývojáři nebude cizí a zároveň se drží původních konceptů. Ve výsledku tak není problém následování Java Android dokumentace a ukázek a jejich aplikování v C#. Místo typických zapouzdřených Java properties můžeme použít klasické C# properties, stejně tak delegates namísto Java event listenerů.[14]

Proč Xamarin

Hlavním důvodem vybrání Xamarinu je preference .NET ekosystému. Nabízí se zde také možnost zahrnutí mobilní aplikace a web API v jednom projektu díky jeho integraci ve formě doplňku pro **Microsoft Visual Studio**, což přináší výhody v podobě sdílení kódu pro datové modely apod. Vývoj je možný i v oficiálním IDE **Xamarin Studio**, které nově podporuje i emulaci iOS.

4 Webová architektura

Web je navržen okolo tří základních konceptů, které je nutné si před ponořením se do jeho funkcionality vyjasnit. Jde o zdroj, URI a reprezentace. V následujících kapitolách se na ně budeme často odkazovat.

Zdroj

Zdrojem se chápe vše, co má identifikátor (URI). Jde o konceptuální mapování na jednu či více entit. Entita může být prakticky cokoliv, PDF dokument, položka v databázi, chytré zařízení i jiný systém.[15]

URI

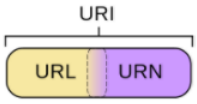
Neboli **Uniform Resource Identifier** je textový řetězec sloužící pro specifikaci zdrojů. Má následující strukturu: `schéma:hierarchická_část?dotaz#fragment`.[15] Příkladem může být: `http://www.web.cz/prispevky/42?jazyk=cs#diskuze`.

- **Schéma** – Určuje, o jaký druh URI se jedná a jakou syntax musí celé splňovat.
- **Hierarchická část** – Obsahuje identifikátor zdroje v rámci hierarchie.
- **Dotaz** – Je nepovinná část sloužící k bližšímu určení zdroje. Často ve formě `klíč=hodnota`. Jednotlivé parametry jsou oddělené ampersandem (&).
- **Fragment** – Je nepovinná část popisující sekundární zdroj na základě primárního, například určitou sekci v rámci příspěvku.[16]

URI vs. URL vs. URN

Společně s URI se často skloňují další dvě zkratky, URL a URN. Jaké jsou mezi nimi rozdíly? URI je nejobecnější, může popisovat zdroj jako takový, ale i způsob jeho dosažení.[17] V praxi se URI a URL používají často jako synonyma.[15]

- **URL** – Neboli **Universal Resource Locator** popisuje lokaci zdroje.
- **URN** - Neboli **Universal Resource Name** identifikuje zdroj jako takový a nesděluje jeho přesnou lokaci.[17] Například ISBN: `urn:isbn:978-1-449-33771-1`.



Obrázek 4: Eulerův diagram vztahu URI, URL a URN
Zdroj: [16]

Reprezentace a typ média

Reprezentací je myšlena struktura zdroje ve vybraném okamžiku měnící se dle potřeby. Při požadavku na zdroj je vrácena reprezentace tohoto zdroje.

Každá reprezentace má svůj typ média (také content-type), což je formát přenášeného zdroje mezi klientem a serverem. Skládá se ze dvou části, **top-levelu** popisující obecný formát zdroje a **subtypu**, který určuje specifický formát dat.[15] Například `image/jpeg`, což udává, že zdroj je obrázek, konkrétněji se ztrátovou kompresí a bez alfa kanálu.

API nejčastěji pracuje se strukturovanými daty. Využívá tedy primárně typy určené pro přenos objektů serializovaných na textový řetězec, který je snadno na druhém konci deserializovaný. Oblíbené typy jsou `application/json` a `application/xml`, které jsou snadno čitelné pro člověka i počítač. V praktické části použijeme JSON, pro jeho menší velikost v případě komplexních struktur společně se snazším a rychlejším parsováním.[18]

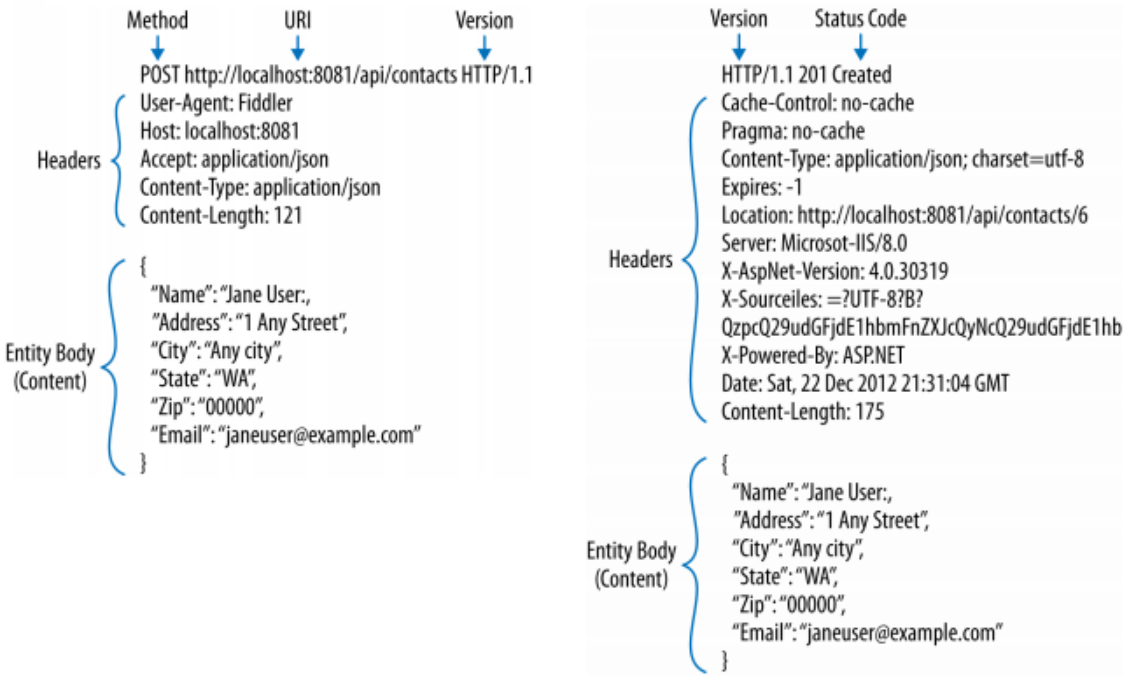
4.1 HTTP

Neboli **HyperText Transfer Protocol** je bezestavový protokol aplikační vrstvy určený pro přenos zdrojů mezi klientem a serverem. Je základním stavebním kamenem pro World Wide Web od samotného počátku v roce 1991.[19] Díky flexibilnímu designu dokáže do určité míry tolerovat prodlevy a zastaralá data. Toto umožňuje prostředníkům zasahovat do komunikace, což přináší výhody jako cachování, kompresi, routování apod.[15]

Komunikace je postavena na systému výměny zpráv ve formě požadavku a odpovědi. Klient odešle požadavek se všemi náležitostmi. V první řádce je uvedena metoda, URI požadovaného zdroje a verze protokolu. Následují volitelné hlavičky požadavku. Poslední je tělo požadavku, které je nepovinné a jde o obsah odesílané zprávy, například JSON řetězec.[20]

Po přijetí požadavku se server na základě první řádky rozhodne, zda je schopen ho obsloužit. Pokud ano, je zpracován na základě hlaviček a těla.[15]

V obou případech je odeslána odpověď, ta nese v první řádce verzi protokolu a status kód, dále volitelné hlavičky odpovědi a tělo.[21]



Obrázek 5: Příklad požadavku (vlevo) a odpovědi
Zdroj: [15]

4.1.1 Metody

Indikují požadovanou akci nad zdrojem. Občas se uvádějí jako HTTP verbs, tedy slovesa. Metody mají několik rysů, podle kterých se mohou kategorizovat. První takovou vlastností je **bezpečnost** neboli příznak, zdali se jedná o read only metodu neměnící stav systému. Druhou je **idempotence**, která říká, jestli při opakovaném zaslání identického požadavku docílíme stejného výsledku. Třetí a poslední je **cachovatelnost** určující, zda je možné uložení výsledku do mezi paměti a tím urychlit odpověď při dalším požadavku. HTTP poskytuje standartní sadu metod, zaměříme se pouze na některé z dostupných.[15]

- **GET** – Požadavek na získání zdroje. Krom návratu samotného zdroje by metoda neměla mít žádný jiný efekt. V praxi lze využít i pro zaslání zdrojů, ale jsme omezeni maximální délkou URL, jelikož zdroje jsou jeho součásti.

- **POST** – Požadavek na zpracování zaslaných zdrojů v těle požadavku. Může být použit prakticky na cokoliv, zdroje nejsou součástí URL, můžou tedy mít libovolnou velikost a být i binární.[22]
- **PUT** – Požadavek na vložení nebo přepsání entity, když víme URI této entity.
- **DELETE** – Požadavek na smazání entity na specifikovaném URI.
- **PATCH** – Požadavek na částečnou aktualizaci entity na specifikovaném URI.[15]

Tabulka 2: Vlastnosti HTTP metod
Zdroj: [15]

Metoda	Bezpečná	Idempotentní	Cachovatelná
GET	Ano	Ano	Ano
POST	Ne	Ne	Ne
PUT	Ne	Ano	Ne
DELETE	Ne	Ano	Ne
PATCH	Ne	Ano	Ne

4.1.2 Hlavičky

Jsou pole sloužící pro předání doplňujících informací v požadavcích a odpovědích. Podobně jako dotaz v URI mají podobu klíče a hodnoty. S vývojem HTTP jsou hlavičky rozšiřovány a máme možnost použít i vlastní. Mohou být rozdělovány do čtyř skupin podle jejich účelů:

- **Hlavičky zprávy** – Jsou společné pro požadavek i odpověď. Nesou informace o prostřednících (Cache-Control), samotné zprávě (Transfer-Encoding) a požadavku (Connection, Date).
- **Hlavičky požadavku** – Obsahují informace o požadavku (Host, Expect), klientovi (User-Agent, From), content negotiation (Accept), a kondicionálních požadavcích (If-Match).
- **Hlavičky odpovědi** – Jsou v nich uvedeny informace o zdroji (Allow, Server), kontrolní data (Age, Location), vybrané reprezentaci (ETag, Last-Modified) a změnách v autentizaci (Proxy-Authenticate).
- **Hlavičky reprezentace** – Zaměřují se tělo zprávy (Content-Type, Length, Location) nebo cachování (Expires).[15]

4.1.3 Status kódy

Každá odpověď musí mít status kód a popis výsledku v uživatelsky přívětivé formě indikující výsledek požadavku. Status kódy se klasifikují do pěti kategorií:

- **Informativní (1xx)** – Požadavek byl přijat a je zpracováván.
- **Úspěšné (2xx)** – Požadavek byl úspěšně doručen, porozuměn a akceptován.
- **Přesměrování (3xx)** – Další akce je nutná pro dokončení požadavku.
- **Klientská chyba (4xx)** – Požadavek není validní a nemohl být zpracován.
- **Serverová chyba (5xx)** – Server nebyl schopen zpracovat požadavek.[15]

Tabulka 3: Časté status kódy
Zdroj: [23]

Status kód	Význam
200 Ok	požadavek úspěšný
201 Created	nová entita vytvořena
202 Accepted	požadavek přijat a zatím nezpracován
304 Not Modified	přesměrování na objekt z cache
400 Bad Request	požadavek nečitelný a neměl by být opakován
401 Unauthorized	požadavek vyžaduje autentizaci
404 Not Found	požadovaný cíl nenalezen
408 Request Timeout	požadavek nevznesen v určeném časovém intervalu
500 Internal Server Error	server narazil na neočekáváný stav
503 Service Unavailable	server není schopen zpracovat požadavek

4.1.4 Content Negotiation

Jak bylo zmíněno v oddílu reprezentace a typ média, zdroj může mít několik reprezentací. Content negotiation umožňuje klientovi v požadavku specifikovat očekávanou reprezentaci skrze hlavičku `Accept`.[15] Pokud server není schopen poskytnout požadovanou reprezentaci zdroje, měl by odpovídajícím způsobem klienta informovat. Ve skutečnosti ovšem server může odeslat odpověď s jakoukoliv reprezentací, pak již záleží na klientovi, jak si s ní poradí.

4.1.5 Cachování

Je efektivní způsob urychlení komunikace v neustále rozrůstajícím se prostředí, kterým je Internet. Může být prováděno na straně klienta, například uložením odpovědí s CSS a JavaScript soubory v cachi prohlížeče a tím omezit často opakující se požadavky. Ale i serveru v podobě prostředníků, kteří vracejí cachované reprezentace zdrojů a snižují tak zatížení cílového serveru.[15] Jak bylo uvedené v tabulce 2 (Vlastnosti HTTP metod), využití cachování je možné jen s metodou GET.

Expirace

Cachovaná odpověď je brána za expirovanou, pokud její stáří přesáhne `max-age` uvedené v `Cache-Control` hlavičce nebo aktuální čas přesáhne expirační čas specifikovaný v `Expires` hlavičce.[15]

Validace

Po expiraci musí být odpověď znovu validována, tedy odeslána na server kondicionálním požadavkem a zkontrolována její aktuálnost. Pokud je validní, server odešle odpověď s aktualizovaným expiračním časem. V případě změn, server odešle novou odpověď s aktuální reprezentací, která vymění starou v cachi.[15]

4.1.6 Autentizace

HTTP nabízí několik způsobů, jak autentizovat klienty a omezit tak přístup k některým zdrojům nebo celým sekcím. Zaměříme se však pouze na metodu využívající tokeny, konktrétně Google Sign-In postavenou na OAuth 2.0.

OAuth 2.0

Je framework umožňující klientovi poskytnout svou identitu v nějaké službě (například Google+) třetí straně (API), aniž by jí sděloval své přístupové údaje. Klient zasílá pouze časově omezený token vystavený autentizačním serverem vybrané služby. Třetí strana tak nemusí uchovávat přístupové údaje klienta a stačí ji pouze ověřit validitu tokenu.[24] Tento přístup je vhodný zejména na mobilních zařízeních, kde pro klienta představuje ve většině případech jedno až dvě klepnutí.

4.2 Web API

Neboli **Application Programming Interface** je programové rozhraní pro webový server, ke kterému přistupují různorodí klienti skrze metody HTTP protokolu za účelem interakce se zdroji.[15]

Při tvorbě API se můžeme držet několika stylů, které uvádějí předpisy a nejlepší praktiky, aby výsledné API mělo slušně řečeno hlavu a patu, nejčastěji SOAP, RPC nebo **REST**. Budeme se snažit o návrh REST API, tudíž se zaměříme na tento styl.

Před pokračováním si musíme definovat základní pojmy, na které se budeme odkazovat. Tyto pojmy jsou obecné a používají se i mimo API, uvedeme si je ale v kontextu ASP.NET Core.

Routování

Je způsob určení požadované funkcionality z routy (cesty) pomocí routovacích tabulek. Po přijetí požadavku na API je nalezen odpovídající controller, který následně zavolá akci, kde je provedena aplikační logika a vrácena odpověď.[27]

Controller

Je třída, na kterou je mapována URL. Zpracovává příchozí požadavky a vrací zpět odpovědi. Základní konvence uvádí pojmenování množným číslem s příponou `Controller`. Controller pro správu uživatelů je tedy `UsersController`, jeho defaultní routa pak `/api/users`.

Akce

Je metoda controlleru, někdy také nazývaná jako akční metoda. Může být stejně jako controller mapována v URL. Podle základní konvence název neobsahuje příponu `Action`, ale jednotné číslo názvu controlleru. Akce pro aktualizaci uživatele je tedy `UpdateUser` s výslednou routou `/api/users/UpdateUser/94`, kde 94 je parametr ID uživatele.[3][28]

[3] Pokud je vypnuté Automatické mapování CRUD metod.

4.2.1 REST

Neboli **Representational State Transfer** je zdrojově orientovaný styl tvorby API uvádějící šest předpisů. Po jejichž splnění se API dá nazvat jako RESTful a považovat za snadno rozšiřitelné a udržitelné po dlouhou dobu.[15]

Klient-server architektura

Princip separování odpovědností klientů a serverů, což umožňuje snadnou rozšiřitelnost a nezávislý rozvoj s bezproblémovým nahrazením komponent na obou stranách.[25]

Bezestavovost

Každý požadavek obsahuje veškeré informace potřebné k jeho splnění. Server tedy není zatížen sledováním stavů všech svých klientů.

Jednotné rozhraní

Jde o definici rozhraní mezi klienty a servery zjednodušující interakce. Rozpadá se do čtyř kategorií:

- **Identifikace zdrojů** – Předmětem interakcí jsou zdroje s URI a reprezentací.
- **Sebe-popisující zprávy** – Každá zpráva obsahuje veškeré informace potřebné pro interakci.
- **Transformace zdrojů reprezentacemi** – Viz content negotiation.[15]
- **HATEOAS**[4] – Klient je schopen dynamicky objevovat veškerou funkcionalitu API skrze linky poskytované v odpovědích. Například při požadavku na stav bankovního účtu odpověď obsahuje linky na další možné funkcionality, jako je vklad, výběr, převod apod.[26]

Vrstvený systém

Komponenty jsou vrstveny na sebe a žádná tak nemá rozhled na celý systém. To umožňuje přidání vrstev mezi klienta a koncový server v podobě prostředníků, kteří poskytují služby jako cachování, kompresi apod.

[4] Neboli Hypermedia as the Engine of Application State.

Cache

Každý požadavek nese příznak, zda je jeho obsah cachovatelný.

Kód na požádání

Funkcionalita může být delegována na klienta zasláním spustitelného kódu. Nejčastěji ve formě JavaScriptu za účelem snížení zatížení serveru. Tento předpis přináší i určité nevýhody a je považován za volitelný.[15]

4.2.2 ASP.NET Core 2.0

Je open-source framework vyvíjený společností **Microsoft** pro tvorbu moderních webových aplikací. V druhé části práce bude využit při návrhu API komunikujícího primárně s mobilními zařízeními. Standardně se řídí REST stylem a ulehčuje tak následování jeho předpisů pro tvorbu **RESTful** API. Pro spuštění základního API není potřeba zdlouhavé konfigurace. Na druhou stranu umožňuje velkou míru customizace, pokud je potřeba. Nyní se podíváme na některé z hlavních rysů a předností frameworku.

Dekorace atributy

Nabízí handlery, které umožňují provedení kódu před samotným vstupem do controlleru. Dále pre a post-process filtry dovolující provedení kódu před akcí i po ní. Filtrů je k dispozici několik typů, třeba výjimkové, které pomáhají oddělit řešení neošetřených výjimek od kódu akce.

Patří sem i atributy pro definici nestandardního routování, kde je možné specifikovat HTTP metodu, verzování nebo samotnou routu. Například přidáním atributu `[HttpGet]` na akci říkáme, že může přijmout pouze GET požadavky.

Další možností je omezení na datový typ, minimální a maximální hodnotu, ale i regulární výraz přímo pro jednotlivé parametry routy. Ve výsledku může vypadat následovně: `[Route("NazevAkce/{id:min(10)}")]`.

Automatické mapování CRUD metod

HTTP metody jsou automaticky mapovány na akce controllerů podle jejich názvů. Pokud máme controller `Users` a v něm deklarovanou akci `GetUser`, při GET požadavku s routou `/api/users/42` bude tato metoda zavolána. To samé platí pro POST, PUT a DELETE požadavky.

Vestavěný content negotiation

Výsledek akce, co má návratovou hodnotu jako datovou strukturu je automaticky serializován do zvolené reprezentace. Standardně je vrácen JSON, není-li reprezentace v požadavku specifikována.

Globální zachytávání výjimek

Všechny neošetřené výjimky je možné zachytávat a řešit na jednom místě. K dispozici je i několik způsobů logování.

IActionResult

Pohodlný způsob zabalení výsledků akcí společně se status kódem a relevantními hlavičkami. Například vrácení metody Ok s parametrem nově vytvořené entity při POST požadavku, která vytvoří odpověď se status kódem `200 Ok` a `Location` hlavičkou odkazující se novou entitu.[29]

5 Gamifikace

Pro někoho pouhý marketingový „*buzzword*“[5], pro jiného mocný nástroj pro uchopení pozornosti, pohled na gamifikaci je velice subjektivní a jednoznačná definice není snadná. Svůj pohled na gamifikaci vyjádřilo v minulosti mnoho autorů z různých oblastí zaměření, na některé z nich se podíváme v následujících odstavcích. Nejdříve ale něco o počátcích tohoto termínu.

5.1 Počátky

Koncept gamifikace není ničím novým, ve vojenském odvětví se hry a simulace využívají pro trénink jednotek po stovky, ne-li tisíce let.[30]

Samotný termín gamifikace není tak starý, zásluhy na jeho vytvoření si připisuje programátor **Nick Pelling**, který ho použil v roce 2002 jako součást svého startupu[6] Conundra Ltd.[7], s cílem implementace herních prvků do komerčních elektronických zařízení. Tento pokus nebyl úspěšný a společnost byla zanedlouho zrušena pro nezájem ze strany zákazníků. Sám **Pelling** přiznává, že jeho vize gamifikace byla *„o desetiletí moc brzo“*.[31]

Gamifikace se do obecného povědomí dostala až v roce 2010, kdy byla chápána jako: *„začleňování sociálních a odměňovacích aspektů her do softwarového prostředí“*[32] a viděna jako lukrativní doplněk v pracovním prostředí pro motivaci zaměstnanců.

První společností, která začala nabízet implementaci gamifikace jako služby byla společnost **BunchBall** v roce 2007. Zanedlouho začali s gamifikací svých firemních procesů i giganti jako **Microsoft** a **IBM**.[33]

[5] Buzzword je slovo nebo fráze aktuálně v módě. Vyskytuje se často v marketingu pro oslnění laiků.

[6] Startup je nově založený či začínající podnikatelský záměr.

[7] Kopie webové stránky společnosti dostupná na: http://www.nanodome.com/conundra.co.uk

5.2 Definice

Samozvaný sériový podnikatel a veřejný řečník **Gabe Zichermann** ve své knize **Gamification By Design** popisuje gamifikaci jako:

„Proces využívající herního myšlení a herních mechanik k zapojení uživatelů a řešení problémů.“[30]

Dále tvrdí, že takovéto chápání gamifikace je „*silné a flexibilní, jelikož může být aplikováno na jakýkoliv problém, jenž může být vyřešen ovlivněním motivace a chování člověka*“.

Designer a výzkumník **Sebastian Deterding** nabízí definici na úrovni akademické půdy jako:

„Použití herních prvků a technik v neherních prostředích.“[34]

Zároveň zdůrazňuje, že může být dostačující využít pouze některé herní prvky, a aplikace není omezena na digitální technologie, přestože většina dnešních implementací má digitální podobu.[34]

Jeden z nejstarších průkopníků v oboru **Yu-kai Chou** nabízí definici z trochu jiného úhlu pohledu:

„Dovednost čerpání zábavných a návykových prvků z her a jejich následná aplikace v produktivních činnostech reálného světa.“[35]

Svůj přístup nazývá Human-focused design, návrh zaměřený na člověka, přesněji na jeho pocity, motivace a zapojení do děje, což je výrazná změna oproti Function-focused designu, který se soustředí na dosažení nejrychlejšího výsledku bez ohledu na lidskou motivaci dosažení cíle.

Důvod proč gamifikaci nazýváme gamifikací je takový, že herní průmysl byl první, který dokonale dokázal zabavit a motivovat člověka. Proč tedy nevyužít léta vývoje a zkušeností z her a neaplikovat je i do jiných prostředí pro lepší výsledky? Přesně o to se gamifikace snaží, o integraci herního myšlení a herních prvků v neherních prostředích pro získání a udržení zájmu uživatelů.[35]

5.3 Co není gamifikace

Gamifikace není hrou. Na to, aby byla klasifikována jako hra, nesplňuje dvě důležité věci. První je absence hratelnosti, gamifikací něčeho nevytváříme hru. Druhou je záměr, není totiž tvořena primárně pro zábavu.

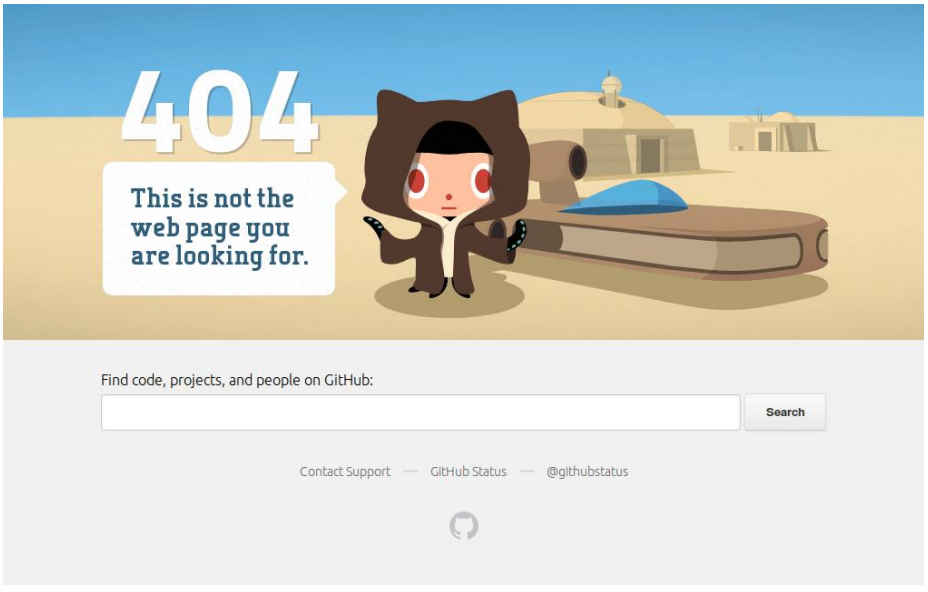
Gamifikace také není jen o bodech a odměnách, jak si někteří myslí. Ačkoliv to jsou její prvky, není to její jádro.[35] Více o jádru gamifikace v sekci jádro gamifikace.

5.4 Související pojmy

Krom hry bývá gamifikace často zaměňována i s dalšími pojmy. Na některé z nich se podíváme a najdeme umístění gamifikace mezi nimi.

5.4.1 Gameful/Playful design

Jde přikrášlení a vylepšení uživatelského požitku za pomoci herního myšlení. Tyto vylepšení nepřidávají na funkčnosti a většina uživatelů na ně nemusí vůbec narazit. Asi nejsnazší a zároveň velice populární aplikace playful designu poslední doby je customizovaná chybová stránka[8] pro web.[36]



Obrázek 6: Chybová stránka služby GitHub
Zdroj: https://github.com/404

[8] Chybová stránka slouží k informování uživatele o chybě, která nastala, nejznámější HTTP 404 v případě nenalezení souboru.

5.4.2 Serious Game/Simulace

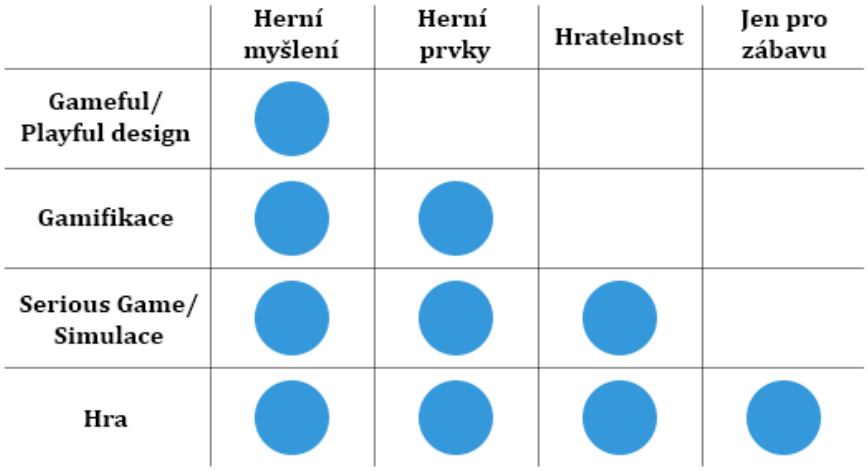
Vypadá a chová se jako normální hra a staví dokonce i na všech jejich prvcích. Je však vytvořena za určitým účelem a zábava to primárně není. Nejčastěji se využívá ke vzdělávání, výcviku nebo předání poselství, dokáže totiž přiblížit situace z reálného světa bez následků v něm. Na druhou stranu následky ve světě virtuálním bývají nehezké a potencionálně katastrofické. Spadají sem i hratelné simulace.[36]

Příkladem může být simulační hra od společnosti **IBM** s názvem **City One**[9] vydaná roku 2010. Cílem hry bylo navržení prosperujícího města a překonání výzev které je dnes sužují, jako je přelidnění, znečištění, neefektivní infrastruktura apod.[37] Hráč si tedy mohl vyzkoušet, jak je složité přijít s efektivním návrhem města, a přitom v průběhu nezničit život tisícům obyvatel toho reálného.

Vyskytuje se otázka, zda například populární počítačová hra podobného zaměření **SimCity** není také seriózní hrou. Odpověď je snadná, není, byla vytvořena s primárním účelem poskytnutí zábavy.

5.4.3 Hra

Obsahuje všechny elementy popisované v předchozích pojmech, je ale vytvořena primárně za účelem rekreace a zábavy.



Obrázek 7: Grafické znázornění rozdílů mezi pojmy
Zdroj: [36], vlastní zpracování

[9] Více informací na: http://www-01.ibm.com

5.5 Jádro gamifikace

Značný podíl gamifikovaných systémů je neefektivních, v některých případech může gamifikace mít dokonce i negativní dopad. Jak je to možné? Vždyť jsme si řekli, že hry dokonale zabavují a motivují člověka. Stačí použít nějaké z herních prvků a uživatelé zajisté neodolají! Takto gamifikace bohužel nefunguje a nesmíme zapomenout na fakt, že všechny hry nejsou dokonalé, existují i špatné a není jich málo.[35]

Dle **Yu-kai Choua** bychom neměli začínat gamifikaci otázkou „Co?“, ale „Proč?“ a „Jak?“. Tedy jak apelovat na základní lidské motivátory, které nás ženou kupředu. **Chou** přišel s frameworkem **Octalysis**, který staví na osmi takovýchto motivátorech v rámci gamifikace.

Epic Meaning & Calling

Uživatel se domnívá, že byl vyvolen pro daný úkon, nebo pracuje vůči cíli většímu, než je on sám. Příznakem tohoto motivátoru je uživatel ochotný obětovat svůj čas pro dobro všech ostatních.

Empowerment of Creativity & Feedback

Uživatel potřebuje vykazovat kreativitu, ale hlavně vidět její výsledky a dostat zpětnou vazbu. Ideálem je řešení umožňující uživateli tvorbu vlastního obsahu, zabaví sám sebe a zároveň i ostatní, pokud se rozhodne obsah sdílet.

Social Influence & Relatedness

Tento motivátor zahrnuje všechny sociální elementy pohánějící člověka, příkladem může být přátelství, konkurence či závist. Člověk má potřebu obklopovat se lidmi, místy nebo událostmi, ke kterým má nějaký vztah. Uživatel je motivován dosáhnout úspěchů ostatních uživatelů z jeho okolí.

Unpredictability & Curiosity

Touha po nahlédnutí do křišťálové koule a zahlédnutí budoucnosti. Nejasno motivuje uživatele k častému přemýšlení o možných scenáriích, co kdyby.

Avoidance & Loss

Člověk se instinktivně vyhýbá negativním událostem, což může představovat ztrátu krátkodobého postupu či přiznání si, že vše, čeho doposud docílil, bylo zbytečné, protože chce skončit. Uživatel si ukončení tedy pořádně rozmyslí, v některých případech je snazší pokračování než přiznání porážky. Časově omezené příležitosti sem také spadají, jelikož vyvolávají pocit možné ztráty při oddálené reakci.

Scarcity & Impatience

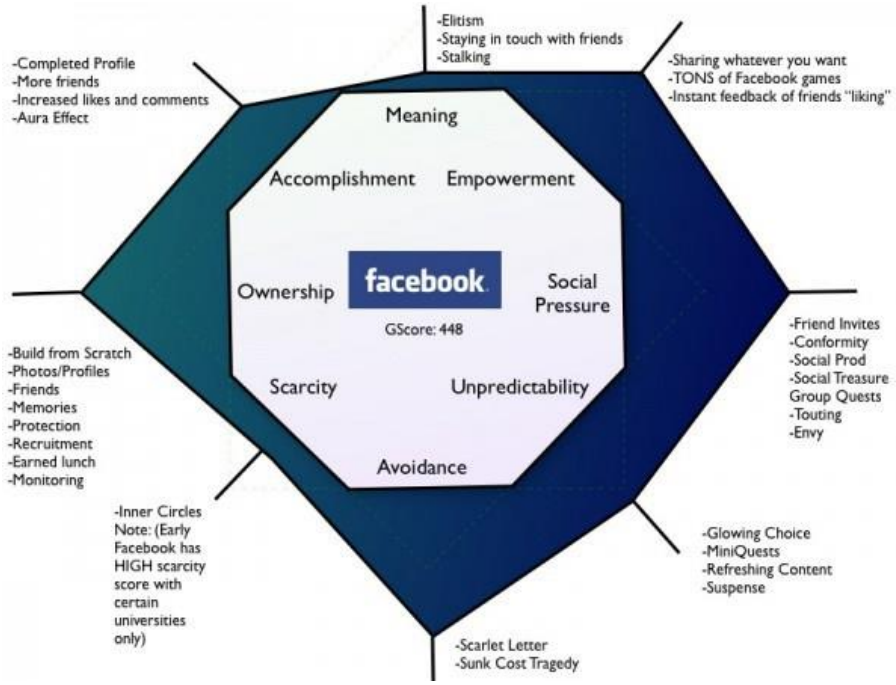
Uživatel touží po tom, co nemůže mít. Příkladem je otevření truhlice s odměnou jen jednou denně, uživatel tak celý den přemýšlí, co dostane z další.

Ownership & Possession

Uživatel je motivován pocitem vlastnictví a má zájem toho vlastnictví stále zlepšovat a rozšiřovat. Může se jednat o měny zahrnuté v systému, virtuální, ale i fyzické objekty. Uživatel trávící čas „*šperkováním*" svého portfolia prohlubuje tento pocit.

Accomplishment & Development

Dělání pokroků, rozvíjení dovedností a eventuální překonání výzev. Výzva musí být přiměřená, snadno získaný odznak bez výzvy nemá žádný význam.[38]



Obrázek 8: Aplikace frameworku na sociální síť Facebook

Zdroj: http://yukaichou.com

5.6 Herní prvky

Herní prvky byly v předchozích částech mnohokrát zmíněny, je na čase se podívat, co jsou vlastně zač. Jde o nástroje s původem ve hrách, které při správném použití slibují smysluplnou odezvu od uživatelů. Zaměříme se jen na pár základních, které budou aplikovány v druhé části práce.

5.6.1 Body

Jsou nedílnou součástí gamifikace a měly by být zahrnuty i v případě, že je uživatel nemá možnost vidět. Mohou pomoci při sledování interakcí uživatelů se systémem, což je důležité při ladění funkcionality.

Zkušenostní body

Jsou nejdůležitější ze všech, jelikož slouží jako nástroj pro sledování, určení postavení a směrování uživatele v systému. Uživatel je dostává za všechny interakce, i v případě, že dosáhl maximální úrovně, nemají horní hranici a neměly by klesat.

Dovednostní body

Bonusové body obdržené za volitelné a alternativní aktivity. Slouží jako odměna pro ty, kterým splnění hlavního cíle nestačí.

Reputační body

Nejvíce komplexní, použité v případě, kdy potřebujeme ukazatel důvěryhodnosti mezi dvěma nebo více stranami a nemůžeme ho sami garantovat. Implementace reputačních bodů není snadná, jde o důležitý ukazatel a uživatelé ho mohou snadno zneužít. Příkladem je systém reputace společnosti **eBay**, kde každý uživatel má svojí reputaci na základě toho, jak jsou s ním spokojeni ostatní.

5.6.2 Úrovně

Úrovně ve většině případech slouží jako ukazatel postupu uživatele na časové ose. Přechody mezi úrovněmi nejsou lineární. Měly být snadné na pochopení, rozšiřitelné, flexibilní a vyvážené. Příkladem mohou být vysokoškolské tituly, všechny jasně uvádějí, jaká úroveň vzdělání byla dosažena.

5.6.3 Žebříčky

Jsou snadným a přehledným nástrojem pro srovnávání různých typů ukazatelů v systémů.

Návrh žebříčku

Při návrhu žebříčku je důležité přijít s řešením, které nebude uživatele odrazovat, což znamená zobrazovat uživatele v žebříčku za každé situace bez ohledu na jeho pozici. První možnou cestou je postavení uživatele doprostřed žebříčku a zobrazení jeho přímého okolí, pod ním se tak nachází uživatele šlapající mu na paty a nad ním uživatel, kterého musí překonat. Druhou možností je řez žebříčku pro získání menšího vzorku, například pouze výběr přátel nebo nějaké kategorie.

5.6.4 Odznaky

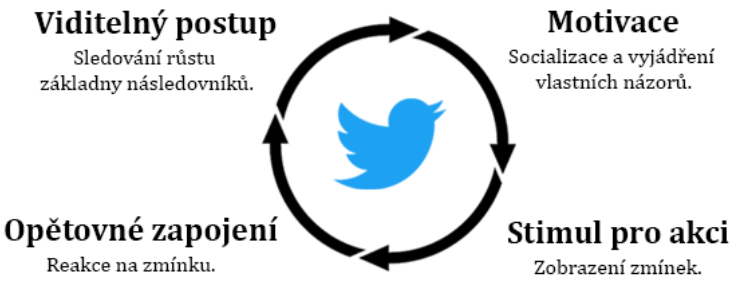
Jsou jedním z nejvíce viditelných prvků, a tudíž výborným prostředkem pro propagaci produktu a značení splnění cílů a postupu v rámci systému. Touha po odznacích má více příčin, sbírání je pro mnohé lidí velkým motivátorem, někdo zase preferuje překvapení ve formě nečekaného odznaku. Odznaky jsou ovšem často vnímány jako nudné, příčinou je špatná implementace.

5.6.5 Výzvy a úkoly

Výzvy a úkoly pomáhají vest uživatele správným směrem, někteří vstoupí do systému bez představy o jeho cílech a možnostech. Uživatel by měl mít stále k dispozici nějakou výzvu a zajímavé činnosti ke splnění. Někteří uživatele plní jednu výzvu za druhou, jiným stačí pouze jedna pro získání jejích pozornosti. Naším úkolem je vytvořit mnoho zajímavých možností, kudy se uživatel může vydat.

5.6.6 Smyčky zapojení

Správný návrhář musí vědět, jak zapojit uživatele do systému, ale hlavně jak ho po opuštění motivovat k návratu. Toto je kritické k udržení uživatelské základny. Uživatel, kterému chybí motivace k návratu se do systému nevrátí. Cílem smyček je maximalizace zapojení a znovuzapojení uživatelů. Jsou složeny ze čtyř stádií, motivace, stimul pro akci, opětovné zapojení a viditelný postup.[30]



Obrázek 9: Smyčka zapojení sociální sítě Twitter
Zdroj:[30], vlastní zpracování

5.7 Typy uživatelů

Při návrhu gamifikovaného systému je důležité znát svoji cílovou skupinu, abychom mohli vytvořit takový zážitek, který bude uživatele vést požadovanou cestou a dostatečně je motivovat.

S jednoduchým způsobem na klasifikaci přišel profesor a herní výzkumník **Richard Bartle** v roce 1996. Na základě jeho eseje vznikl v roce 1999 test s názvem **Bartle Test of Gamer Psychology**, který rozlišuje uživatele do čtyř skupin podle kladených otázek.[39] Toto klasifikovaní není však pro naše účely úplně vhodné. Nepočítá s faktorem, že se v systému vyskytují uživatelé, kteří normálně nemají zájem účastnit se a chtějí jen odměnu.

Klasifikací zaměřenou přímo na gamifikaci je **Marczewski's Player and User Types Hexad** od experta na gamifikaci **Andrzeje Marczewskiho**. Jde o nástavbu na Bartleho koncept s rozšířenou klasifikací do šesti hlavních skupin.

Socializers

Vyhledávají především sociální interakce s ostatními a zajímají se hlavně o části systému, které jim toto umožňují. Nebrání se spolupráci vůči většímu cíli, na který by jinak sami nedosáhli.

Free Spirits

Požadují sebevyjádření a smysluplné zapojení. Dělí se na creators a explorers. Creators touží po tvorbě nového obsahu, ať už pro sebe či komunitu. Jejich uživatelské profily budou nejvíce na očích. Explorers nechtějí být omezení při prozkoumávání. Zážitky se rádi pochlubí a narazí na nejvíce chyb v systému.

Achievers

Touží po mistrovství a být nejlepší nebo alespoň dosáhnout nějakého určitého cíle. Dělají to především pro sebe, potřeba sdílet úspěchy je sekundární. Chtějí se učit novým věcem a neustále se zlepšovat. Konkurenci vidí jako překážky ve své cestě.

Philanthropists

Motivuje je účel a význam. Rádi mají pocit, že jsou součástí něčeho většího. Jsou velmi aktivní ve své komunitě a nevadí jim opakující se činnosti, pokud z nich mají pocit, že tím někomu pomáhají. Za pomoc neočekávají odměnu.

Players

Jsou motivováni odměnou, účastní se jen pro její získání. Dělí se na čtyři podskupiny:

- **Self-Seekers** – Pomáhají jiným, ale bez odměny nehnou prstem. Mohou být užiteční, ale musíme očekávat spíše kvantitu než kvalitu.
- **Consumers** – Udělají potřebné pro získání odměny, jsou ale radši, pokud není potřeba vyvíjet větší úsilí. Typickým příkladem jsou jedinci, kteří nakupují neustále ve stejném obchodě jen kvůli věrnostním programům.
- **Networkers** – Vyhledávají sociální interakce jako socializers, s cílem využití nově získaných kontaktů pro svůj prospěch. Jdou ve stopách úspěšnějších s nadějí na rozšíření jejích vlivu.
- **Exploiters** – Mají blízko k free spirits s oblibou v prozkoumávání, ovšem pouze k dosažení odměn. Pokud naleznou mezeru v systému, tak ji nenahlásí, naopak ji rádi využijí. Alespoň dokud se nenajde někdo, kdo ji dokáže využít lépe než oni.

Disruptors

Jsou motivováni změnou, chtějí narušit systém a vyvolat změny, ať už pozitivní, nebo negativní. Správný přístup k nim může systému prospět, špatný zase velice uškodit. Ačkoliv tvoří malý podíl uživatelů jsou velmi vlivní. Dělí se na čtyři podskupiny:

- **Griefers** – Chtějí negativně ovlivnit druhé jen, protože můžou. Ať už chtějí dát najevo svůj negativní postoj vůči systému, nebo pro zábavu.

V gamifikovaném systému nemají místo a v našem zájmu je udělat vše pro změnu jejich záměru nebo zbavení se jich.

- **Destroyers** – Touží po rozbití systému, ať už podvodem, či využitím mezery v pravidlech, hlavně pokazit zážitek ostatním. Jejich záměr je podobný předchozí skupině.
- **Influencers** – Chtějí pozitivně změnit chod systému ovlivněním ostatních. Při správném přístupu se z nich stanou advokáti systému. V horším případě se z nich mohou stát griefers.
- **Improvers** – Využívají podvody a mezery, ale v nejlepším úmyslu. Jsou podobní free spirits v tom, že touží po prozkoumání systému, nalezení chyb a jejich odstranění.[40]

Obrázek 10: Rozdělení hráčů dle Marczewskiho
Zdroj: [40]

6 Analýza a návrh systému

V teoretické části jsme si definovali a popsali jednotlivé prvky potřebné pro naplnění cíle. Nyní je čas na základě těchto poznatků analyzovat a navrhnout systém skládající se ze čtyř hlavních částí: QR kódy, gamifikace, mobilní aplikace a webové API. Nejdříve si definujeme účel a název systému, poté si stanovíme požadavky a prozkoumáme již existující projekty s podobným zaměřením v Česku. Následně detailněji rozebereme jednotlivé části systému a v poslední řadě se podíváme na možnosti financování. Nyní již k samotnému účelu a názvu systému.

Účelem systému je sběr otisků bezdrátových sítí za pomocí QR kódů a gamifikace jako motivátorů uživatele. Sebrané údaje vhodně uchovávat a umožnit jejich možné zpracování a využití, což může představovat mapování pokrytí Česka bezdrátovými sítěmi. Dále tvoření různých statistik o těchto sítích, například počet nezabezpečených Wi-Fi sítí nebo zjištění, jaký mobilní operátor nabízí lepší připojení ve vybrané lokaci.

Je dobré zmínit, že získané otisky jsou anonymní a nenesou vazbu na jejich zdroj. Není tedy možné zpětně dohledat autora otisku. A ačkoliv to není nutný požadavek, otisky jsou díky anonymitě veřejně dostupné skrze API pro využití třetími stranami.

Jako název systému volíme **SeenIt**, což je v doslovném překladu „*viděl jsem to*“ a odráží uživatelský záměr. Důvodem pro název v angličtině je možné rozšíření působení mimo Česko a celkový trend osvojování anglických termínů v našem mateřském jazyce.

6.1 Požadavky na systém

Uvedeme si požadavky pro jednotlivé části systému tvořené na základě poznatků z teoretické částí, ostatních uvedených zdrojů a vlastních zkušeností.

QR kódy

Musíme zvolit vhodnou fyzickou reprezentaci QR kódů pro určení sběrových míst. Na tuto reprezentací máme několik požadavků:

- **Minimální cena** – Důležitý faktor pro adopci systému. Velké pořizovací náklady znamenají méně sběrových míst.
- **Snadné umístění** – Situování QR kódů do prostředí by neměl být komplikovaný proces.
- **Malý formát a viditelnost** – Fyzická reprezentace by měla zabírat co nejméně místa a zároveň být výrazná pro snadnou identifikaci a zapamatování.
- **Odolnost** – Odolání venkovním podmínkám bez nutnosti časté výměny.

Gamifikace

Je nutné přijít s návrhem, který bude dostatečně motivovat uživatele v interakci se systémem. Především zaměření se na motivaci po delší dobu, nejen po prvním spuštění. Podle průzkumu totiž 25 % uživatelů použije aplikaci pouze jednou a 75 % ji po 90 dnech odstraní.[41] Na základě poznatků z teorie vyvodíme požadavky:

- **Naplnit základní lidské motivátory** – Pro co možná nejefektnější motivaci.
- **Využít efektivně herní prvky** – Především smyčky zapojení pro výše zmíněné důvody.
- **Počítat se všemi možnými typy uživatelů** – Pozitivní i negativní. Zavrhnutím některých z nich přijdeme o znatelnou část uživatelské základny.
- **Poskytnout odměnu** – Především hmatatelnou, kterou někteří vyžadují.

Mobilní aplikace

Některé z těchto požadavků jsou obecné a dají se aplikovat na většinu aplikací:

- **Intuitivní uživatelské prostředí** – Snadná navigace a přehledné pohledy. Zahrnuje i vhodné využití animací a gest.
- **Co nejmenší instalační velikost** – Použití jen důležitých multimédií.
- **Responzivní** – Komunikační a zdlouhavé procesy na pozadí by měly být asynchronní a neblokovat vlákno uživatelského prostředí.
- **Minimální potřebná oprávnění** – Uživatele neradi udělují oprávnění a dlouhý list při instalaci na starších verzích Android může některé odradit.
- **Žádné hard-coded datové struktury** – Přidání nových prvků, jako například odznaků tak nevyžaduje aktualizaci aplikace.[42]

- **Snadná autentizace** – Registrace účtu speciálně pro aplikaci může být odrazující a nevhodná. Až 68 % uživatelů odstraní aplikaci kvůli složité registraci.
- **Smysluplné chybové hlášky** – Poskytnutí jasných informací o chybě a jejím možném řešení.
- **Potlačení nechtěných notifikací** – Umožnění uživateli vybrat jaké notifikace chce zobrazovat. Až 71 % uživatelů uvádí mezi důvody pro odstranění aplikace otravné notifikace.[43]

Web API

Stejně jako u požadavků mobilní aplikace se jedná o obecné požadavky aplikovatelné na všechna web API:

- **Využití základních rout pro přístup ke zdrojům** – Pro získání uživatele použít základní routu `/users/1` místo vlastní `/users/getUser/1`.
- **GET požadavky nemění stav** – Vrací pouze zdroje.
- **Content negotiation** – Klient rozhoduje o reprezentaci zdroje v požadavku.
- **Verzování** – API podporuje verzování controllerů pro zpětnou kompatibilitu se staršími klienty.
- **Správné status kódy** – Každá odpověď obsahuje vhodný status kód.[44]
- **Cachování** – Využité pro časově náročné požadavky.
- **Logování** – Důležité při vývoji i provozu. Pomůže zjistit příčiny chyb.

6.2 Existující projekty

Projektem s podobným zaměřením v Česku je **Wifileaks**[10]. Jeho účelem je ale pouze sběr Wi-Fi sítí, jak je zřejmé z názvu. Výhodou tohoto projektu je staří, které se datuje od roku 2011. Od této doby se nasbíralo téměř tři milióny záznamů o Wi-Fi sítích, což je úctyhodné číslo. Stále je možné stažení aplikace a přispívání do databáze, ale poslední aktivita ze strany tvůrců je z roku 2013. Wifileaks využívá pouze některé gamifikační prvky, převážně jde o žebříčky. Jsou k dispozici i některé

[10] Dostupný na: http://www.wifileaks.cz

detailnější ukazatele o účasti. Ovšem tím „*gamifikace*“ končí a přidaná hodnota pro uživatele také. Je otázkou, jestli byla nedostatečná motivace důvodem pro poměrně krátkou životnost projektu.

Dalším relativně blízkou snahou v Česku bylo umisťování QR kódů na veřejná místa. Především pro účely marketingu a sdělení informací. Jedním z uvedených a pro nás zajímavých míst jsou památky a atraktivity. Zdroje uvádí, že QR kódy tak plnili funkci jakési informační tabule, uživatel po oskenování získal informace o místě. Konkrétní projekt zabývající se touto tématikou nese název **Za poklady příhraničí aneb turistika 21. století**.[11] Je nutné podotknout, že krom získání informací projekt nenabízel žádnou jinou motivaci pro skenování QR kódů. Podobně jako u Wifileaks se poslední články k této tématice váží k roku 2013. Můžeme tak přepokládat, že snaha byla neúspěšná a projekt již není aktivní.

Oba projekty nejsou brány jako přímá konkurence našeho systému kvůli jejích neaktivitě, což neznamená, že je nemůžeme využít pro inspiraci a poučení za lepším návrhem.

6.3 Komponenty systému

Je na čase definované požadavky na jednotlivé komponenty naplnit. Zprvu si konečně určíme uživatelsky přívětivou reprezentaci QR kódů a podíváme se na jejich specifika. Poté se vrhneme na gamifikaci, kde se budeme věnovat zejména herním prvkům. Následuje mobilní aplikace s hlavním zaměřením na uživatelské prostředí a upřesnění sejmutí a obsahu otisku. Předposlední je na řadě webové API s návrhem jednotlivých etap vývoje, a způsobu ukládání otisků.

Zdrojový kód uvedený v následujících kapitolách je ve většině případech zkrácený a upravený kvůli formátování. Jde tedy spíše o demonstraci nežli přesný postup.

[11] Více informací na: http://www.silesiatourism.com/www/cz/geocaching

6.4 QR kódy

Podle stanovených požadavků je vhodné umístění QR kódů na **samolepky**. Máme tak k dispozici velkou variaci, od levných a méně odolných papírových, po fóliové, co vydrží déle. Snadnost aplikace je zřejmá díky lepivé straně. Samolepka nám taky poskytuje svobodu ve stylování, můžeme použít výrazné barvy a složité vzory. Jedním z problému jsou náklady na tisk, což můžeme do určité míry mitigovat hromadným předtisknutím.

Od této chvíle budeme QR kód a jeho reprezentaci nazývat samolepkou. Jde o vhodnější pojmenování z uživatelského hlediska, sběr samolepek zní lépe než sběr QR kódů.

6.4.1 Druhy samolepek

Máme dva druhy, první a hlavní jsou samolepky **turistických atrakcí**. Jsou umístěny například u památek a pro jejich vytvoření musí být navázán kontakt s majitelem. Důvodem je předejití problému nechtěné samolepky na atrakci. Rovnou je zde tak prostor pro domluvu možné spolupráce, například slevu na vstupném po sebrání samolepky u pokladny. Takováto hlubší integrace nabízí možnost financování systému popisovanou v oddílu financování systému. Druhou výhodou je přidaná a „*hmatatelná*“ hodnota pro uživatele. Češi velmi slyší na slevy a zlevněné vstupné nebo suvenýr může představovat lákavou odměnu.

V pozadí pak stojí **komunitní** samolepky, u kterých o přidání žádají uživatelé prostřednictvím aplikace. Je nutné uvést polohu a detaily o lokaci pro validaci administrátorem. Po přidání samolepky do systému je s uživatelem dohodnuto její umístění. Při dostatečném financování je možné zaslání samolepky poštou, jinak elektronicky. U tohoto druhu je tedy vhodné umožnit zhotovení samolepky na uživatelovo náklady.

6.4.2 Umístění samolepek

Preferované umístění samolepek je ve venkovních prostorech, kde je GPS lokalizování pro validaci nejpřesnější, a tudíž nejmenší odchylky. Interiéry jsou také

možností, ačkoliv zde lokalizování na Wi-Fi a mobilní síti poměrně ztrácí na přesnosti.

Samolepka atrakce je umístěna v jejím areálu, ideálně u poklady nebo vstupu. Komunitní naopak na veřejných místech s volným vstupem. Její získání představuje větší výzvu, jelikož umístění závisí vyloženě na uživateli a může být i skrytá. Je tak spíše zaměřena na uživatele dobrodruhy.

6.4.3 Obsah samolepek

V QR kódu je na úplném začátku uveden text pro neuživatele systému. Vhodný v případě, když samolepku oskenuje někdo mimo naší aplikaci. Dozví se tak informace o projektu a je odkázán na adresu, kde si může stáhnout naší aplikaci.

Následuje text určený pro identifikaci příslušné samolepky. Je složen primárně z interního identifikátoru samolepky v systému. Tento text je společně s otiskem odesílán na server pro validaci. Validace je popsána v oddílu validace otisku.

Vnořený obsah uvedený při tvorbě samolepky je uložen na serveru. Máme tak k dispozici téměř neomezenou kapacitu a zároveň uchováme nízkou verzi QR kódu, což znamená rychlejší čtení při uchování minimálního formátu a možnost vysoké úrovně korekce. V případě samolepky atrakce je vnořeným obsahem například URL adresa webu nebo zajímavosti o ní. U komunitní může jít o jakýkoliv text na přání uživatele, který musí být samozřejmě schválen administrátorem.

6.5 Implementace gamifikace

Snaha o naplnění motivátorů a pokrytí všech uživatelských typů se odráží v celém návrhu, nebudeme je tedy jednotlivě vyjmenovávat a uvádět jejich řešení. Detailněji se ale podíváme na herní prvky, jelikož jsou obsaženy ve všech interakcích uživatele se systémem.

6.5.1 Body

Z bodových systému uvedených v oddílu body využíváme všechny. Zkušeností a reputační body se uživateli jeví jako jedny. Pro nás však mají rozdílné účely a

budeme je uchovávat odděleně. Body se připočítávají i po dosažení maximální úrovně.

Zkušenostní body

Uživatel je dostává za sebrané samolepky. Jsou hlavním pohonem při získávání úrovní. Snadně řečeno, čím více samolepek uživatel sbírá, tím větší je jeho úspěšnost v rámci systému. Za získání samolepky uživatel obdrží minimální počet bodů, který je dále pouze kladně ovlivňován dvěma faktory, vzácností a rychlostí. Samolepka, kterou má v portfoliu malé procento uživatelů je považována za vzácnější a patřičně odměněna. Pokud si uživatel dá na čas a sbírá jednu samolepku za druhou, je také odměněn. Proces výpočtu zkušenostních bodů je uveden v Příloha 1: Vypočet bodů za sebrání samolepky

Dovednostní body

Jde o sekundární body za splnění výzev. Trochu s postupem pomůžou, ale slouží spíše jako motivace pro odhodlané uživatele toužící po první pozici v žebříčku. Jsou ovlivněny dvěma faktory, časem a obtížností, dřívější splnění náročné výzvy znamená více bodů. Ovlivnění je opět jen kladné. Uživatele, který si dal záležet se splněním výzvy nechceme demotivovat záporným ukazatelem stržených bodů.

Reputační body

Nemají na postup žádnou vazbu. Jsou ukazatelem důvěryhodnosti uživatele, který je získává pouze věnováním od jiného uživatele ve formě kladného nebo záporného hlasu. Tyto dva druhy poté zobrazujeme vedle celkové reputace. Tím předcházíme problému, kde se uživatelovo reputace jeví jako kladná. Ale ve skutečnosti má například 42 pozitivních a 30 negativních bodů, celkově tedy kladných 12. Stimulem pro udělení reputačních bodů může být jakákoliv interakce mezi uživateli. Přímá v diskuzi, i nepřímá jako zhodnocení samolepky, kterou uživatel vytvořil.

Ukazatel důvěryhodnosti slouží hlavně pro uživatele jako možná odpověď na následující otázky. Vyplatí se vyrazit vstříc samolepce uživatele se zápornou reputací? Mám v diskuzi věřit uživateli se špatnými ohlasy? Finální volba je na nich, můžeme jim však poskytnout pomocnou ruku.

6.5.2 Úrovně

Ukazují uživateli postup a jsou jasně stanovené. Pro přechod na další úroveň uživatel potřebuje specifický počet bodů. Rozložení úrovní není lineárního charakteru, začínáme rychlým rozjezdem a prvních pár samolepek přenese uživatele přes nejvíce úrovní. Postupně však snižujeme tempo a zvedáme obtížnost dosažení další úrovně. Rychlost postupu je závislá na odhodlání uživatele, uvádíme dva scénáře, nejhorší a nejlepší možný. Znamenají průběh při minimálním a maximálním počtu získaných bodů za samolepku.

V praxi to vypadá následovně. Uživatel začíná na nulté úrovni, první sebraná samolepka v obou scénářích znamená přechod na první úroveň. Na druhou úroveň již však v nejhorším případě potřebuje dvě další samolepky a v nejlepším stále jednu. Počet potřebných samolepek se zvedá. Na poslední, stou úroveň uživatel musí získat přibližně 5200 samolepek v nejhorším případě a v nejlepším jen 3450. Pohled na rozložení úrovní je dostupný v Příloha 2: Rozložení úrovní.

Tituly

Doplňují úrovně a jde pouze o pomocný indikátor postupu a přikrášlení portfolia. Jejich alokace na základě dosažené úrovně je následující:

- **Začátečník** – Úroveň 0–14.
- **Výletník** – Úroveň 15–29.
- **Průzkumník** – Úroveň 30–59.
- **Všudybyl** – Úroveň 60–100.

6.5.3 Žebříčky

Sledujeme dva typy, žebříček globální a přátel. V obou stavíme uživatele doprostřed, pokud nezvolí mód zobrazení prvních pozic. Defaultně v nich určuje pořadí počet dosažených bodů, uživatel má ale na výběr i řazení dle úrovně a počtu sebraných samolepek.

Pro zmenšení vzorku je k dispozici třídění podle kategorií. Hlavními kategoriemi jsou samolepky atrakcí a komunitní. Každá má pak vlastní podkategorie, v případě atrakcí například hrady, zámky apod.

6.5.4 Odznaky

Stejně jako reputační body, neposouvají uživatele v žebříčku a slouží jako odměna za vykonávané činnosti a interakce v rámci systému. Uživatel předem neví, za co vše obdrží odznak, je zde tak určitý moment překvapení při jeho získání.

Příkladem udělení odznaku je vytvoření samolepky. Uživatel dostane odznak „*Content Creator*“, který bude zdobit jeho portfolio.

6.5.5 Výzvy

Jsou úkoly s časovým omezením a automatickou účastí. Časové omezení je variabilní a může být závislé na obtížnosti výzvy, některé jsou aktivní v rámci hodin, jiné dnů. Obtížnost má čtyři úrovně, lehká, normální, těžká a nemožná. Tyto úrovně složí jako ukazatel toho, zda je výzva vhodná pro konkrétního uživatele. Občasný uživatel tak ví, že těžká nebo nemožná výzva není nejspíše určena primárně pro něj.

Výzvy se nemusí přijímat, postup v nich je automaticky sledován a aktualizován. Při jejich dokončení uživatel obdrží dovednostní body na základě obtížnosti a rychlosti splnění.

Cíle výzev jsou proměnlivé, příkladem může být sebrání tří samolepek z hradů během jednoho týdne. Publikaci výzev má na starosti administrátor systému.

6.5.6 Smyčky zapojení

Představují v našem systému primárně notifikace zobrazované jako reakce na vybrané události. Tyto události představují:

- Přidání nového příspěvku do diskuze.
- Získání odznaku nebo splnění výzvy na pozadí.
- Sebrání samolepky přítelem v okolí.
- První získání komunitní samolepky (pouze jejímu tvůrci).

Jednotlivé notifikace je možné vypnout, respektujeme uživatelovo preference, jelikož mohout mít negativní dopady, pokud si je nepřeje dostávat.

Pasivní smyčku může představovat i touha pro sledování postupu přátel nebo sledování obsazení prvních pozic v žebříčku.

6.6 Návrh mobilní aplikace

Mobilní aplikace je vstupní branou uživatele do systému, první dojem musí být pozitivní. Splnění všech požadavků je tedy velice žádoucí. Návrh nám výrazně zlehčuje knihovna zpětné kompatibility **Support Library** od Googlu. Přináší podporu nových funkcí do nižších API levelů, a především komponenty materiálního designu, například `Snackbar` nebo `TabLayout`. Prvky této knihovny jsou využity napříč celou aplikací.

6.6.1 Parametry projektu

Většinu těchto parametrů je nutné uvést v manifestu. Visual Studio nám ulehčuje práci a jejich zvolení je možné ve formuláři vlastností projektu a nemusíme nic manuálně upravovat.

- **Package name** – Držíme se konvence a používáme `cz.uhk.seenit`.
- **Minimální API level** – Je 19 (4.4 **KitKat**), máme tak k dispozici všechny požadované funkcionality a nemusíme řešit zpětnou kompatibilitu. Přijdeme zhruba o 5 % uživatelské základny vlastnící starší zařízení,[45] což je poměrně zanedbatelné číslo a většina vlastníků těchto zařízení není naší cílovou skupinou.
- **Cílový API level** – Je 27 (8.1 **Oreo**) a zaměřujeme se tak na poslední verzi.
- **Oprávnění** – Pro naše účely nám stačí: ACCESS_FINE_LOCATION, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, BLUETOOTH, CAMERA, GET_ACCOUNTS, INTERNET s SYSTEM_ALERT_WINDOW

6.6.2 Autentizace

Používáme tokenovou formu Google Sign-In postavenou na Oauth 2.0 z **Auth API** v Google Play Services.

Jako první je nutné vytvořit projekt v Google API konzoli[12]. V něm následovně zapnout Google+ API a vygenerovat dvě OAuth klientské ID. Jedno pro mobilní

[12] Dostupná na: https://console.developers.google.com

aplikaci a druhé pro API, účelem těchto ID je validace tokenů. Při generování ID mobilní aplikace je potřeba uvést package name.

Při přihlášení do mobilní aplikace přikládáme do požadavku vygenerované ID API. Získáme tak nazpět uživatelský token, který mobilní aplikace odesílá v každém požadavku, kde z něj po validaci zjistíme uživatelovo identitu. Validace tokenu je popsána v oddílu autentizace klienta.

Požadavky přihlášení

Během přihlášení je nutné specifikovat požadavky na zdroje a sekce uživatelova Google profilu, ke kterým chceme přistupovat. Nejdůležitější položkou je požadavek na token popisovaný výše. Dále potřebujeme přístup do těchto sekcí:

- **profil** – Pro získání základních informací jako email, jméno a příjmení a URL profilového obrázku v rámci Google.
- **plus.circles.read** – Pro čtení uživatelovo kruhů na Google+ a tím mít možnost importovat přátele v do našeho systému.
- **plus.stream.write** - Pro publikování příspěvků na Google+ profil uživatele, když bude uživatel chtít publikovat získanou samolepku nebo splněnou výzvu.

Implementace přihlášení

Pokud není uživatel v Androidu přihlášen pod Google účtem a pokusí se přihlásit v naší aplikaci, je automaticky vyzván pro přihlášení nebo registraci Google účtu skrze systémovou aktivitu viditelnou na obrázku 11 (Fáze přihlášení) vlevo.

Samotná implementace přihlášení je řešena v hlavní aktivitě. V metodě `OnCreate` inicializujeme potřebné komponenty. Do konfigurace `GoogleApiClienta` přidáme požadavky přihlášení uvedené výše. V `OnStart` se pokusíme o tiché přihlášení, které je možné v případě, že uživatel byl v nedávné době již přihlášen. Tato metoda je vhodná zejména při ztrátě focusu aplikace, v tomto stavu totiž uživatele automaticky odhlašujeme a při získání focusu opět přihlašujeme. Když tiché přihlášení selže, provedeme plnohodnotné přihlášení s callbackem, což ve výsledku vyvolá systémový dialog s volbou Google účtu, pod kterým uživatel chce vystupovat v aplikaci. Tento dialog je na obrázku 11 (Fáze přihlášení) uprostřed. Posledním

krokem je udělení přístupu ke zvoleným zdrojům účtu z požadavků uvedených v konfiguraci. Dialog pro udělení je na obrázku 11 (Fáze přihlášení) vpravo. Pokud je udělen přístup a požadavek o přihlášení na Google API byl úspěšný, odešleme požadavek na vytvoření nebo aktualizaci účtu v našem systému. Provedení tohoto požadavku je řešeno v oddílu autentizace klienta.

Kód 1: Implementace autentizace

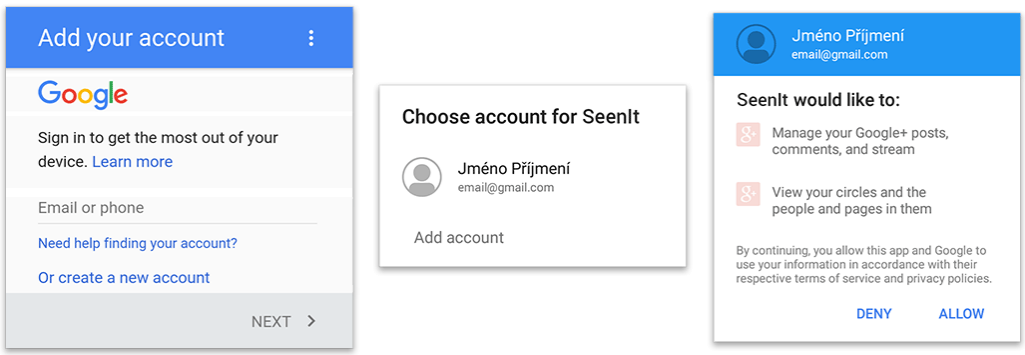
```
// Vytvoření požadavku přihlášení
var signInOptions = new GoogleSignInOptions.Builder(DefaultSignIn)
    .RequestIdToken(ApiId) // Přiložení OAuth ID API
    .RequestScopes(new Scope(Scopes.Profile)).Build(); // Specifikace sekcí

var googleApiClient = new GoogleApiClient.Builder(this)
    .EnableAutoManage(this, this) // Zapnutí automatického životního cyklu
    .AddApi(Auth.GOOGLE_SIGN_IN_API, signInOptions).Build();

protected override void OnStart()
{
    var pendingesult = Auth.GoogleSignInApi.SilentSignIn(GoogleApiClient);
    if (pendingResult.IsDone)
        HandleResult(pendingResult.Get()); // Tiché přihlášení úspešné
    else
        pendingResult.SetResultCallback(new ResultCallback(HandleResult));
}

protected override void OnActivityResult(int requestCode, Result resultCode,
    Intent data)
{
    var result = GoogleSignInApi.GetSignInResultFromIntent(data);
    HandleResult(result);
}

private void HandleResult(GoogleSignInResult result)
{
    if (result.IsSuccess)
        // Uživatel přihlášen, zpracuj profilové informace
    else
        // Uživatel nepřihlášen, zkus přihlášení znovu
}
```



Obrázek 11: Fáze přihlášení

6.6.3 Udělení oprávnění

Od šesté verze Androidu **Marshmallow** se oprávnění udělují za běhu aplikace, místo při instalaci. Musíme tak přizpůsobit návrh, abychom nenarazili na problémy s neoprávněným přístupem. Oprávněním spadající do kategorie normální jsou automaticky udělována, jde o všechny oprávnění nemající přístup k soukromým údajům uživatele.

Nejdříve se podíváme na verzi Androidu, pro API level menší než 23 nemusíme nic řešit. Pro novější verze je prvním krokem zjištění, zda máme oprávnění na funkcionalitu. Pokud ano, provedeme ji. Jinak zjistíme, zda je vhodné zobrazit dialog s vysvětlením proč je oprávnění potřebné. Jestli je vysvětlení vhodné, zobrazíme jej a po jeho odkliknuti požádáme, jinak požádáme rovnou. V obou případech je uživateli předložen systémový dialog požádání o udělení oprávnění, který lze odmítnout a přijmout. Přijmutím provedeme funkcionalitu. Odmítnutím je systémové požádání zobrazeno znovu, tentokrát je již přítomna možnost zrušení dalšího zobrazení. Zvolením této možnosti cyklus končí a funkcionalita se neprovede. Uživatel pak musí jedině udělit oprávnění ručně v nastavení. O tomto faktu ho informujeme, jelikož všechny požadované funkcionality jsou klíčové pro chod aplikace. Krom neustálého upozorňování a zablokování fragmentů nic jiného dělat nemůžeme. Všechny dialogy jsou k vidění na obrázku 12 (Dialogy oprávnění).

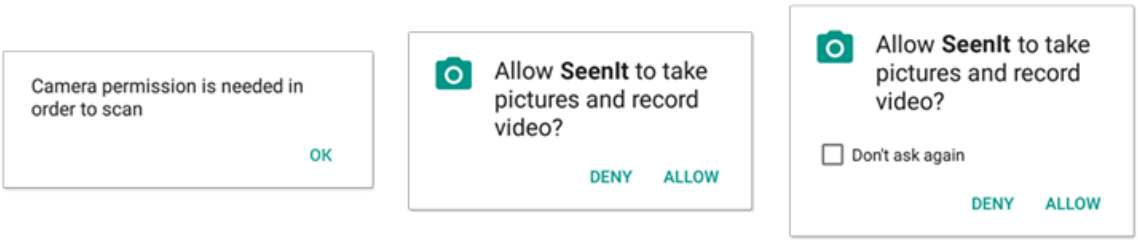
Kód 2: Prvotní požádání o oprávnění

```
if (Build.VERSION.SdkInt < BuildVersionCodes.M)
    // Oprávnění uděleno během instalace, proveď funkcionalitu

// Máme oprávnění ke kameře?
if (CheckSelfPermission(Application.Context, Permission.Camera) != Granted)
{
    // Oprávnení nemáme. Měli bychom ukázat vysvětlení pro oprávěení?
    if (ShouldShowRequestPermissionRationale(Activity, Permission.Camera))
    {
        // Ano, ukaž dialog s vysvětlením a následně požádej
        new AlertDialog.Builder(Activity)
            .SetMessage("Camera permission is needed in order to scan")
            .SetPositiveButton("OK", delegate
                { RequestPermissions(new[] { Permission.Camera }, 42); })
            .Create().Show();
    }
    else
        RequestPermissions(new[] { Permission.Camera }, 42); // Ne, požádej
}
else // Oprávnění máme, proveď funkcionalitu
```

Kód 3: Callback požádání o oprávnění

```
public override void OnRequestPermissionsResult(int requestCode,
    string[] permissions, Permission[] results)
{
    // Udělil uživatel oprávnění?
    if (CheckSelfPermission(Context, Permission.Camera) == Granted)
        // Ano, proveď funkcionalitu (zvolil allow)
    else
        // Ne, zvolil deny nebo Don't ask again, zkusíme požadat znovu
        RequestPermissions(new[] { Permission.Camera }, 42);
```



Obrázek 12: Dialogy oprávnění

6.6.4 Navigace

Support Library zde také nezklame, nabízí efektivní způsob navigace v podobně navigation draweru. Doslovný šuplík je centrální navigační prvek, ze kterého se dostaneme na vše důležité. Výhodou je možnost skrytí a zároveň dostupnost ve všech fragmentech aktivity. Pro jeho zobrazení stačí swipe z levého rohu obrazovky nebo kliknutí na ikonu šuplíku na aplikační liště. Je navíc běžně využíván, dokonce i oficiálními Android aplikacemi, pro uživatele tak není ničím novým a nepředstavuje překážku. Náš šuplík obsahuje odkazy na následující fragmenty, jejichž vizuální návrh je dostupný v přílohách.

Přehled

Pohled přizpůsobený kontextu. Nepřihlášený uživatel bez profilu zde vidí uvítání a možnost přihlášení přes Google účet. Přihlášení reprezentuje klasické Google+ tlačítko. Pod ním se nachází odkazy na podmínky služby, které by si uživatel měl před přihlášením důkladně přečíst. Po kliknutí na přihlášení je spuštěna aktivita pro výsledek s Google SignIn intentem. Následuje přihlášení popisované v oddílu autentizace. Nepřihlášený uživatel má přístup pouze na tento fragment a o aplikaci.

Již přihlášený uživatel zde uvidí přehled své aktivity za poslední týden. Přehled je rozšířen o srovnání s přáteli, jestliže nějaké v rámci Google+ nebo systému má.

Profil

Nejdůležitější pohled pro uživatele, má zde přístup ke svému kompletnímu portfoliu zahrnující:

- **Úroveň a získané body** – Aktuální úroveň uživatele s vyobrazením potřebných bodů pro přechod na další úroveň. Bodové systémy jsou popsány v oddílu body. Titul odpovídající úrovni je také obsažen.
- **Sebrané samolepky** – Seznam získaných samolepek uživatelem. S možností zobrazení jejich detailu, tj. čas obdržení, počet získaných bodů, vzácnost v době sebrání a nyní. Dále zobrazení lokace samolepky na mapě uvnitř detailu, její název a ostatní doplňující informace.
- **Vytvořené samolepky** – Seznam samolepek vytvořených uživatelem. Je zde i tlačítko pro vytvoření nové. Proces vytvoření je uveden v oddílu druhy.
- **Přátele** – Seznam přátel s proklikem na detail jejich profilu. Cizí profil není tak detailní a obsahuje omezené údaje. Je zde i možnost importování přátel z Google+ kruhů.
- **Získané odznaky** – List odznaků uživatele s možnou navigací na detail, kde se nachází údaje zvoleného odznaku. Detail obsahuje název, čas a důvod obdržení odznaku a stejně jako samolepka také vzácnost.
- **Splněné výzvy** – Historie splněných výzev s tlačítkem na přechod do aktivních výzev. Obsahuje údaje o času splnění.

Důležitým prvkem profilu je také nastavení reprezentované ikonou v aplikační liště. Umožní úpravu preferencí profilu a celé aplikace. Uživatel zde může například zvolit úroveň sdílení svého portfolia s ostatními, vypnout a zapnou určité typy notifikací a diskuzi. Další možností profilu je udělení reputačních bodů.

Sken

Hlavní pohled pro nás, probíhá zde skenování samolepek s QR kódy. Není zde moc prvků uživatelského prostředí, jelikož potřebujeme co nejvíce místa pro zobrazení plochy s náhledem kamery. Navíc zobrazíme pouze stručné instrukce pro skenování. Uživatel při přechodu na tento fragment uvidí ihned náhled připravený

ke skenování, pokud již udělil povolení přístupu ke kameře. Řešení získání povolení je popisováno v oddílu udělení oprávnění.

Proces sejmutí otisku při skenu je popsán v oddílu sejmutí otisku. V případě validního skenu, je uživatelovi udělena příslušná samolepka a spuštěna nová aktivita pro výsledek, kde uživatel uvidí detail samolepky. Tento detail je podobný detailu samolepky z fragmentu profilu a jedním z důvodu proč nepoužijeme přímo tento detail je nutnost zobrazení informací o validaci. Musíme uživatele informovat o nevalidní samolepce, druhým scénářem je upozornění o zaslání již získané samolepky.

Pro skenování QR kódů je využit `BardcodeDetector` dostupný z **Vision API** v Google Play Services. Dokáže číst za špatných podmínek a při různých úhlech a je možné specifikovat i zaměření, tím optimalizovat detekci pouze na QR kódy. Při detekování kódu je vyvolán callback, ve kterém již můžeme pracovat s polem detekovaných položek.

Plocha náhledu kamery je také řešena pomocí Vision API. Je vytvářena nad zvoleným pohledem a při vytvoření je vyvolán callback `SurfaceCreated`, kde je nutné ověřit již zmíněné povolení kamery.

Kód 4: Inicializace skenování

```
var detector = new BarcodeDetector.Builder(Activity) // Vytvoření čtečky
    .SetBarcodeFormats(BarcodeFormat.QrCode).Build();
detector.SetProcessor(this); // Nastavení handleru na fragment pro callback

var camera = new CameraSource.Builder(Activity, detector) // Vytvoření kamery
    .SetRequestedPreviewSize(640, 480)
    .SetAutoFocusEnabled(true).Build();

var preview = view.FindViewById<SurfaceView>(ViewId);
preview.Holder.AddCallback(this); // Přidání callbacku při startu kamery

camera.Start(preview.Holder); // Finální start kamery v callbacku
```

Mapa

Pohled podpory orientace a směrování uživatele, nabízející zobrazení existujících samolepek na mapě ve formě markerů. Uživatel tak objevuje samolepky v okolí pro vybrání dalšího možného cíle. Sebrané samolepky jsou vizuálně rozlišeny od těch, které uživatel ještě nemá. Při kliknutí na marker je zobrazen detail s informacemi a možností nahlášení poškození nebo absence samolepky.

Mapa je řešena skrze **Google Maps API** pro Android s přidanými markery podle GPS koordinací a parametrů samolepek. Podobně jako u autentizace, opět musíme v Google API konzoli povolit odpovídající API a vygenerovat klíč mobilní aplikace.

Výzvy

Fragment vyhrazený pro listování aktivních a nedávno ukončených výzev. Uživatel má tak přehled o svém postupu v nich, popřípadě dokončení. Jednotlivé výzvy zobrazují globální statistiky, například počet uživatelů, kteří je již splnili. Dále platnost, jelikož výzvy mají omezení časový pro jejich splnění. Systém výzev je rozebrán v oddílu výzvy.

Žebříčky

Pohled určený příznivcům konkurence. Poskytne promítnutí uživatelovo pozice v různých žebříčcích. Typy a módy žebříčku jsou uvedené v oddílu žebříčky. Jednotlivé pozice v žebříčku jsou interaktivní a po kliknutí se dostaneme na profil uživatele. Pořadí je určeno typem žebříčku. V globálním rozhodují celkové získané body, v žebříčku určité kategorie samolepek zase počet sebraných za nejkratší čas.

Zobrazení položek žebříčku řeší `RecyclerView` ze Support Library, což je pokročilejší a flexibilnější verze `ListView`. Uspořádání položek je určeno `LinearLayoutManagerem`. Samotou zprávu kolekce má pak na starosti adaptér, řeší její modifikace a různé callbacky, třeba kliknutí na položku. Adaptér si dále sám podle potřeby vytváří jednotlivé pohledy pro položky, takzvané `ViewHoldery`. V metodě `OnCreateViewHolder` jsou vytvářeny a v `OnBindViewHolder` naplněny podle kontextu.

Diskuze

Fragment příznivců sociálních interakcí, poskytuje jednoduchý chat pro uživatele v blízkosti. Uživatel vidí počet aktivních uživatelů v okolí, ne však seznam jejich jmen. Důvodem je uchování anonymity, pokud si uživatel nepřeje být viděn. Po odeslání zprávy je všem v okolí zaslaná notifikace jako impulz pro znovuzapojení. Historie není na straně klienta uchovávána a zavřením aplikace je obsah diskuze vymazán. Není aplikován ani žádný filtr vulgarismů, odhodlaný člověk si vždy najde cestu i přes sebelepší cenzuru.

Diskuze na základě lokace zařízení je možná díky neustálému sledování lokace popisovanému v oddílu sledování lokace. Uživatel má také na výběr vypnutí diskuze a s ní souvisejících notifikací v nastavení, v takovém případě je fragment schován.

O zobrazení zpráv se stará již zmíněný **`RecyclerView`**, ve kterém jsou pohledy zpráv obsahující uživatelův profilový obrázek, jméno, text zprávy a časovou značku. Ve spodní části se nachází textové pole s tlačítkem pro napsání a odeslání zprávy. Zprávy uživatele jsou od ostatních rozlišeny zarovnáním doprava a modrou barvou pozadí. Kliknutím na profilový obrázek zprávy se dostaneme na profil uživatele.

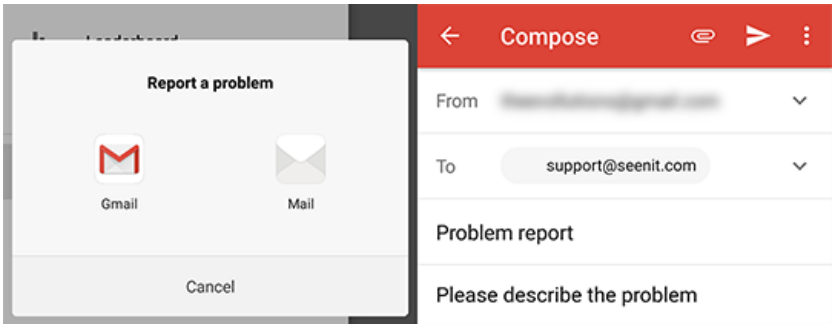
O aplikaci

Zde uživatel najde základní informace o aplikaci, kontakt, a především možnost nahlášení problému. Tlačítko nahlášení problému funguje pouze jako odkaz na spuštění aktivity odeslání emailu. Nemusíme tak vytvářet žádný formulář a uživatel si může vybrat svou preferovanou emailovou aplikaci. Po startu aktivity je zobrazen selektor emailové aplikace, následuje přechod na předvyplněnou kompozici emailu.

Většina uživatelů tuto funkcionalitu nevyužije a možná ani nikdy neuvidí, je však dobré ji mít. I malé zapojení do děje v pozadí dokáže některým přinést dobrý pocit. Po vyřešení obdrženého problému můžeme uživatele obdarovat odznakem za užitečný feedback a tento pocit dále prohloubit.

Kód 5: Nahlášení problému

```
var intent = new Intent(Intent.ActionSendto, Uri.Parse("mailto:address"));
intent.PutExtra(Intent.ExtraSubject, "Problem report");
intent.PutExtra(Intent.ExtraText, "Please describe the problem");
StartActivity(Intent.CreateChooser(intent, "Report a problem"));
```



Obrázek 13: Selektor aplikace a kompozice nahlášení problému

6.6.5 Sledování lokace

Po autentizaci začínáme se sledováním lokace zařízení, která probíhá po celou dobu, co je aplikace na popředí. Pro nejpřesnější výsledky požadujeme od uživatele zapnutí GPS služeb. Samotné sledování probíhá skrze službu **`LocationServices`** v Google Play Services. Služba nabízí možnost přidání callbacku při aktualizaci lokace, dále uvedení požadované přesnosti a intervalu aktualizací. V callbacku získáme poslední známou lokaci, tu si společně s časovou značkou uložíme.

Důvodem pro neustálé sledování lokace je možná pomalá lokalizace především na starších zařízeních. Zjištění lokace až při skenu samolepky by tak mohlo být zdlouhavé. Takto si při skenu vezmeme poslední uloženou lokaci a použijeme ji.

Kód 6: Sledování lokace

```
// Vytvoření klienta pro získání lokace
var locationClient = LocationServices.GetFusedLocationProviderClient(this);

var locationRequest = new LocationRequest() // Nastavení priority a intervalu
    .SetPriority(LocationRequest.PriorityHighAccuracy)
    .SetInterval(TimeSpan.FromMinutes(2).TotalMilliseconds)
    .SetFastestInterval(TimeSpan.FromMinutes(1).TotalMilliseconds);

var locationCallback = new DummyLocationCallback();

// Přidání callbacku, když je aplikace na popředí
locationClient.RequestLocationUpdatesAsync(locationRequest, locationCallback);
// Odebrání callbacku, když je aplikace na pozadí
locationClient.RemoveLocationUpdates(locationCallback);

public class DummyLocationCallback : LocationCallback // Callback třída
{
    public override void OnLocationResult(LocationResult result)
    {
        if (result.Locations.Any())
            // Ulož získanou lokaci
        else
            // Žádná lokace není k dispozici
```

6.6.6 Sejmutí otisku

Jako bylo uvedeno v oddílu navigace, sejmutí se provádí v pohledu sken. Je započato již při přechodu na pohled, ještě před detekcí QR kódu. Důvodem je možná prodleva získání údajů. V případě, že uživatel sice otevře pohled, ale neoskenuje žádný QR kód, jsou obdržené údaje uchovány na pozdější možné odeslání. Díky neustálému sledování lokace si můžeme zpětně zjistit polohu sejmutého otisku.

Od API levelu 18 (4.3 **Jelly Bean**) je možné získat dostupné Wi-Fi sítě i bez zapnuté Wi-Fi, což nám ulehčuje práci, jelikož nemusíme žádat o její zapnutí. Musíme však zkontrolovat stav této možnosti, někteří uživatele ji mohou mít zakázanou. Získání Bluetooth zařízení již vyžaduje jeho zapnutí, musíme o něj uživatele požádat.

Pro získání údajů o konektivitě využíváme systémové služby, `WifiManager`, `BluetoothManager`, `TelephonyManager` a `ConnectivityManager`. V případě Wi-Fi a Bluetooth je nutné při přechodu na pohled zaregistrování broadcast receiverů a spuštění skenu/objevení zařízení. U mobilního signálu je po vytvoření instance `TelephonyManageru` vše potřebné bez prodlevy k dispozici. Získání informací o zařízení umožňuje třída `Build`.

Ověření zařízení

Před sejmutím provádíme ověření, zda se uživatel nesnaží o podvod prostřednictvím falšování lokace zařízení. Uživatel se seznamem QR kódů systému by bez tohoto opatření mohl odesílat nepravdivé údaje a posbírat všechny samolepky téměř instantně. Zjištění falešné lokace poskytuje objekt v callbacku uvedeného v oddílu sledování lokace. Toto však není dostatečné ověření, některé nástroje umožňují obejití. Proto se ujistíme i o tom, že aplikace běží na fyzickém zařízení a ne emulátoru, jelikož většina pokusů od podvody pochází právě z emulátorů Existuje mnoho způsobů a většina z nich používá ověřování parametrů třídy `Build`, kterou také využíváme.

Složení otisku

Záleží na výrobci a API levelu, některé tyto kombinace odhalují více informací než jiné, především o mobilním signálu. Zajímají nás údaje o samotném zařízení, jeho

lokaci, Wi-Fi a mobilních sítích a Bluetooth zařízeních. Nyní se podíváme na hlavní parametry těchto sekcí.

Parametry zařízení:

- **Manufacturer** – Výrobce zařízení.
- **Model** – Modelové označení zařízení.
- **RadioVersion** – Verze rádiového firmwaru.
- **Sdk** – API level Androidu.
- **ActiveNetworkType** – Typ aktivního připojení (Wi-Fi, Mobile).
- **IsActiveNetworkMetered** – Ukazatel měřitelnosti aktivního připojení.

Parametry lokace:

- **Accuracy** – Horizontální přesnost, rádius (metry).
- **Altitude** – Nadmořská výška (metry).
- **Latitude** – Zeměpisná šířka.
- **Longitude** – Zeměpisná délka.
- **Speed** – Rychlost pohybu po zemi (metry za sekundu).

Parametry Wi-Fi sítě:

- **Bssid** – MAC adresa routeru.
- **Ssid** – Název sítě.
- **Capabilities** – Schopnosti sítě (autentizace, šifrování).
- **Frequency** – Frekvence (MHz).
- **ChannelWidth** – Šířka kanálu (MHz).
- **Level** – Síla signálu (dBm).

U mobilního signálu sledujeme parametry zařízení a operátora. Tyto údaje jsou dostupné na všech zařízeních se SIM kartou. Informace o telefonních věžích v okolí již tak přístupné nejsou a velká část zařízení je neposkytuje. V tomto případě hledáme sílu signálu aktuálního připojení jinou cestou. Pro získání informací o mobilním signálu stačí pouze přistoupit k systémové službě.

Parametry mobilního zařízení a operátora:

- **NetworkType** – Typ připojené sítě (GSM, LTE).
- **PhoneType** – Typ sítě telefonu (GSM, CDMA, …).
- **SimOperatorName** – Jméno operátora SIM karty.
- **SimCountryIso** – Zkratka země SIM karty.
- **DataState** – Stav datového připojení.
- **DataNetworkType** – Typ datového připojení (EDGE, LTE, …).

Parametry telefonních věží (pokud jsou k dispozici):

- **Ci** (Cell ID) – ID věže.
- **Pci** (Physical Cell Id) – Fyzické ID věže (0-503).
- **Earfcn** (Absolute RF Channel Number) – Číslo rádiového kanálu.
- **Tac** (Tracking Area Code) – Kód oblasti GSM a WCDMA sítí.
- **Mcc** (Mobile Country Code) – Kód země věže (0-999).
- **Mnc** (Mobile Network Code) – Kód operátora věže (0-999).
- **Dbm** – Síla signálu (dBm).
- **Level** – Úroveň signálu (0-4).

Uvedené parametry jsou LTE věže, GSM, CDMA a WCDMA věže mají rozdílné. Z parametrů je možné zjištění její polohy věže Google **Geolocation API**, což by se dalo využít při zpracování otisků.

Parametry Bluetooth zařízení:

- **Address** – MAC adresa zařízení.
- **MajorDeviceClass** – Typ zařízení (počítač, telefon, …).
- **DeviceClass** – Podřízený typ zařízení (laptop, desktop, …).
- **Name** – Název zařízení.
- **Type** – Typ, BR/EDR (přenos audia), LE (přenos dat) nebo obojí.

Jak je uvedeno v kapitole otisk, filtrujeme Bluetooth zařízení na základě parametru `DeviceClass` a ignorujeme všechna mobilní. Do otisku vkládáme pouze počítače (desktop, server), skenery, tiskárny a síťové prvky (AP).

6.7 Návrh web API

Používáme nejnovější framework postavený na .NET s názvem **ASP.NET Core 2.0**. Oproti staršímu ASP.NET přináší spoustu výhod. Především ve formě lepšího výkonu a multiplatformní podpory. Dále nabízí většinu hlavních funkcionalit již v základu a nemusíme tak používat doplňky třetích stran. Šablona projektu je ASP.NET Core 2.0 API bez autentizace. Je tak vytvořena standartní struktura s konfigurací, kterou v jednotlivých kapitolách upravujeme.

6.7.1 Verzování

Je vhodné pro udržitelnost API, zejména kvůli zpětné kompatibilitě se staršími klienty. Použitá je možnost specifikace verze v požadavku vlastní hlavičkou. Nemusíme tak upravovat routy a komplikovat tím požadavky. Tato varianta také umožňuje nastavení defaultní verze při nespecifikování, v našem případě jde o verzi 1.0. Pro vyžádání verze stačí do požadavku přidat hlavičku **`api-version`.**

Kód 7: Verzování API

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddApiVersioning(o => // Zapnutí verzování
    {
        // Nastavení metody z hlavičky
        o.ApiVersionReader = new HeaderApiVersionReader("api-version");

        // Nastavení defaultní verze na 1.0, pokud není specifikována hlavička
        o.AssumeDefaultVersionWhenUnspecified = true;
        o.DefaultApiVersion = new ApiVersion(1, 0);

// Použití verzováni na controllerech
namespace SeenIt.Web.Api.Controllers.v1 // Jmenný prostor controlleru 1.0
{
    [ApiVersion("1.0", Deprecated = true)] // 1.0 označeno jako zastaralé
    public class UsersController : Controller

namespace SeenIt.Web.Api.Controllers.v2 // Jmenný prostor controlleru 2.0
{
    [ApiVersion("2.0")] // 2.0
    public class UsersController : Controller
```

6.7.2 Content negotiation

Klienty nelimitujeme na jedinou reprezentaci a u všech akcí s návratovou hodnotou poskytujeme více možností. V případě požadavku na nepodporovanou reprezentaci vracíme status kód 406. Defaultní reprezentace je JSON.

Kód 8: Konfigurace content negotiation

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(c =>
    {
        // Respektování Accept hlavičky z požadavku
        c.RespectBrowserAcceptHeader = true;
        // Informování o nepodporované reprezentaci (vrácení 406)
        c.ReturnHttpNotAcceptable = true;
        // Přidání reprezentací (JSON je defaultní, přidáme XML)
        c.InputFormatters.Add(new XmlSerializerInputFormatter());
        c.OutputFormatters.Add(new XmlSerializerOutputFormatter());
```

6.7.3 Status kódy

Každá akce má návratovou hodnota typu **`IActionResult`**. Tento typ umožňuje specifikaci status kódu a automatický content negotiation. Existuje mnoho předem vytvořených obalů s implicitním status kódem a můžeme vytvářet i vlastní.

Kód 9: Specifikace status kódů

```
[HttpGet]
public async Task<IActionResult> GetUsers()
{
    var users = await _mongoDb.GetUsers();
    if (!users.Any())
        return NotFound(); // Návrat 404 při nenalezení žádného uživatele

    _logger.LogInformation($"Found {users.Count} users."); // Logování

    return Ok(users); // Návrat 200 a pole uživatelů ve zvolené reprezentaci
    return StatusCode(200, users); // Také možné jako
```

6.7.4 Logování

Pro naše účely je dostačující vestavěné logování. Nemá tak bohatou funkcionalitu, jako varianty třetích stran, ale nabízí automatické logování MVC akcí bez složité konfigurace. Při neošetřené výjimce je uveden i celý stack trace a společně s rozšířením **`Serilog.Extensions.Logging.File`** je možný i zápis do souboru.

Kód 10: Konfigurace vestavěného logování

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    // Logování konzole pro information a výše
    loggerFactory.AddConsole();
    loggerFactory.AddDebug(); // Logování debug pro information a výše
    loggerFactory.AddFile("logs/log_{Date}.txt"); // Specifikace souboru
```

Akce **`GetUsers`** uvedená v kódu 9 (Specifikace status kódů) vytvoří následný log. Log je zkrácený kvůli formátování a TS zastupuje časovou značku.

```
TS [INF] Request starting HTTP/1.1 GET http://localhost:8080/api/users
TS [INF] Executing action method "…UsersController.v1.GetUsers" with
arguments(null) - ModelState is Valid
TS [INF] Found 12 users.
TS [INF] Executing ObjectResult, writing value "…Mvc.ControllerContext".
TS [INF] Executed action "…UsersController.v1.GetUsers" in 28.3626ms
TS [INF] Request finished in 30.2107ms 200 application/json; charset=utf-8
```

6.7.5 Vkládání závislostí

Neboli dependency injection. Není opět potřeba žádných doplňků třetích stran a využijeme vestavěné. Požadovanou službu stačí pouze zaregistrovat v konfiguraci a je připravena ke vložení. Příklad je viditelný v kódu 11 (Vkládání závislostí), kde vkládáme již zmíněné logování a naši službu pro přístup k databázi. Jejich využití je vidět v kódu 9 (Specifikace status kódů).

Kód 11: Vkládání závislostí

```
public interface IMongo
{
    ICollection<User> GetUsers();
}

public class Mongo : IMongo
{
    private static readonly MongoClient Client = new MongoClient(
    {
        Server = new MongoServerAddress("localhost", 27017)
    });
    private readonly IMongoDatabase Db = Client.GetDatabase("SeenIt");

    ICollection<User> IMongo.GetUsers() => Db.GetCollection<User>("Users");
}

public void ConfigureServices(IServiceCollection services)
{
    services.AddTransient<IMongo, Mongo>(); // Registrace služby
}

public class UsersController : Controller
{
    private readonly ILogger<UsersController> _logger;
    private readonly IMongo _mongoDb;

    // Služby jsou vloženy v konstruktoru controlleru
    public UsersController(ILogger<UsersController> logger, IMongo mongoDb)
    {
        _logger = logger;
        _mongoDb = mongoDb;
```

6.7.6 Cachování

Ani v cachování nás ASP.NET Core nezklame. V základu nabízí jednoduchou službu, kterou snadno vkládáním závislostí použijeme pro cachování časově náročnějších dotazů. Takovým dotazem je například výpočet globálního žebříčku, který nám stačí aktualizovat jednou za 5 minut. Služba umožňuje nastavení absolutní i posouvající se expirace. Dále prioritu záznamu pro případ docházející paměti a nutnosti promazání cache. Manuální odstranění položky je také možné.

Kód 12: Cachování výsledku získání žebříčku

```
public async Task<IActionResult> GetGlobalLeaderboard()
{
    // Pokud položka s klíčem není v cachi
    if (!_cache.TryGetValue("DummyKey", out object cachedLeaderboard))
    {
        cachedLeaderboard = GetLeaderboard(); // Vypočítej žebříček

        // Nastav expiraci na 5 minut
        var cacheEntryOptions = new MemoryCacheEntryOptions()
            .SetAbsoluteExpiration(TimeSpan.FromMinutes(5));

        // Ulož žebříček do cache pro další požadavek
        _cache.Set("DummyKey", cachedLeaderboard, cacheEntryOptions);
    }
    return Ok(cachedLeaderboard); // Vrať cachovaný žebříček
}
```

6.7.7 Databáze

Použitá je **NoSQL** databáze **MongoDB**, kde jsou data uložené ve struktuře **BSON**, vhodné zejména pro ukládání dynamických dat, což otisky představují. K přístupu využíváme **.NET MongoDB Driver**, nutná konfigurace je tak minimální a jediná větší starost je instalace MongoDB na serveru. Inicializace driveru je uvedena v kódu 11 (Vkládání závislostí) ve třídě Mongo.

Indexy

MongoDB samozřejmě podporuje indexování pro rychlejší vyhledávání. Sloupec **`_id`** je automaticky indexován, k dispozici je i indexování ostatních jednotlivých nebo složených sloupců, a to sestupně i vzestupně.

Kód 13: Vytvoření indexu v MongoDB

```
_mongoDb.Users.Indexes
    .CreateOneAsync(Builders<User>.IndexKeys.Ascending(k => k.Name));
```

Datový model

Vytváření datových modelů je podobné jako například u frameworku **NHibernate**. Jde o třídy s atributy a anotacemi upřesnění chování. Pro ID objektu se používá typ `ObjectId` dostupný z driveru. Jednotlivé atributy se dají označit jako nepovinné. Datový model `User` uváděný v předchozích ukázkách je definován v následujícím kódu společně s jeho BSON reprezentací.

Kód 14: Datový model a jeho reprezentace v MongoDB

```
public class User // Zjednodušený model uživatele
{
    [BsonId]
    public ObjectId Id { get; set; }
    public string Email { get; set; }
    public string Name { get; set; }
    public string PhotoUrl { get; set; }
    public int CollectedStickersCount { get; set; }
    public DateTime FirstTimeActive { get; set; }
    public DateTime LastTimeActive { get; set; }
}

{ // Uživatel uložený podle modelu v BSON dokumentu
    "_id" : ObjectId("59bebca0be2b9e2a34764545"),
    "Email" : "email@seenit.com",
    "Name" : "Jméno Příjmení",
    "PhotoUrl" : "https://seenit.cz/photo.jpg",
    "CollectedStickersCount" : 4,
    "FirstActive" : ISODate("2017-09-16T23:35:59.631Z"),
    "LastActive" : ISODate("2018-03-19T22:56:46.366Z")
}
```

Operace

Je možné provádět více způsoby, klasicky BSON dokumenty, buildery a fluent API. Využíváme fluent API všude kde to je možné, jeho syntaxe je velmi podobná populárními dotazovacímu .NET API `System.Linq`. Pro pokročilejší funkcionality jako například projekci však již musíme využít buildery. Základní operace vyhledání, vložení a aktualizace jsou v následujícím kódu.

Kód 15: Základní operace s fluent API v MongoDB

```
// Vyhledání uživatelů
var users = _mongoDb.GetUsers()
    .Find(u => u.CollectedStickersCount > 0) // Má alespoň 1 samolepku
    .SortByDescending(u => u.CollectedStickersCount) // Seřadíme podle počtu
    .ThenBy(u => u.Name) // Pak podle jména
    .ToList();

// Vložení nového dokumentu uživatele
await _mongoDb.Users.InsertOneAsync(new User
{
    Email = newUser.Email,
    Name = newUser.Name,
    PhotoUrl = newUser.PhotoUrl,
    CollectedStickersCount = 0,
    FirstActive = DateTime.UtcNow,
    LastActive = DateTime.UtcNow
});

// Aktualizace časové značky posledního přihlášení uživatele
var updateDefiniton = Builders<User>.Update.CurrentDate(u => u.LastActive);
_mongoDb.Users.UpdateOne(u => u.Id == newUser.Id, updateDefiniton);
```

6.7.8 Autentizace klienta

Klient při každém požadavku uvádí svůj token poskytnutý od **Google Sign-In**. Pro validaci tokenu je nutné provést požadavek na Google API, kde jako parametr vložíme token. Získáme tak JSON odpověď s nároky tokenu. Zajímají nás hlavně položky **aud** (OAuth ID API) a **azp** (OAuth ID mobilní aplikace). Pokud se tyto údaje shodují s těmi vygenerovanými Google konzolí, tak je token validní a tím uživatel ověřený. Mezi položkami je i **sub** neboli Google ID účtu uživatele, pod kterým vystupuje v rámci systému.

Autentizace je potřebná pro veškeré write operace v systému. Dále i pro některé read, především získání citlivých informací, třeba uživatelského profilu. Pokud validace neproběhne úspěšně, všechny tyto akce vracejí status kód 401.

Vytvoření a aktualizace účtu

Vystačíme s položkami získanými z validace tokenu, máme k dispozici: Google ID, email, jméno a URL profilového obrázku. Pokud profil uživatele neexistuje, vytvoříme ho, jinak aktualizujeme. Aktualizace představuje přepsání časové značky poslední aktivity a úprava položek, pokud došlo například ke změně profilového obrázku.

Kód 16: Validace tokenu z mobilní aplikace

```
var uri = $"https://www.googleapis.com/oauth2/v3/tokeninfo?id_token={token}";
var response = await new HttpClient().GetAsync(new Uri(uri));
var content = await response.Content.ReadAsStringAsync();
var token = JsonConvert.DeserializeObject<ValidatedToken>(content);

if (token.Audience != Constants.WebClientId)
    // Token není určený pro naše Web API

if (token.AuthorizedParty != Constants.MobileClientId)
    // Token nepochází z naší mobilní aplikace
```

6.7.9 Validace otisku

Po přijetí otisku provádíme validaci na základě dvou kritérií:

- **Podle ID samolepky** – Vyhledání samolepky podle ID, pokud v systému neexistuje, tak není validní a nejedná se o naši samolepku.
- **Podle lokace** – Přijetí validního ID neznamená, že uživatel neposlal otisk z jiné lokace. Porovnáme tedy koordinace s koordinacemi samolepky v systému. Položka přesnosti lokace umožňuje určení maximální povolené odchylky.

Splněním obou kritérií je otisk považován za validní a uložen do databáze pod patřičnou samolepkou. Ještě předtím ale zjistíme, jestli uživatel již samolepku nesebral, abychom ho mohli případně informovat. Více otisků v rámci jedné samolepky od jednoho uživatele nám nevadí, ba naopak.

6.8 Financování systému

Systém takového rozsahu se neobjede bez externího financování. Provoz webového API, ať už v cloudu nebo na vlastním hardwaru je nákladný. Nemluvě o rozšíření infrastruktury v případě úspěchu systému. Výdaje představuje i tisk samolepek. Sice je dle našeho návrhu možné hromadné předtisknutí, ale i přes to jsou kvalitní podklady poměrně nákladné. Následná přeprava samolepky k uživateli také není zadarmo. Podíváme se tedy na dvě možnosti financování, dotace a reklamy.

6.8.1 Dotace

Byly zmíněny již v oddílu QR kódy a pro systém by byly ideálním způsobem financování. Evropská unie nabízí škálu dotačních programů, některé se zaměřují na podporu turistiky a cestovního ruchu, což je poměrně blízko našemu návrhu. Pro

udělení dotace je však nutné splnění několika podmínek a celý proces je obsáhlou záležitostí. Z tohoto důvodu bereme dotace jako potenciální možnost financování a vsázíme na jistější řešení v podobě reklam.

6.8.2 Reklamy

Google ani zde nezaspal a skrze Google Play Services nabízí nástroje pro snadnou implementaci různých typů reklamy ze služby **AdMob**. Využijeme jediný a nejméně intrusivní typ, bannery. Zabírají malou část obrazovky a uživatele tak příliš neruší.

Podobně jako u Google Sign-In je nutné vytvoření nového AdMob projektu[13] pro naší aplikaci. Po vytvoření je vygenerováno unikátní `UnitId`, které použijeme v pohledu. V přehledu projektu je také možné sledovat statistiky aplikace, zisk za určité období a vybrat specifické poskytovatele reklam.

Přidání banneru do pohledu je snadné a provádí se globálně pro celou aktivitu. V požadavku na reklamu můžeme nastavit parametry jako pohlaví, datum narození a lokaci pro lépe cílené reklamy. V našem případě využíváme jen lokaci, díky jejímu neustálému sledování v aplikaci je přidána automaticky. Máme také možnost sledování různých callbacku, například při zavření reklamy.

Kód 17: Přidání reklamy do pohledu

```
var adView = FindViewById<AdView>(Resource.Id.ad_view);
var adRequest = new AdRequest.Builder().Build(); // Požadavek na reklamu
adView.LoadAd(adRequest); // Načtení reklamy do pohledu
```

[13] Dostupné na: https://apps.admob.com

7 Shrnutí výsledků

Hlavním cílem práce byla analýza a návrh systému pro sběr otisků bezdrátových sítí za pomocí QR kódů a gamifikace jako motivátorů uživatele. Nastaly tak tři hlavní problémy, u kterých jsme částečně nastínili řešení již v úvodu.

Prvním problémem bylo určení zdrojového zařízení pro otisky. Rozhodli jsme se pro mobilní aplikaci navrženou ve vývojovém nástroji Xamarin určenou mobilním zařízením na platformě Android. Zaměření bylo především na uživatelský požitek a možnosti otisku, které jsou ve výsledku poměrně rozsáhlé. Většinu požadované funkcionality jsme pokryli nástroji dostupnými z Google Play Services.

Druhým problémem bylo vybrání služby pro uchování a možné zpracování otisků. Jasnou volbou bylo webové API navržené konkrétně v ASP.NET Core 2.0. Tento framework nám nabídl většinu funkcionality již v základu a zbývalo téměř jen určení databáze. Využili jsme dokumentovou MongoDB kvůli dynamické struktuře otisků.

Třetím problémem byla motivace uživatelů, tedy jak celý proces generování otisků gamifikovat. Bylo nutné přijít s návrhem míst a fyzickou reprezentací QR kódu pro umožnění jejich získávání uživatelem a tím mu poskytnout cíl. Dospěli jsme k samolepkám umístěným u turistických atrakcí a na veřejných místech. Implementace gamifikace se pak odráží v celém návrhu. Podařilo se nám naplnit téměř všechny lidské motivátory, použít všechny zmíněné herní prvky a apelovat na většinu typů uživatelů.

Ve výsledku jsme dospěli k poměrně komplexnímu návrhu systému složeného z více komponent.

8 Závěry a doporučení

Práce nám přiblížila problematiku QR kódů, gamifikace a vývoje mobilní aplikace a webového API. V teoretické části jsme se detailně podívali na všechny tyto prvky zvlášť, zjistili jsme tak vhodné postupy a praktiky, na základě kterých jsme definovali požadavky na systém v druhé části práce.

Na těchto požadavcích jsme následně navrhli systém a podařilo se nám je dodržet všechny s drobnými odchylkami, na které se nyní podíváme. Jde především o problémy spojené s negativně vlivnými uživateli. Prvním je reprezentace QR kódu, samolepky sice splňují všechna uvedená kritéria. Může ale nastat problém s vandalismem, kde se uživatel po oskenování rozhodne poškodit samolepku a tím odebrat možnost její získání dalším uživatelem, což pro nás znamená méně otisků.

Druhým problémem je zasílání falešných otisků pro podvádění a rychlejší sběr samolepek. Tomuto scénáři se snažíme aktivně bránit, ale vynalézavý uživatel si cestu může najít, jak je zřejmé z větších geolokačních projektů.

Posledním potencionálním problémem je větší náročnost na spotřebu energie mobilního zařízení. Vyžadujeme neustále zapnuté lokalizační služby, když je aplikace na popředí, což se negativně projeví na výdrži baterie především na starších zařízeních.

Tato práce by mohla být dále využita jako podklad pro vývoj a následný provoz navrhovaného systému. Je zde také plno místa pro rozšíření tématu. Jednou cestou je zpracování získaných otisků například promítnutím pokrytí bezdrátovými sítěmi na mapě. Další je zahrnutí návrhu přesnější lokalizace uvnitř budovy na základě Wi-Fi sítí, kde standartní metody nejsou často dostačující. Navázání na problematiku v rámci diplomové práce je tedy možné.

9 Seznam použité literatury

[1] QR code. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-03-16]. Dostupné z: https://en.wikipedia.org/wiki/QR_code

[2] PANIGRAHY, Nilanchala. Xamarin Mobile Application Development for Android. Second Edition. Birmingham, UK: Packt Publishing, 2015. 296. ISBN 978-1-78528-037-5

[3] Mobile Operating System Market Share Worldwide. In: StatCounter Global [online]. [cit. 2018-03-12]. Dostupné z: http://gs.statcounter.com/os-market-share/mobile/worldwide

[4] Android Oreo. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-03-13]. Dostupné z: https://en.wikipedia.org/wiki/Android_Oreo

[5] Getting Started with the NDK. In: Android NDK | Android Developers [online]. [cit. 2018-03-13]. Dostupné z: https://developer.android.com/ndk/guides/index.html

[6] Android Runtime. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-03-13]. Dostupné z: https://en.wikipedia.org/wiki/Android_Runtime

[7] The Activity Lifecycle. In: Android Developers [online]. [cit. 2018-03-14]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle.html

[8] Fragments. In: Android Developers [online]. [cit. 2018-03-14]. Dostupné z: https://developer.android.com/guide/components/fragments.html

[9] App Manifest Overview. In: Android Developers [online]. [cit. 2018-03-14]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-intro.html

[10] Services. In: Android Developers [online]. [cit. 2018-03-14]. Dostupné z: https://developer.android.com/guide/components/services.html

[11] UserDictionary. In: Android Developers [online]. [cit. 2018-03-14]. Dostupné z: https://developer.android.com/reference/android/provider/UserDictionary.html

[12] Xamarin. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-03-14]. Dostupné z: https://en.wikipedia.org/wiki/Xamarin

[13] MCLEMORE, Mark, Brad UMBAUGH a Craig DUNN. Architecture. In: Xamarin Documentation - Xamarin | Microsoft Docs [online]. 2018 [cit. 2018-03-14]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/android/internals/architecture

[14] MCLEMORE, Mark a Craig DUNN. API Design. In: Xamarin Documentation - Xamarin | Microsoft Docs [online]. 2018 [cit. 2018-03-16]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/android/internals/api-design

[15] BLOCK, Glenn, Pablo CIBRARO, Pedro FÉLIX, Howard DIERKING a Darrel MILLER. Designing evolvable web APIs with ASP.NET. Sebastopol, CA: O'Reilly Media, 2014. ISBN 978-1-449-33771-1.

[16] Uniform Resource Identifier. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-02-23]. Dostupné z: https://cs.wikipedia.org/wiki/Uniform_Resource_Identifier

[17] BERNERS-LEE, Tim. Uniform Resource Identifier (URI): Generic Syntax: URI, URL, and URN [online]. 2005 [cit. 2018-02-24]. Dostupné z: http://www.ietf.org/rfc/rfc3986.txt

[18] JSON vs XML. In: W3Schools Online Web Tutorials [online]. [cit. 2018-02-24]. Dostupné z: https://www.w3schools.com/js/js_json_xml.asp

[19] Hypertext Transfer Protocol. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-02-24]. Dostupné z: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[20] HTTP - Requests. In: HTTP Tutorial [online]. [cit. 2018-02-24]. Dostupné z: https://www.tutorialspoint.com/http/http_requests.htm

[21] HTTP - Responses. In: HTTP Tutorial [online]. [cit. 2018-02-24]. Dostupné z: https://www.tutorialspoint.com/http/http_responses.htm

[22] HTTP Methods: GET vs. POST. In: W3Schools Online Web Tutorials [online]. [cit. 2018-02-24]. Dostupné z: https://www.w3schools.com/tags/ref_httpmethods.asp

[23] FIELDING, Roy. Hypertext Transfer Protocol -- HTTP/1.1: Status Code Definitions [online]. 1999 [cit. 2018-02-24]. Dostupné z: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[24] HARDT, Dick. The OAuth 2.0 Authorization Framework [online]. 2012 [cit. 2018-02-27]. Dostupné z: https://tools.ietf.org/html/rfc6749

[25] Representational state transfer. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-02-27]. Dostupné z: https://en.wikipedia.org/wiki/Representational_state_transfer

[26] HATEOAS. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-02-27]. Dostupné z: https://en.wikipedia.org/wiki/HATEOAS

[27] WASSON, Mike. Routing in ASP.NET Web API. In: Web API Guidance | Microsoft Docs [online]. 2012 [cit. 2017-02-27]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api

[28] Controllers and Action Methods in ASP.NET MVC Applications. In: Learn to Develop with Microsoft Developer Network | MSDN [online]. [cit. 2018-02-28]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd410269%28v=vs.98%29.aspx

[29] KURTZ, Jamie a Brian WORTMAN. ASP.NET Web API 2: Building a REST Service from Start to Finish. Second Edition. New York, NY: Apress, 2014. Expert's voice in ASP.NET. ISBN 978-1-4842-0110-7.

[30] ZICHERMANN, Gabe a Christopher CUNNINGHAM. Gamification by design: Implementing Game Mechanics in Web and Mobile Apps. Sebastopol, CA: O'Reilly Media, 2011. 208. ISBN 978-1-4493-9767-8.

[31] PELLING, Nick. The (short) prehistory of "gamification". In: Funding Startups (& other impossibilities): Getting to "yes" in a world of "no" [online]. 2011 [cit. 2018-01-27]. Dostupné z: https://nanodome.wordpress.com/2011/08/09/the-short-prehistory-of-gamification

[32] MANGALINDAN, Jp. Play to win: The game-based economy. In: Fortune [online]. 2010 [cit. 2018-01-27]. Dostupné z: https://web.archive.org/web/20121112074424/http://tech.fortune.cnn.com/2010/09/03/the-game-based-economy [arch. 2012-11-12]

[33] Gamification: History. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-01-27]. Dostupné z: https://en.wikipedia.org/wiki/Gamification#History

[34] DETERDING, Sebastian, Rilla KHALED, Lennart NACKE a Dan DIXON. Gamification: Toward a definition. CHI 2011 Gamification Workshop Proceedings [online]. 2011 [cit. 2017-12-05]. Dostupné z: https://www.researchgate.net/publication/273947177_Gamification_Toward_a_definition

[35] CHOU, Yu-kai. What is Gamification. In: Yukaichou [online]. 2015 [cit. 2017-12-05]. Dostupné z: http://yukaichou.com/gamification-examples/what-is-gamification

[36] MARCZEWSKI, Andrzej. What's the difference between Gamification and Serious Games?. In: Gamasutra – The Art & Business of Making Games [online]. 2013 [cit. 2018-01-27]. Dostupné z: https://www.gamasutra.com/blogs/AndrzejMarczewski/20130311/188218/Whats_the_difference_between_Gamification_and_Serious_Games.php

[37] VANHEMERT, Kyle. IBM CityOne Is SimCity For the Real World. In: Gizmodo – We come from the future. [online]. 2010 [cit. 2018-01-27]. Dostupné z: https://gizmodo.com/5530030/ibm-cityone-is-simcity-for-the-real-world

[38] CHOU, Yu-kai. Octalysis – complete Gamification framework. In: Yu-kai Chou: Gamification & Behavioral Design [online]. 2015 [cit. 2017-12-05]. Dostupné z: http://yukaichou.com/gamification-examples/octalysis-complete-gamification-framework

[39] Bartle taxonomy of player types. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-12-2]. Dostupné z: https://en.wikipedia.org/wiki/Bartle_taxonomy_of_player_types

[40] MARCZEWSKI, Andrzej. User Types. In: Gamified [online]. 2015 [cit. 2017-12-2]. Dostupné z: https://www.gamified.uk/user-types

[41] BAIDYA, Amartya. Mobile App Retention Challenge: 75% Users Uninstall An App Within 90 Days. In: Dazeinfo: Business News On Smartphone, Social Media, Ecommerce [online]. 2016 [cit. 2018-04-09]. Dostupné z: https://dazeinfo.com/2016/05/19/mobile-app-retention-churn-rate-smartphone-users/

[42] VONNEGUT, Sarah. Mobile Application Security: 15 Best Practices for App Developers. In: Checkmarx - Application Security Testing and Static Code Analysis [online]. 2015 [cit. 2018-04-09]. Dostupné z: https://www.checkmarx.com/2015/08/19/mobile-application-security-15-best-practices-for-app-developers/

[43] BABICH, Nick. The Guide to Mobile App Design: Best Practices for 2018 and Beyond. In: UXPin Blog [online]. 2018 [cit. 2018-04-09]. Dostupné z: https://www.uxpin.com/studio/blog/guide-mobile-app-design-best-practices-2018-beyond/

[44] JAUKER, Sarah. Mobile Application Security: 15 Best Practices for App Developers. In: Thinking Mobile | Thoughts & ideas on enterprise mobility [online]. 2014 [cit. 2018-04-10]. Dostupné z: https://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/

[45] Dashboards. In: Android Developers [online]. [cit. 2018-04-10]. Dostupné z: https://developer.android.com/about/dashboards/index.html

10 Přílohy

Příloha 1: Vypočet bodů za sebrání samolepky

Příloha 2: Rozložení úrovní

Příloha 3: Návrh loga a samolepky systému

Příloha 4: Návrh pohledů mobilní aplikace 1

Příloha 5: Návrh pohledů mobilní aplikace 2

Příloha 6: Návrh pohledů mobilní aplikace 3

Příloha 1: Vypočet bodů za sebrání samolepky

Kód 18: Implementace výpočtu bodů

```
const double BasePoints = 1000.0;
const double RarityBasePoints = 350.0;
const double SpeedBasePoints = 150.0;
const double MaxTimeDiffHours = 168.0;

// Výpočet bodů vzácnosti
var rarityFactor = (1.0 - (double) usersWithStickerCount / usersTotalCount)
    .Clamp();
var rarityPoints = RarityBasePoints * rarityFactor;

// Výpočet bodů rychlosti
var timeDiffHours = (sticker.Timestamp - user.LastTimestamp).TotalHours;
var speedFactor = ((MaxTimeDiffHours - timeDiffHours) / MaxTimeDiffHours)
    .Clamp();
var speedPoints = speedFactor * SpeedBasePoints;

// Finální body
var finalPoints = (int) Math.Ceiling(BasePoints + rarityPoints + speedPoints);

// Extension metoda na limitováni hodnoty rozsahem
public static double Clamp(this double value, double min = 0, double max = 1)
{
    if (value.CompareTo(min) < 0)
        return min;
    if (value.CompareTo(max) > 0)
        return max;
    return value;
}
```

Základ – Bázová hodnota, tj. konstanta, konkrétně 1000. Je to minimální počet bodů, které uživatel obdrží za samolepku za jakékoliv okolnosti.

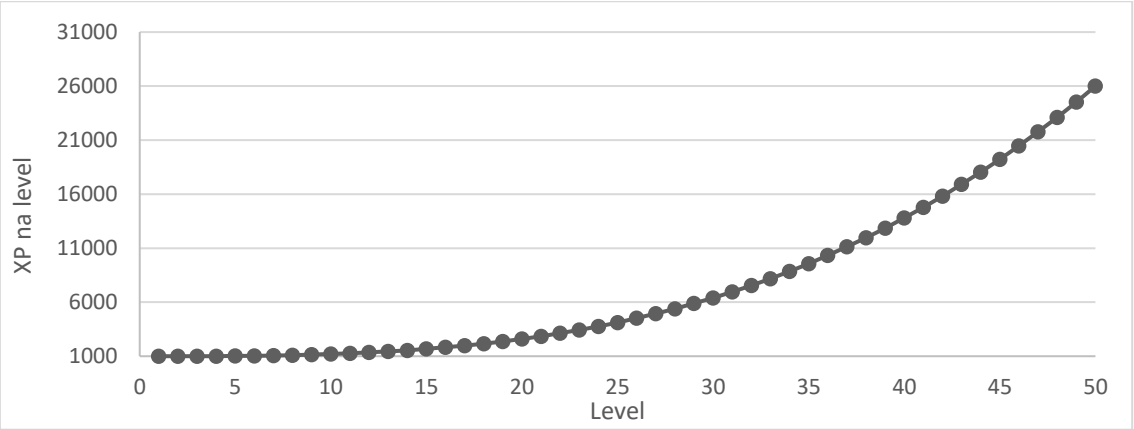
Vzácnost – Odvíjí se od vzácnosti samolepky, tedy kolik procent uživatelské základy danou samolepku vlastní. Bázovou hodnota je 350 a je násobena lineárním faktorem vzácnosti.

Rychlost – Závisí na uplynulém času od sebrání poslední samolepky. Bázová hodnota je 150 a je násobena lineárním faktorem rychlosti.

Příloha 2: Rozložení úrovní

Tabulka 4: Přehled bodů a samolepek na úroveň

Úroveň	XP na úroveň	Kumulativní XP	Počet samolepek na úroveň (nejhorší)	Počet samolepek na úroveň (nejlepší)
1	1 000	1 000	1	1
2	1 001	2 001	3	2
3	1 005	3 006	4	3
4	1 012	4 018	5	3
5	1 025	5 043	6	4
6	1 043	6 086	7	5
7	1 068	7 154	8	5
8	1 102	8 256	9	6
9	1 145	9 401	10	7
10	1 200	10 601	11	8
11	1 266	11 867	12	8
12	1 345	13 212	14	9
13	1 439	14 651	15	10
14	1 548	16 199	17	11
15	1 675	17 874	18	12



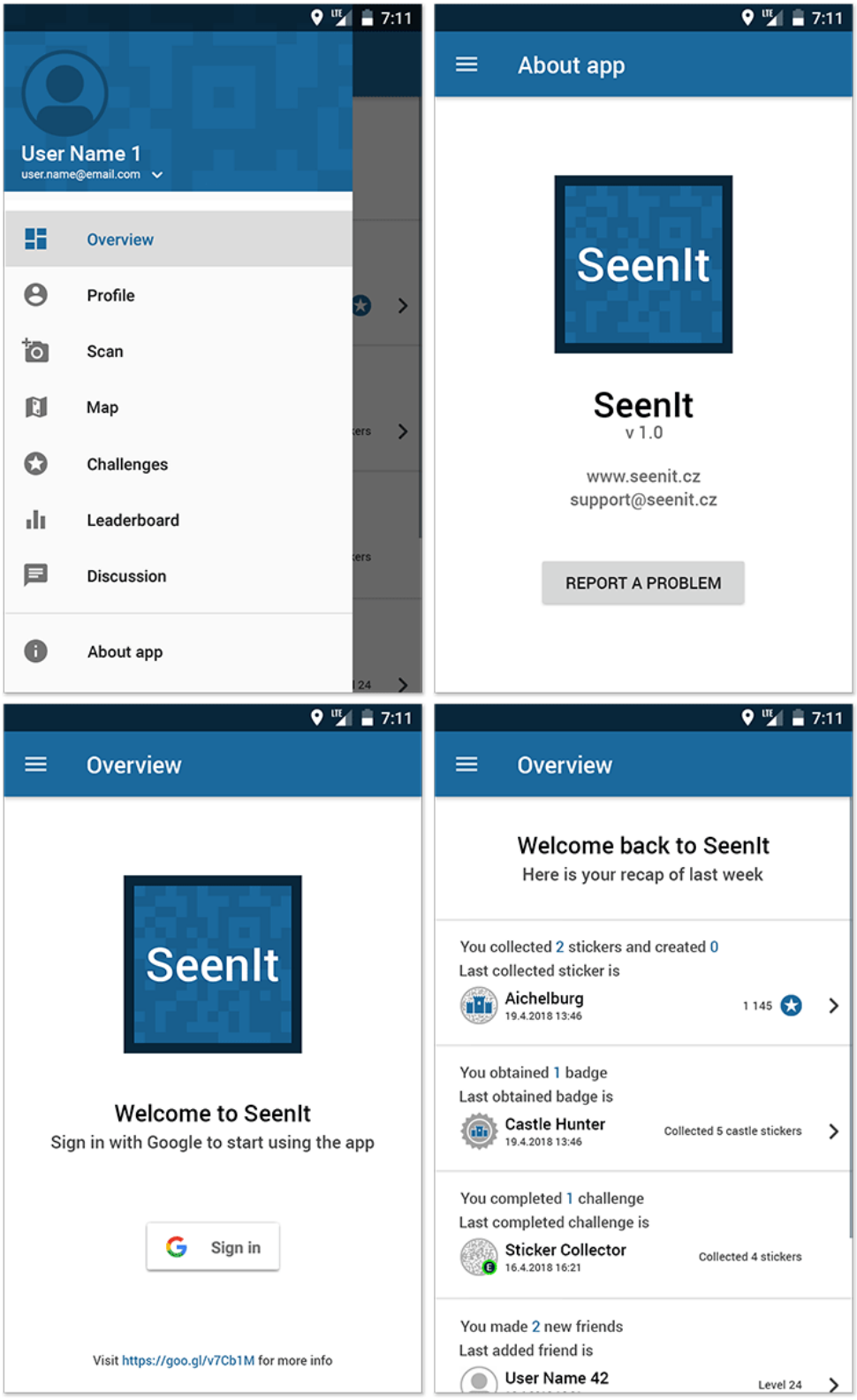
Obrázek 14: Nárůst potřebných bodů na level

Příloha 3: Návrh loga a samolepky systému

Obrázek 15: Návrh loga

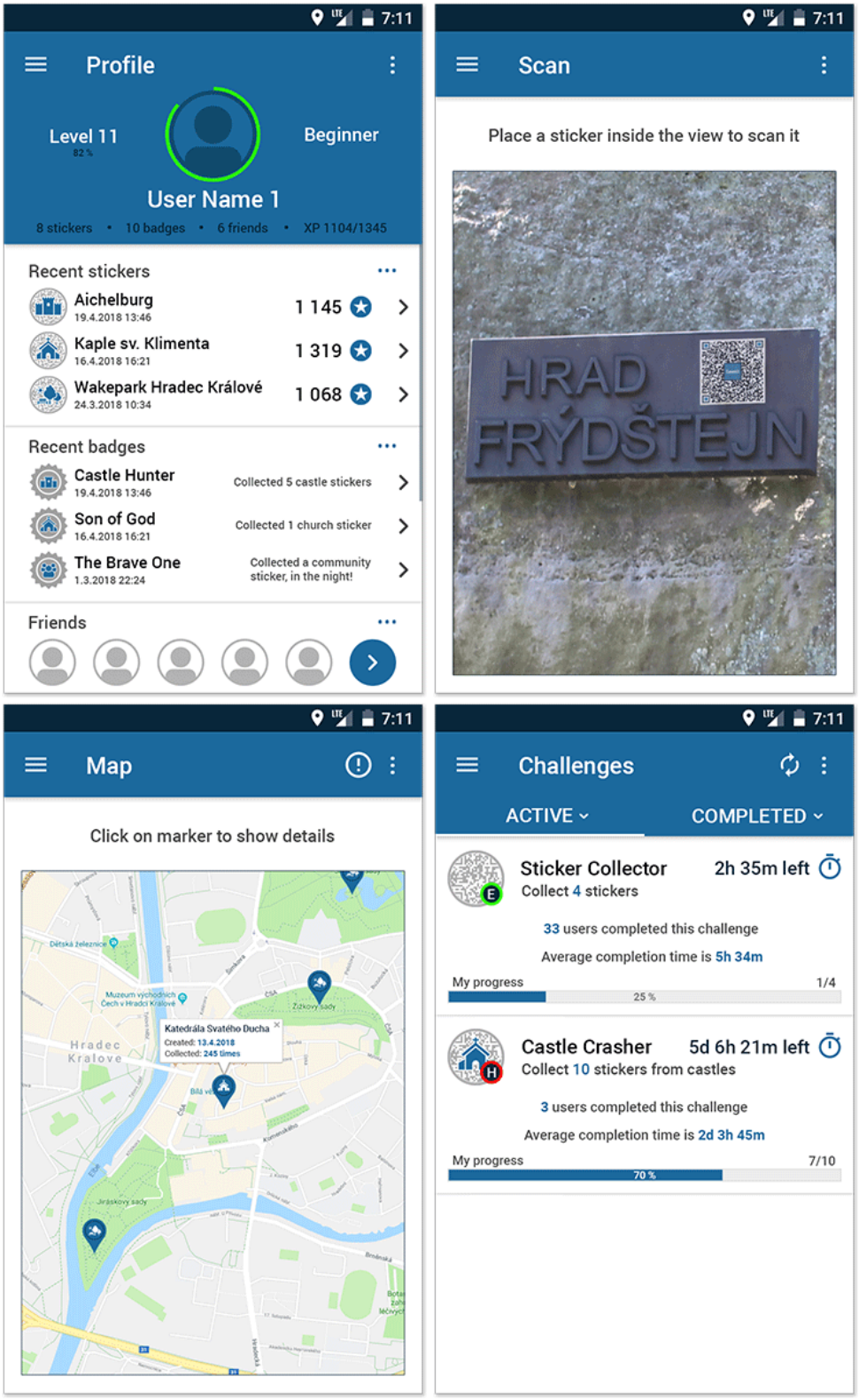
Obrázek 16: Návrh samolepky s QR kódem

Příloha 4: Návrh pohledů mobilní aplikace 1



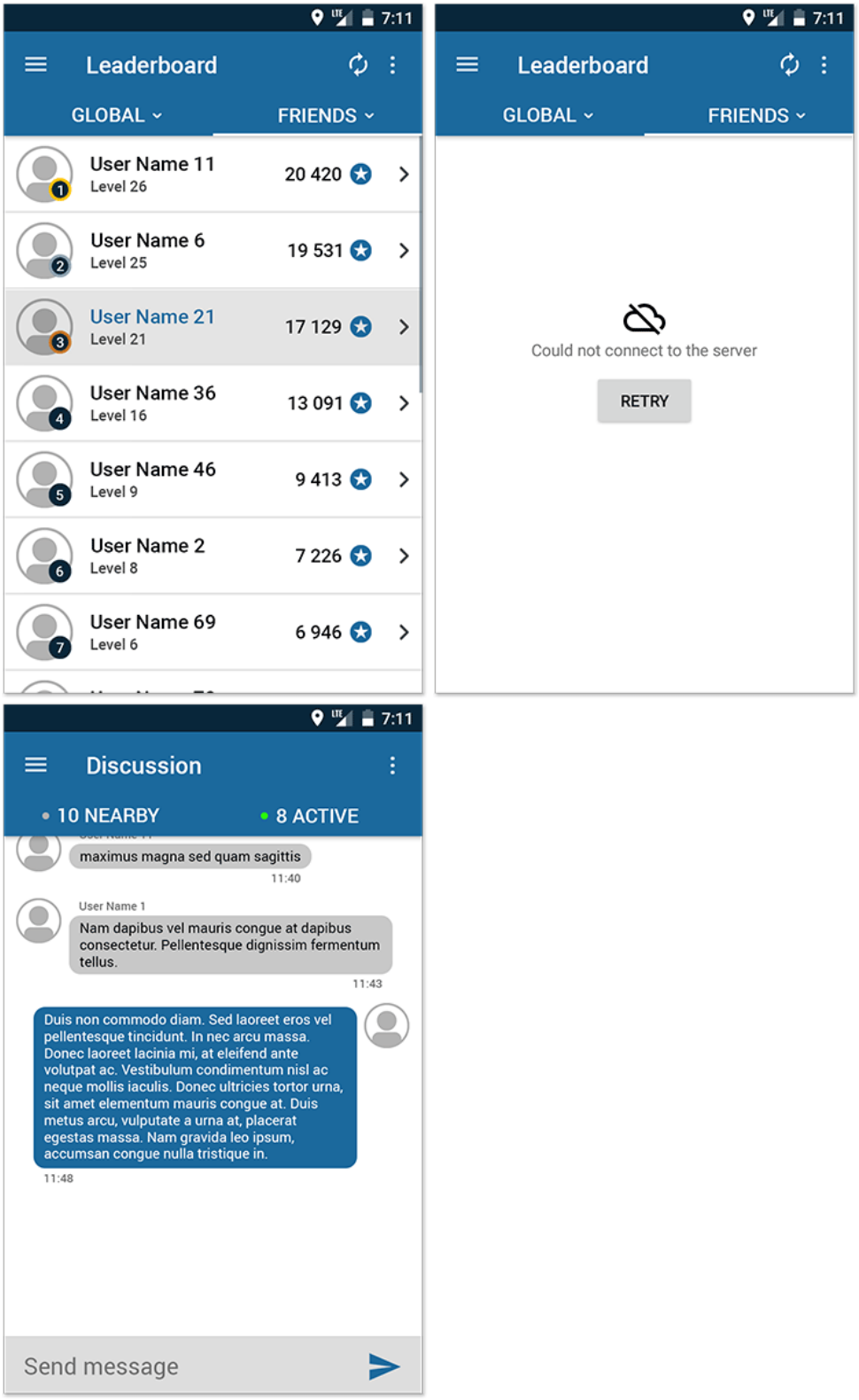
Obrázek 17: Návrh pohledů mobilní aplikace 1

Příloha 5: Návrh pohledů mobilní aplikace 2



Obrázek 18: Návrh pohledů mobilní aplikace 2

Příloha 6: Návrh pohledů mobilní aplikace 3



Obrázek 19: Návrh pohledů mobilní aplikace 3

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Chvátal Lukáš	Kostelní 484, Plesná	I1500370

TÉMA ČESKY:

Sběr fingerprintů s využitím QR kódů a principů gamifikace

TÉMA ANGLICKY:

Collecting fingerprints using QR codes and gamification principles

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je analýza a návrh systému pro sběr informací o bezdrátových sítích v dané oblasti s využitím QR kódů a principů gamifikace se zaměřením na platformu Android využívající Web API.

Osnova:
1. Úvod
2. Gamifikace
3. Xamarin Android
4. Web API
5. Analýza a návrh systému
6. Shrnutí výsledků, Závěr, Literatura

SEZNAM DOPORUČENÉ LITERATURY:

ZICHERMANN, Gabe a CUNNINGHAM Christopher. Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps. 1. Sebastopol, CA: O'Reilly Media, 2011. ISBN 978-1-449-39767-8.

BLOCK, Glenn, Pedro FELIX, Howard DIERKING, Darrel MILLER a Pablo CIBRARO. Designing Evolvable Web APIs with ASP.NET: Harnessing the Power of the Web. 1. Sebastopol, CA: O'Reilly Media, 2014. ISBN 978-1-449-33771-1.

PANIGRAHY, Nilanchala. Xamarin Mobile Application Development for Android. 2. Birmingham, UK: Packt Publishing, 2015. ISBN 978-1-78528-037-5.

Podpis studenta: ..... **Datum:** 16.4.2018

Podpis vedoucího práce: ..... **Datum:** 16.4.2018