**Univerzita Hradec Králové**

**Fakulta informatiky a managementu**

**Katedra informatiky a kvantitativních metod**

# Gamifikace sběru rádiových fingerprintů

Diplomová práce

Autor: Kateřina Hanušová

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž, PhD.

Hradec Králové srpen 2017

Prohlášení:

Prohlašuji, že jsem diplomovou práci *Gamifikace sběru rádiových fingerprintů* zpracovala samostatně a s použitím uvedené literatury.

V Hradci Králové dne Kateřina Hanušová

Poděkování:

Chtěla bych poděkovat svému vedoucímu Ing. Pavlu Křížovi, PhD. za odborné vedení, pomoc a rady při zpracování diplomové práce a Kristýně Kamenické za grafické zpracování map.

# Anotace

Diplomová práce sestává z teoretické a praktické části. Její součástí je také aplikace pro mobilní telefony s operačním systémem Android a serverová aplikace. Práce se zabývá tvorbou hry s cílem nasbírání dat pro výzkum indoor lokalizace. V teoretické části se zabývá způsobem lokalizace. Jsou zde popsána data, která se budou sbírat, a očekávané výsledky. Následuje analýza a návrh řešení. Potom je popsán proces vývoje software a podle těchto principů je vyvíjena aplikace. Samotný vývoj jak klientské, tak serverové části je popsán v dalších kapitolách. Nakonec jsou zpracována posbíraná data a vyhodnoceny výsledky.

# Annotation

**Title**: Gamification of radio-fingerprint acquisition

Diploma thesis consists of a theoretical part and a practical part. It also includes an Android based mobile application and a server-side application from collected data. The work deals with game creation in order to obtain data for indoor localization research. The theoretical part deals with the localization method and describes the data to be collected and the expected outcomes. The practical part contains the analysis and the suggestion of a solution. Then, the software development is described and the application is developed according to these principles. The development of both client and server components are described in this part. Finally, the collected data and outcomes are evaluated.

# Obsah

# 1 Úvod

Diplomová práce se zabývá tématem indoor lokalizace. Navazuje na předchozí výzkum Dominika Matoulka (2015), Vojtěcha Leona (2016) a na práci Pavla Kříže, Filipa Malého a Tomáše Kozla týkající se lokalizace za pomoci Bluetooth low energy beaconů.

V současné době je všem přístupná dostatečně přesná lokalizace ve venkovních prostorách, zejména na základě GPS. Ve vnitřních prostorách k ní však stále nebyl nalezen vhodný ekvivalent. Existují již metody lokalizace uvnitř budov, ale jsou stále nepřesné. Asi nejnovější možností je k lokalizaci využít Bluetooth low energy zařízení. Jejich možnosti jsou stále zkoumány. Výzkumu těchto možností se věnuje i FIM UHK. Je však potřeba průběžně získávat množství dat a pouze v několika málo lidech je to náročné – jak z časového hlediska, tak kvůli tomu, že je vhodné mít co nejvíce zařízení.

Hlavním cílem práce je pro podporu tohoto výzkumu vytvořit hru na mobilní zařízení, prostřednictvím které bude možné posbírat data. Hra by se měla odehrávat v prostorách FIM UHK a měla by být určena primárně pro studenty. Při této hře by studenti chodili na předem určená místa ve škole, kde by prováděli nějaké akce. Při těchto akcích by byla data pořízena. Tím by bylo docíleno pravidelné získávání množství záznamů z různých zařízení.

Pro potvrzení funkčnosti hry budou jejím prostřednictvím posbírána a následně vyhodnocena data. Vyhodnocení nebude spočívat přímo v lokalizaci, ale v tom, jaká data se podařilo posbírat a zda jsou pro lokalizaci použitelná.

Text je psán tak, aby primárně sloužil lidem, kteří budou na projektu pokračovat. Některé ustálené výrazy (např. beacon) budou ponechány v angličtině. Stejně tak výrazy, ke kterým nebude nalezen vhodný, dostatečně vypovídající, český ekvivalent nebo by jejich překlad mohl způsobit nějaké nejasnosti.

# 2 Indoor lokalizace

V dnešní době existuje několik způsobů, jak lokalizovat mobilní telefon. Většinou jsou založeny na základě GSM, GPS nebo jejich kombinaci.

GSM síť je tvořena skupinou buněk o různých velikostech. Jedná se o makro, mikro, piko a deštníkové buňky. Ty mohou nabývat velikosti od několika metrů až do desítek kilometrů (Comdeal, s.r.o., 2003-2011). Zařízení je lokalizováno pomocí znalostí o přesném umístění vysílacích stanic a chování radiových vln. Výhodou je nízká spotřeba baterie a vysoké pokrytí signálem, který projde i pevnou překážkou. Existuje několik možností zjištění aktuální polohy pomocí GSM signálu, ale výsledek stále zůstává velice nepřesný – přesnost se pohybuje mezi desítkami metrů až desítkami kilometrů dle zvolené metody (Kocman, 2011).

Další metodou je lokalizace na základě GPS. Ta využívá družice na oběžné dráze Země a dokáže zjistit polohu s přesností do deseti metrů. Výhodou této metody je, že signál může využívat neomezený počet uživatelů, je zdarma a lokalizace je přesnější než v případě GSM. Vedle těchto výhod má ale tato metoda jednu velkou nevýhodu, kterou je požadavek přijímače na nezakrytý výhled na oblohu, tzn. v budovách nefunguje vůbec a v zastavěných oblastech může mít s lokalizací problémy. Další nevýhodou je, že proces lokalizace trvá až několik minut od zapnutí přístroje (Bergmann, 2005).

Kombinací dvou výše uvedených způsobů lze dosáhnout rychlejší a přesnější lokalizace zařízení při pobytu venku. V budovách či hustě zastavěných oblastech však zůstává nepřesnost GSM lokalizace. Pro tyto případy jsou vyvíjena jiná řešení, většinou založena na radiových sítích (např. IEEE 802.11-WiFi) a zjištění síly jejich signálů poskytovaných jednotlivými zařízeními vysílajícími signál uvnitř budovy. Přesnost je ovlivněna několika okolnostmi, například vlastnostmi vysílačů a přijímačů a vlastnostmi prostředí, což ovlivňuje přenos signálu. Kombinací s Bluetooth Low Energy (dále jen BLE) technologií lze dosáhnout přesnější lokalizace. Výhodou použití BLE je mj. to, že jejich vysílače, označované jako beacony, mohou být napájeny bateriemi, což umožňuje jejich použití v místech, kde by bylo složité řešit napájení WiFi přístupových bodů (Kříž, Malý, Kozel, 2016).

Stejně jako u lokalizace pomocí GSM jsou i lokalizační metody pomocí WiFi nebo BLE založeny na výpočtu polohy podle síly signálu a u některých metod i na znalostech o

přesném umístění zařízení, tzn. se zvyšující se vzdáleností od vysílače klesá síla signálu a s rostoucím počtem vysílačů roste vypočítaná přesnost polohy zařízení. Základními přístupy jsou triangulace a fingerprinting (Kříž, Malý, Kozel).

## 2.1 Triangulace

Metody triangulace jsou založeny na výpočtu úhlu nebo vzdálenosti signálu (měřenou časem nebo sílou signálu). Například metoda Timing Advance vypočítává polohu na základě doby šíření signálu mezi mobilním zařízením a sítí. Enhanced Observed Time Difference porovnává časové rozdíly mezi příchody signálů od tří a více vysílacích stanic. Angle of Arrival používá k výpočtu směrové antény a znalosti o vyzařovacích charakteristikách – je změřen úhel, pod kterým je přijat signál a poloha je vypočítána jako průnik přímek procházejících vysílačem a přijímačem (Orlich, 2006).

Všechny tyto metody jsou však velice nepřesné při použití v budovách vzhledem k ovlivňování signálu okolním prostředím. Signál se může od překážek různě odrážet a slábnout a tím se stává lokalizace nepřesnou. Více informací k metodám triangulace uvádí například Orlich (2006) a nebudou v této práci dále rozebírány, protože se věnuje lokalizaci pomocí fingerprintů.

## 2.2 Fingerprinting

Fingerprinting je lokalizační metoda sestávající ze dvou fází. V první fázi jsou sesbírány vektory sestávající z RSSI[1] hodnot a dalších nepovinných hodnot naměřených zařízeními ve známých oblastech. Tyto hodnoty jsou uloženy dohromady se souřadnicemi do databáze fingerprintů pro potřeby lokalizace. Druhou fází je samotná lokalizace. V ní zařízení naměří RSSI hodnoty a porovná je s daty v databázi fingerprintů za použití vhodné metody. Nejpoužívanějšími metodami pro výpočet polohy jsou:

- pravděpodobnostní metody
- k-nejbližších sousedů
- neuronové sítě
- support vector machine

[1] Received Signal Strength Indication – indikátor síly přijímaného signálu, udává se v dBm

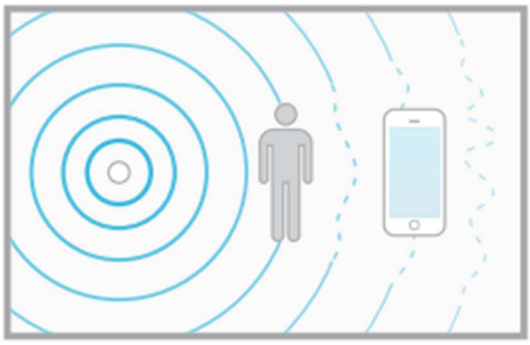
- smallest M-vertex polygon

Přesnost může být ještě zvýšena, pokud je brán v potaz pohyb lokalizovaného objektu, případně přidáním dat z pohybových senzorů zařízení (například akcelerometr, gyroskop apod.) (Kříž, Malý, Kozel, 2016).

### 2.2.1 Beacony

IBeacon je technologie přinášející nové možnosti lokalizace pro aplikace. Tato technologie je využívána Bluetooth energy beacony a může být použita například ke stanovení oblasti, kde se objekt nachází. Mobilní zařízení tak mohou pomocí BLE zjistit, že se dostali do okolí beacon. Výhodou BLE zařízení je, že jsou napájena bateriemi a dokáží fungovat i několik měsíců v kuse. Pro použití na delší dobu mají některá BLE zařízení možnost napájení ze sítě. Většina dnes používaných lokalizačních zařízení je založena na definování geografické lokace (zeměpisných souřadnic). Výhodou beacon je, že se nemusí vztahovat k jednomu konkrétnímu místu, tj. nejsou vázány na přesné geografické souřadnice, ale mohou být umístěny i na pohyblivém objektu pro definování jeho oblasti, například na autě nebo na lodi (Apple Inc., 2014).

Při lokalizaci pomocí BLE se používá RSSI jak pro odhad vzdálenosti, tak i pro stanovení přesnosti vypočítané vzdálenosti. Čím silnější je signál, tím přesnější je odhad. BLE využívá 2,4 GHz frekvence, které mohou být tlumeny různými materiály, například zdmi, dveřmi nebo jinými fyzickými objekty. Protože signál může být ovlivňován i vodou, lidské tělo také tlumí signál. Kvůli tomu se potom sledované zařízení zdá být dál, než ve skutečnosti je, jak znázorňuje Obrázek 1. Z tohoto důvodu Apple doporučuje používat iBeacon spíše pro odkaz polohy s přesností na místnosti (Apple Inc., 2014).



*Obrázek 1 Rušení signálu lidským tělem*

*Zdroj: Apple Inc., 2014*

### 2.2.2 Android podpora BLE

Android verze 4.3 představil podporu BLE a začal poskytovat API umožňující s těmito zařízeními komunikovat. Předpokládaným využitím je přenos malého objemu dat mezi zařízeními nebo pro poskytování informací založených na aktuální pozici uživatele (například reklamy v obchodě) prostřednictvím beacon. Na rozdíl od klasického Bluetooth je BLE navrženo tak, aby spotřebovávalo méně energie. Díky tomu je prostřednictvím nového API možné komunikovat i s BLE zařízeními, která mají přísnější požadavky na úsporu energie, například fitness doplňky nebo proximity senzory. Android také umožňuje definovat, že chce komunikovat pouze se zařízeními podporujícími BLE komunikaci (Android developers, 2017).

Problémem při použití Android API pro BLE komunikaci je to, že umožňuje vyhledávat dostupná zařízení pouze pomocí callbacku, nikoli opakovaně vyhledávat zařízení po požadovanou dobu. Z výkonnostních důvodů ani ve své dokumentaci (2017) nedoporučuje používat vyhledávání zařízení ve smyčce. Pro potřeby lokalizace je však nutné, aby bylo možné cyklicky skenovat dostupná zařízení v zadaném časovém intervalu. Nelze tedy používat pouze Android API, ale je nutné implementovat nějaký mechanismus pro tento typ skenu nebo použít knihovnu, která umožňuje skenovat tímto způsobem.

### 2.2.3 Reprezentace fingerprintu

Jak bylo řečeno výše, při lokalizaci je vhodné kombinovat různé přístupy, například použít BLE beacony tam, kde je slabé nebo zcela nedostupné pokrytí signálem Wi-Fi. Z tohoto důvodu se tato práce zabývá lokalizací pomocí kombinace GSM, Wi-Fi a BLE. Do databáze jsou ukládány postupně sesbírané fingerprinty obsahující informace vedoucí k lokalizaci zařízení. Ty sestávají z informací všech dostupných sítích.

Reprezentace fingerprintu navazuje na předchozí práce týkající se výzkumu lokalizace uvnitř budov, konkrétně na práce Dominika Matoulka (2015), Vojtěcha Leona (2016) a na experimenty v rámci článku Pavla Kříže, Filipa Malého a Tomáše Kozla (2016). Z důvodu konzistence dat a existujících řešení se fingerprint neměnil, proto zde není více rozepsán důvod použití jednotlivých atributů, ale jen jejich popis.

U Wi-Fi sítí je třeba uložit BSSID[2], sílu a frekvenci signálu. U GSM je potřeba zjistit konkrétní CellID, tedy jednoznačný identifikátor buňky, ve které se zařízení nachází a čas

[2] Basic Service Set Identifier, MAC adresa přístupového bodu

mezi začátkem vysílání a obdržením signálu. Pro Bluetooth jsou ukládána data týkající se jednoznačné identifikace vysílače (tedy MAC[3], UUID[4] a major a minor čísla) spolu s dobou obdržení signálu.

Seznam těchto informací je doplněn o předpokládané souřadnice zařízení pro porovnání přesnosti lokalizace. Dále následují informace o lokalizovaném zařízení, aby bylo v budoucnu možno zjistit odchylky měření způsobené konkrétním zařízením, případně typem zařízení.

Všechny lokalizační metody používají vektor naměřených hodnot na místě, ale pro účely výzkumu lokalizace je vhodné použít více měření pro eliminaci extrému. Z více naměřených hodnot z jednoho vysílače je vždy vybrán pro výpočty jejich medián. Aby bylo možné posbírat dostatek informací o naměřených sítích, bude měření probíhat vždy deset sekund.

## 2.3 Metody lokalizace

Lokalizace uvnitř budovy probíhá za pomoci kolekce více fingerprintů. Lokalizovaný klient vytvoří fingerprint místa, kde se nachází. Ten je následně porovnáván se všemi fingerprinty uloženými v databázi a hledá se jeden nebo více nejpodobnějších záznamů. Jak bylo výše uvedeno, u fingerprintů v databázi je uložena i jejich poloha. Přesnost lokalizace tedy mimo jiné závisí také na kvalitě uložených záznamů a samozřejmě na algoritmu použitém pro porovnávání záznamů a výpočet lokality (Kříž, Malý, Kozel, 2016).

### 2.3.1 K-nejbližších sousedů

Pro výpočet polohy byl vybrán algoritmus k-nejbližších sousedů z důvodu jeho jednoduchosti. Výsledky těchto výpočtů nejsou složité na interpretaci a snadno se odhalují chyby výpočtu.

Při použití tohoto algoritmu je hledáno *k* nejbližších sousedů v předem naměřené množině. U nového prvku je porovnávána vzdálenost se známými prvky v databázi. Tímto způsobem je získáno *k* nejbližších prvků a jejich kombinací je odhadnuta pozice naměřeného fingerprintu. Pro výpočet vzdálenosti existuje několik algoritmů.

[3] Media Access Control, adresa síťového zařízení
[4] Universally Unique Identifier – jednoznačný identifikátor

## Euklidovská vzdálenost

Euklidovská vzdálenost udává délku úsečky mezi dvěma body, vychází tedy z Pythagorovy věty. Euklidovská vzdálenost $D_{AB}$ dvou bodů $A = (a_1, a_2)$ a $B = (b_1, b_2)$ je definována následujícím vztahem:

$$D_{AB} = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}.$$

Z tohoto vztahu lze odvodit výpočet euklidovské vzdálenosti naměřeného vektoru fingerprintu $m = (m_1, m_2, \dots, m_n)$ od i-tého fingerprintu $s_i = (s_{i1}, s_{i2}, \dots, s_{in})$ z databáze:

$$D_i = \sqrt{\sum_{j=0}^{N} (m_j - s_{ij})^2},$$

kde $N$ je počet vysílačů v měření (Kříž, Malý, Kozel, 2016).

## Manhattanská vzdálenost

Manhattanská vzdálenost určuje vzdálenost dvou bodů, mezi kterými se lze pohybovat pouze v pravých úhlech ve směru obou os. Manhattanská vzdálenost $D_{AB}$ dvou bodů $A = (a_1, a_2)$ a $B = (b_1, b_2)$ je definována následujícím vztahem:

$$D_{AB} = |a_1 - b_1| + |a_2 - b_2|.$$

Z tohoto vztahu lze odvodit výpočet manhattanské vzdálenosti naměřeného vektoru fingerprintu $m = (m_1, m_2, \dots, m_n)$ od i-tého fingerprintu $s_i = (s_{i1}, s_{i2}, \dots, s_{in})$ z databáze:

$$D_i = \sum_{j=0}^{N} |m_j - s_{ij}|,$$

kde $N$ je počet vysílačů v měření.

## Maximová vzdálenost

Maximová vzdálenost je dána maximální vzdáleností dvou prvků. Nabývá nulové hodnoty právě tehdy, když jsou všechny prvky, ze kterých je určováno maximum, nulové. To nastane pouze ve chvíli, kdy s oba body překrývají. Maximová vzdálenost $D_{AB}$ dvou bodů $A = (a_1, a_2)$ a $B = (b_1, b_2)$ je definována následujícím vztahem:

$$D_{AB} = \max\{|a_1 - b_1|, |a_2 - b_2|\}.$$

Z tohoto vztahu lze odvodit výpočet maximové vzdálenosti naměřeného vektoru fingerprintu $m = (m_1, m_2, \dots, m_n)$ od i-tého fingerprintu $s_i = (s_{i1}, s_{i2}, \dots, s_{in})$ z databáze:

$$D_i = \max\{|m_1 - s_1|, |m_2 - s_2|, \dots, |m_n - s_n|\},$$

kde *n* je počet vysílačů v měření.

## Odhad pozice

Vzdálenosti jednotlivých vektorů od prvku jsou porovnávány zpravidla euklidovskou vzdáleností z důvodu jednoduchosti této metody. Výpočty nejsou nijak složité a snadněji je nalezena možná chyba. Tímto způsobem lze získat *k* nejbližších fingerprintů. Z těch je váženým průměrem jejich pozic vypočítaná odhadovaná pozice *P* měřeného fingerprintu dle následujícího vzorce:

$$P = \frac{\sum_{i=1}^{k} P_i Q_i}{\sum_{i=1}^{k} Q_i},$$

kde $Q_i = \frac{1}{D_i}$ a $P_i$ jsou pozice existujících fingerprintů z databáze (Kříž, Malý, Kozel, 2016). Pro účely lokalizace se očekává, že výsledky měření na jednom místě budou vždy podobné. Díky tomu lze stanovit polohu zařízení. Proto jedním z výstupů práce bude porovnání, jak se jednotlivá měření mezi sebou liší.

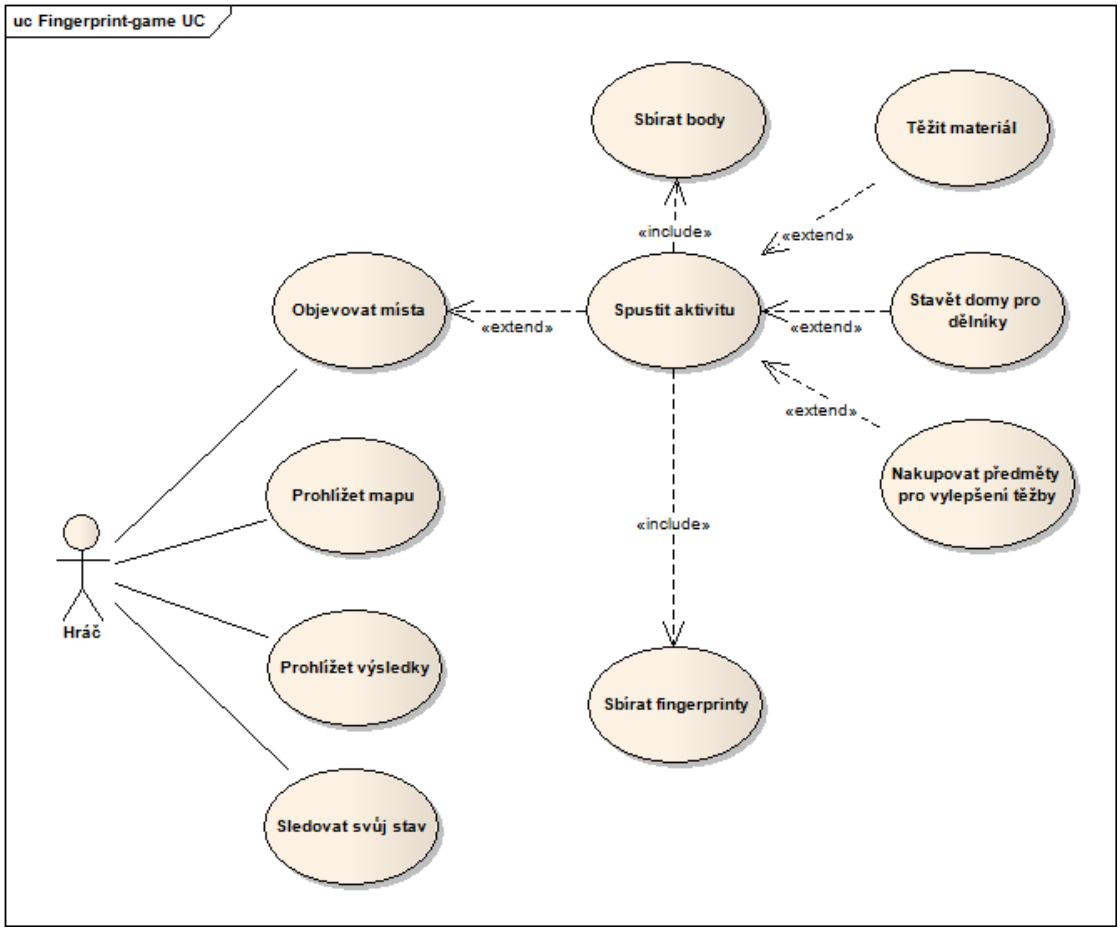
# 3 Analýza

Cílem práce je vytvořit hru, která donutí hráče reálně dojít na nějaké místo a tam provést nějakou aktivitu, při které bude reálně pořízen fingerprint. Na místa ve hře je třeba chodit opakovaně, za účelem sběru většího množství dat. Aby bylo možné zjistit, že se hráč na místě skutečně nachází, je třeba jeho polohu něčím ověřit. Ověření může být formou sejmutí QR kódu, který se na místě nachází, nebo fotografií místa.

Aby hráči byli motivováni ve hře pokračovat, je dobré zakomponovat možnost rozšiřování, tj. vylepšování postavy nebo jiného posouvání ve hře. Na každém patře budovy by mělo být možné hru hrát, proto je potřeba, aby měl uživatel nějakou možnost mezi jednotlivými mapami přecházet. Aby hráči místa ve hře více procházeli, jsou jim na začátku všechna místa skryta a objeví je až sejmutím QR kódu. Aplikace jim zobrazí podíl již objevených míst, takže se jednoduše dozví, jestli mají ještě co objevovat.

Velkou možnost rozšiřování mají hry založené na těžení surovin a nakupování věcí. Cílem takovýchto her může být získání co nejvyššího počtu bodů, vylepšování umístění na žebříčku mezi ostatními hráči, případně zabírání nějakých míst ve hře. Takováto hra může být snadno upravována přidáváním nových prvků, například materiálů, předmětů nebo míst. Hráči by zároveň měli být něčím omezováni. Toho je možné docílit buď kapacitou jednotlivých míst (například omezené množstvím materiálu k těžbě) nebo kapacitou zdrojů (například množství dělníků, kteří mohou vykonávat činnost ve hře). Tímto jsou nuceni chodit na místa opakovaně a činnost opakovat.

Protože sběr informací pro lokalizaci uživatele probíhá několik sekund, je potřeba, aby uživatel danou dobu na místě vydržel. Načítání kódu musí probíhat delší dobu, protože jen tehdy má aplikace informaci o tom, že se uživatel nehýbe. Vzniká zde tedy prostor pro interakci s uživatelem. Vzhledem k tomu, že uživatel zde tráví nějakou dobu, je vhodné, aby byl už při snímání kódu informován o tom, o jaké místo se jedná a případně si navolit parametry pro akci.




*Obrázek 2 Use case diagram*

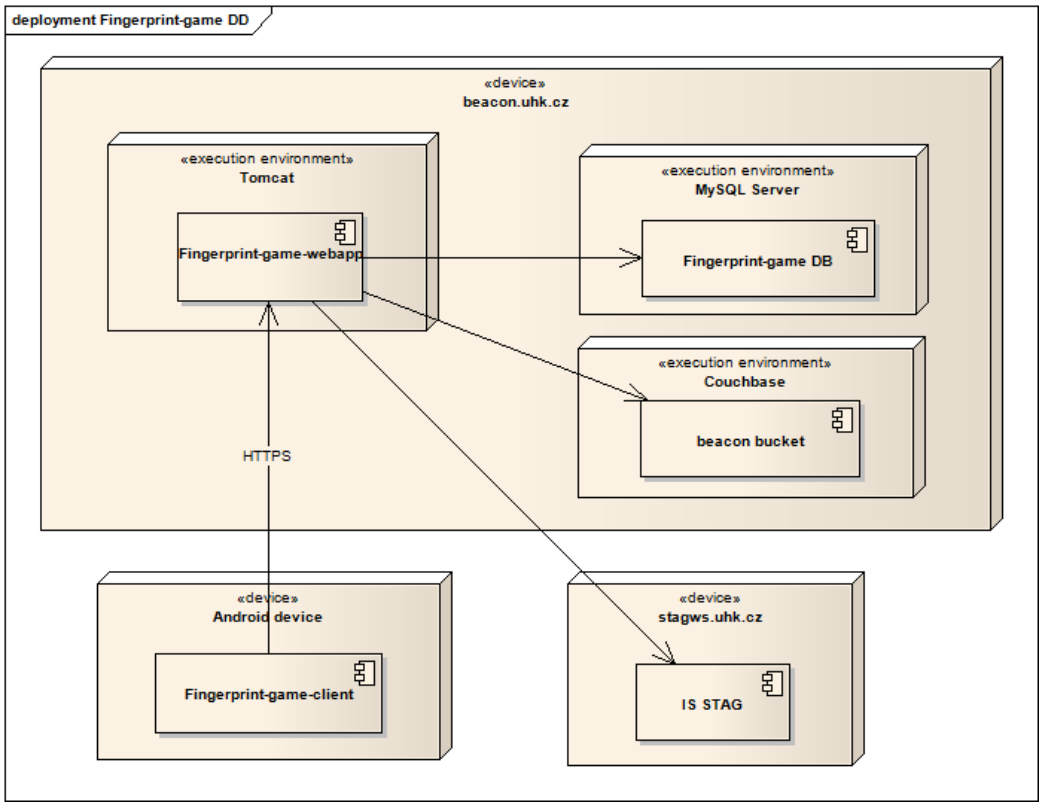
*Zdroj: Vlastní zpracování, 2017*

Obrázek 2 zobrazuje use case diagram hry. Jsou zde vidět činnosti, které má uživatel možnost ve hře provádět. Kombinace těchto činností by měla hráče motivovat k pokračování ve hře a tím i k dalšímu pořizování fingerprintů. Na různých místech bude mít možnost spustit různé typy aktivit. To mu pomůže se ve hře posouvat dál – různé typy materiálů potřebuje k dalším aktivitám.

# 4 Návrh řešení

Vzhledem k tomu, že sběr fingerprintů probíhá pomocí skenu dostupných Bluetooth vysílačů a Wi-Fi sítí, předpokládá se, že uživatelé jsou během používání aplikace připojeni k internetu. Proto může být herní mechanika implementována primárně na serveru. Tam je možné přihlásit uživatele pomocí uživatelských údajů do IS STAG[5] (pro pozdější možnost získání rozvrhu). Na serveru je k dispozici Apache Tomcat[6] a pro ukládání dat aplikace slouží MySQL databáze[7].

Vzhledem k tomu, že si Android stále drží prvenství mezi operačními systémy pro mobilní telefony – v roce 2016 měl stále více než 80% podíl na trhu (Zavřel, 2016), bylo zvoleno, že výsledná aplikace bude psána právě pro tuto platformu. Rozdělení celé aplikace je znázorněno na Obrázku 3.




*Obrázek 3 Deployment diagram*

*Zdroj: vlastní zpracování (2017)*

[5] školní informační systém
[6] http://tomcat.apache.org/
[7] https://www.mysql.com/

## 4.1 Klientská část

Klientská část aplikace komunikuje se serverem pomocí JSONu a všechny požadavky musí být zabezpečeny pomocí HTTPS protokolu z toho důvodu, že uživatelé jsou ve hře přihlášeni a tyto údaje je třeba vždy zasílat na server, aby bylo jasné, o kterého hráče se jedná. Aby bylo možné mít různé verze klientské aplikace a zároveň jen jednu serverovou, je v URL vždy zakomponovaná major i minor verze.

Hra musí obsahovat mapy, ty ale mohou být v průběhu fungování měněny. Bylo by zbytečné kvůli tomu vyzývat uživatele, aby si stahovali novou verzi aplikace. Z tohoto důvodu je dobré obrázky uložit na serveru a v mobilní aplikaci je ukládat do cache paměti zařízení. Kromě map budou tímto způsobem zpracovávány veškeré grafické prvky hry.

Pro příjemnější práci s aplikací je třeba promyslet zobrazování informací uživateli. Aplikace by neměla mít hodně obrazovek, což by ji značně znepřehledňovalo. Zároveň by měla být snadno ovladatelná. Uživatel by měl mít na začátku k dispozici buď mapy, nebo aktuální informace o svém stavu. Vzhledem k budoucí možnosti přesné lokalizace je vhodnější mapa – mohl by vidět, kde se aktuálně ve hře nachází.

Mezi mapami musí být snadné přepínání. Tlačítka na obrazovce by mohla překrývat mapu, což zabírá místo a na mobilních telefonech by pak z mapy byla zobrazena jen malá část. Další možností je zobrazení map vedle sebe. Zde by však bylo složitější přepínání mezi nimi – uživatel musí mít možnost s mapou hýbat, přibližovat a oddalovat ji a zároveň přecházet mezi patry. Uživatelsky nejpřívětivější volbou je tedy rozbalovací menu. Ve výchozím zobrazení bude mapu překrývat pouze jedno tlačítko. To po kliknutí zobrazí další tlačítka pro hru. Nemusí se jednat jen o přepínání pater, ale o veškerou navigaci v aplikaci.

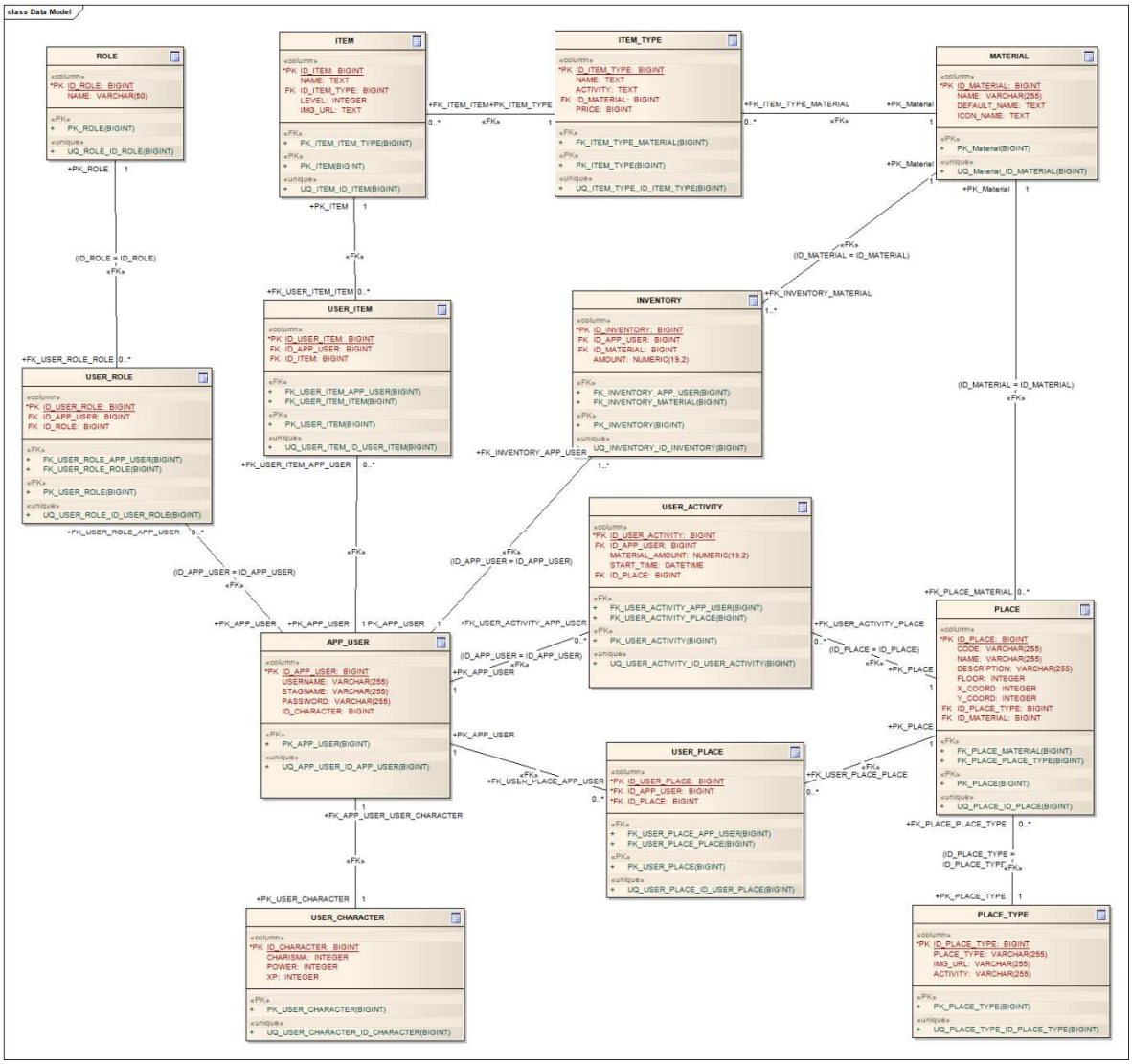
Nový uživatel nebude na začátku znát místa ve hře a bude je muset objevit. Bude mít k dispozici pouze informaci, jaký podíl míst má již objeven. K dispozici bude několik typů míst. První z nich je naleziště, kde může těžit různé typy surovin potřebné k dalšímu fungování ve hře. Jídlo je potřeba pro dělníky, aby mohli těžit. Kamení a dřevo k výstavbě domů pro nové dělníky. Zlatem potom platí těmto najatým dělníkům. Pokud nemá hráč dostatek surovin k pokračování aktivity, tato je okamžitě ukončena.

Pro urychlení těžby si může hráč koupit různé předměty. K dispozici je sekera pro zlepšení těžby dřeva, žebřík pro rychlejší sklizeň a krumpáč pro těžbu kamene. Tyto

předměty lze získat v obchodě a mohou se několikrát vylepšovat. Lepší předmět stojí víc zlata, ale zároveň víc urychlí těžbu dané suroviny.

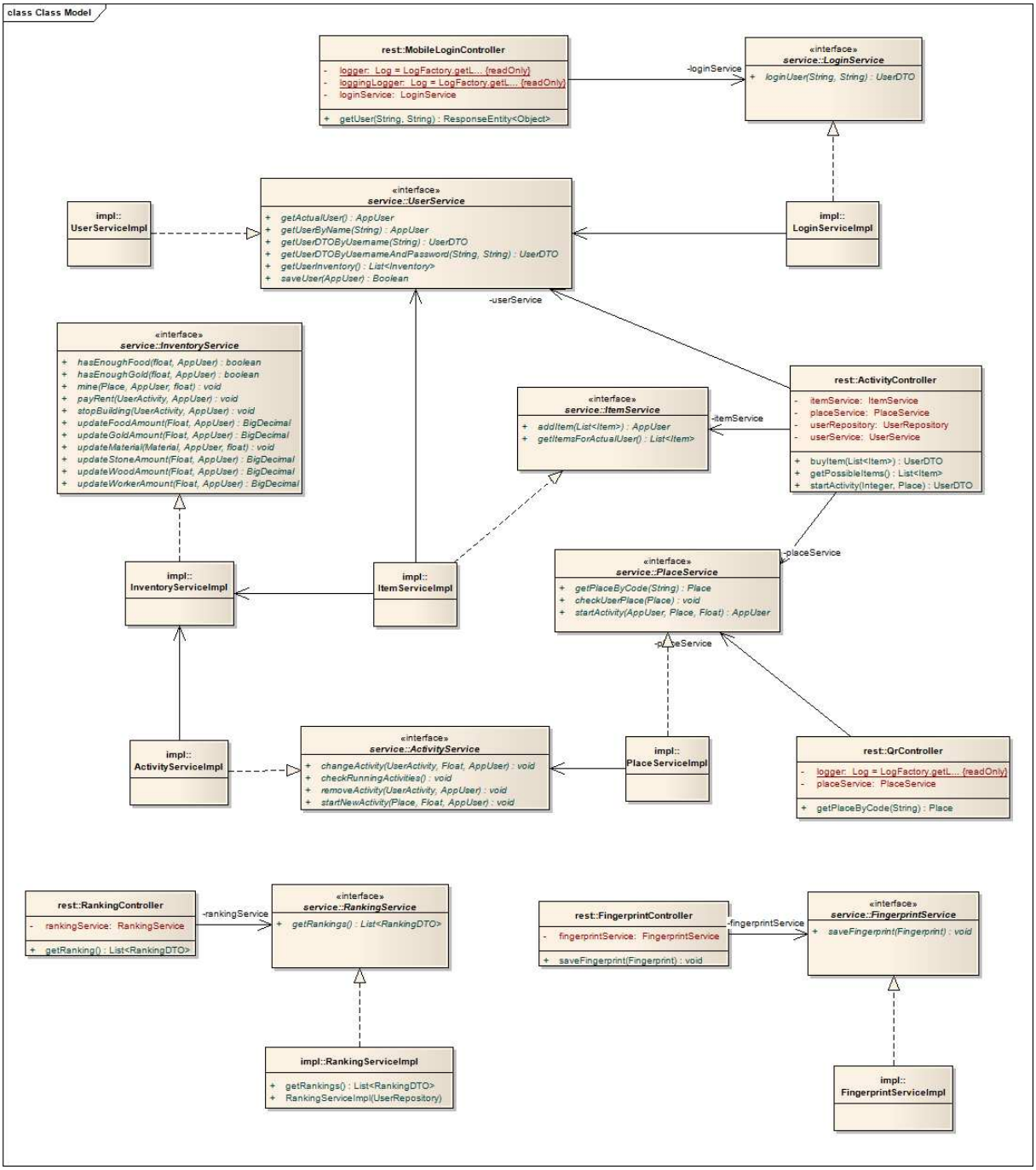
## 4.2 Model

Na základě provedené analýzy byl pro serverovou část aplikace vytvořen databázový model, který je znázorněn diagramem na Obrázku 4. Dále byl vytvořen analytický model tříd na serveru, který je vidět na Obrázku 5. Ten pro větší přehlednost zobrazuje pouze třídy zajišťující hlavní funkcionalitu aplikace a jejich komunikaci, tj. REST controllery přijímající požadavky z klienta a servisní třídy. Modelové třídy v klientské části aplikace by měly s výjimkou drobných změn kopírovat ty serverové, aby byl usnadněn vývoj včetně komunikace mezi nimi.




*Obrázek 4 Databázový model*

*Zdroj: Vlastní zpracování, 2017*




*Obrázek 5 Class diagram*

*Zdroj: Vlastní zpracování, 2017*

# 5 Proces vývoje software a příprava prostředí

Cílem této práce je mimo jiné vytvoření aplikace sestávající ze serverové i klientské části. Z toho důvodu je třeba se zaměřit i na způsob vývoje softwaru. Tyto principy by měly být v průběhu tvorby dodržovány a připraveny pro další možný vývoj aplikace. Kromě čitelnosti kódu a dodržování zásad objektového programování, které by měly být při psaní Java aplikací samozřejmostí, je nedílnou součástí vývoje dodržování zásad Continuous Integration (dále CI).

## 5.1 Continuous integration

Fowler (2006) popisuje continuous integration jako praktiku vývoje, kdy členové týmu pravidelně integrují svoji práci. Každý upravený kód vložený na sdílené uložiště je nejprve ověřen automatickým buildem včetně testů, díky čemuž je množství chyb brzy detekováno. Přestože CI nevyžaduje žádné zvláštní nástroje, je vhodné použít integrační server. Mnoho firem nabízí open source i komerční produkty k těmto účelům.

Termín Continuous Integration je spojován s procesem extrémního programování jako jedna z jeho základních praktik. Předpokládá vysoké procento pokrytí testy – při vývoji je výstupem nejen produkční kód, ale i jeho testy.

Vývoj probíhá tak, že programátor po dokončení úkolu (případně jeho části) spustí lokální build[8]. Pokud kompilace proběhne v pořádku a v testech nejsou odhaleny žádné chyby, je možné změny nahrát do sdílené repository. Je však třeba počítat s tím, že i ostatní pracují na projektu a mohli provést důležité změny. Proto je důležité před každým commitem ze sdíleného uložiště nejprve stáhnout změny kolegů a znovu spustit build i s nimi. Po nahrání kódu do repository je třeba ještě otestovat novou verzi na serveru. To je možné provést ručně, nebo s nástrojem určeným pro CI.

### 5.1.1 Doporučení pro CI

Při CI je dobré dodržovat několik zásad, které umožňují hladký a efektivní vývoj. Čím větší tým na projektu pracuje, tím více by se tyto zásady měly dodržovat. Je však vhodné je zavést i pro jednotlivce nebo malé týmy pro případ pozdějšího růstu projektu

[8] Kompilace kódu, vytvoření spustitelného souboru, spuštění testů

a pro osvojení postupů všemi vývojáři. Tato kapitola byla zpracována na základě článku Continuous Integration od Martina Fowlera (2006).

## Udržování jedné repository

Software se zpravidla skládá z velkého množství souborů, na kterých pracuje mnoho lidí. Tyto soubory je třeba nějakým způsobem hromadně spravovat a udržovat aktuální. Z tohoto důvodu vzniklo množství nástrojů pro správu zdrojových kódů (SCM z anglického Source Code Management nebo VCS podle Version Control System).

Tyto systémy umožňují efektivní spravovat zdrojový kód. S jejich pomocí lze udržovat práci více lidí, snadno dohledat všechny změny apod. Jejich důležitým přínosem je verzování – přiřazení čísla tak, aby bylo zřejmé, o kterou verzi se jedná.

### Verzování

Číslo verze systému se skládá zpravidla ze tří částí – major.minor.patch. Uživatelé systému podle těchto čísel poznají, jak aktuální verzi používají, případně jestli mohou aktualizovat.

Major verze se navyšuje v případě větších změn systému, například nekompatibilní API. Při změně závislosti na vyšší verzi je zpravidla potřeba udělat větší zásah do kódu.

Minor verze se navyšuje s vydáním různých menších změn. Většinou se jedná o novou funkcionalitu a změny jsou zpětně kompatibilní. Závislosti je vhodné udržovat na nejvyšší minor verzi.

Patch verze se navyšuje s vydáním oprav chyb v kódu, proto by závislosti měly vždy obsahovat nejvyšší možnou patch verzi.

Verzování je důležité také při reportu chyb. Uživatelé klientské části aplikace mohou mít nainstalované různé verze aplikace. Proto pokud dojde k chybě, je nutné i vědět, kde přesně a v které verzi k problému došlo. Bez těchto informací se chyby často stávají neopravitelnými, protože aktuální kód už je jiný. Z tohoto důvodu by vždy při vydání nové verze měl být v repository přidán tag a programátor si pak může stáhnout přesně tu verzi, ve které došlo k chybě.

## Automatizace buildu

Při procesu kompilace, přenášení souborů a dalších činnostech spojených s tvorbou systému může snadno dojít k chybě. Z tohoto důvodu existuje zásada, že pokud je možné něco dělat automaticky, vždy tuto možnost využít. Je dobré mít možnost provést build a spuštění aplikace pomocí jediného příkazu.

Mnoho uživatelů pro build využívá nástroje poskytované jejich IDE[9], hlavní branch (master) by však měla vždy obsahovat takový kód, který lze použít na serveru a je spustitelný pomocí připravených skriptů.

## Automatické testy

To, že program běží, neznamená, že funguje správně. Z tohoto důvodu je vhodné zařazovat automatické testy. Ty by měly pokrývat velkou část kódu náchylnou k chybám, například servisní třídy obsahující logiku aplikace.

Testy by měly být snadno spustitelné a snadno kontrolovatelné, tj. jeden test zpravidla kontroluje pouze jednu věc. Pokud test neprojde, z jeho názvu, případně výpisu chyby, musí být jasné, co nefunguje.

## Časté commity

Prvním předpokladem pro commit je, že vývojář má u sebe funkční build včetně procházejících testů. Proto před každým nahráním kódu do sdílené repository je třeba, aby vývojář nahrál do své pracovní kopie změny ostatních kolegů, vyřešil případné konflikty v souborech, spustil testy a spustil aplikaci. Svůj kód může nahrát pouze tehdy, pokud jsou všechny předcházející kroky v pořádku.

Pokud tento cyklus provádějí vývojáři často, například na denní bázi nebo po několika hodinách, mohou rychle odhalit konflikty mezi jednotlivými změnami. Tyto problémy je pak většinou možné jednoduše a rychle vyřešit. Pokud však jsou commitovány velké části kódu po několika týdnech, konflikty mohou být rozsáhle, tzn. složité na vyřešení a někdy dokonce nevyřešitelné.

Časté commity navíc nutí vývojáře rozdělit si práci na menší části. To umožňuje vývojářům sledovat postup vývoje a pomáhá jim k vědomí, že i během pár hodin dokáží vytvořit smysluplnou část programu.

## Integrační server

To, že build prochází u vývojáře, ještě neznamená, že je vše v pořádku. Někomu jinému stejný kód fungovat nemusí, nebo se program může začít chovat jinak na serveru. Problémem mohou být rozdíly mezi prostředími. Z tohoto důvodu by všichni vývojáři měli mít přístup k jednotnému integračnímu serveru a jejich commit se považuje za hotový až ve chvíli, kdy projde build i tam.

[9] Integrated Development Environment – prostředí pro vývoj software

Jestli je vše v pořádku, je možné zjistit dvěma způsoby – manuální buildem nebo integračním serverem. Manuální build spočívá v tom, že vývojář na serveru spustí stejný buildovací skript, jaký používá před commitem. Integrační server se chová jako monitor vzdálené repository. Po každém commitu si automaticky stáhne aktualizovanou verzi aplikace, spustí build a informuje o jeho výsledku (zpravidla formou e-mailu).

## 5.2 Jenkins

Jenkins je open source automatizační server umožňující automatizaci různých druhů úloh, např. buildů, testování, nahrávání na server, spouštění. Může být instalován přes nativní balíčky, Docker nebo jako samostatná služba na prostředí s JVM[10] (Jenkins, 2017). V tomto projektu se používá pro kontrolu a správu serverové části aplikace.

### 5.2.1 Instalace Jenkins

Jenkins je možné na Windows server nainstalovat jako samostatnou službu, nebo nasadit na Tomcat jako aplikaci. Protože na serveru byl již nainstalován Tomcat pro běh jiných aplikací, stačilo vložit soubor .war, stažený z webu Jenkins, do složky webapps[11] Tomcatu. Ten už soubor rozbalí a aplikaci spustí.

Dalším krokem je nastavení adresáře JENKINS_HOME. Ve výchozím nastavení je umístěn ve složce C:/Users/{aktuální uživatel}/.jenkins. Vzhledem k možnosti spolupráce více lidí na projektu a možnosti administrace systému více lidmi je toto umístění značně nepraktické. Hodnotu proměnné je možné nastavit do proměnných prostředí, do CATALINA_OPTS, které se načítají při spuštění Tomcatu, nebo do souboru context.xml, který slouží ke konfiguraci Tomcatu. V projektu je nastavena nová systémová proměná JENKINS_HOME na hodnotu C:\beacon\jenkins, aby byla složka přístupná všem uživatelům s přístupem na server.

Po prvním spuštění Jenkins následuje výběr pluginů. Pro tento projekt byly vybrány pluginy pro e-mailovou notifikaci uživatelů, Maven plugin pro build projektů, GitHub plugin pro komunikaci s repository a plugin pro automatickou zálohu Jenkinse.

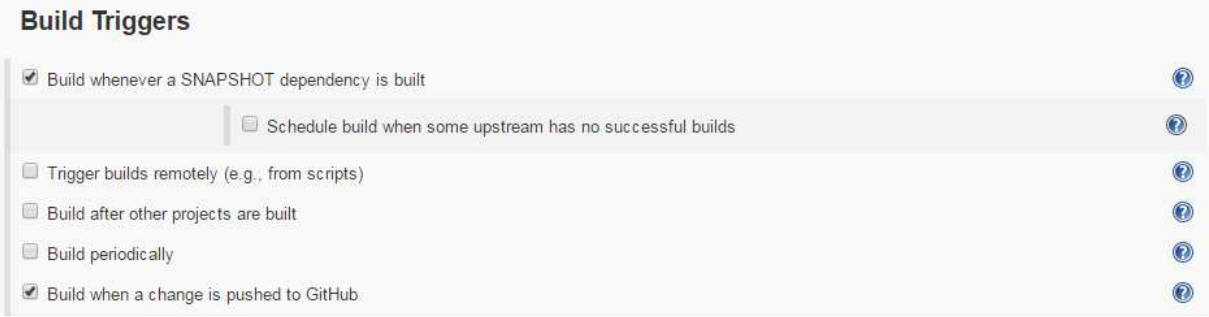
[10] Java Virtual Machine – virtuální stroj používaný pro běh Java programů
[11] Na školním serveru se jedná o složku C:\wampp\tomcat\wabapps

### 5.2.2 Spouštění buildu

U tohoto projektu byl zvolen pro jeho uložení GitHub. Z tohoto důvodu je třeba do Jenkins přidat GitHub plugin. Ten umožňuje stažení kódu ze vzdáleného serveru a jeho následný build. Pro více informací se lze podívat do dokumentace Jenkins (Jenkins, 2017).

U continuous integration je důležité, aby co nejdříve po každé změně proběhly testy. Pokud neprochází, ostatní členové týmu by o tom měli být obeznámeni, aby si poslední verzi projektu nestahovali a neznemožnili si tím práci na projektu.

Plugin také umožňuje spouštění různých událostí. Jednou z nich je automatické spuštění buildu po nahrání změn do repository. Pro zajištění této funkcionality je třeba nejprve povolit build po odeslání změn do repository v nastavení projektu v části Build triggers, jak je vidět na Obrázku 6.




*Obrázek 6 Nastavení GitHub pluginu*

*Zdroj: Vlastní zpracování (2016)*

Druhým krokem je zaregistrovat webhook v nastavení projektu na GitHubu. To zajistí, že po každém commitu do repository odejde z GitHubu zpráva do Jenkinse, kde je nastaveno spuštění buildu po zavolání této URL.

## 5.3 Ztráta dat z Jenkins

Při restartu Tomcatu se stalo, že byl smazán celý obsah adresáře JENKINS_HOME a aplikace se tak chovala jako nově nainstalovaná. Problém nastal po instalaci pluginu pro automatický deploy aplikací, pravděpodobně byl tedy tento plugin špatně nakonfigurován.

Jenkins musel být celý znovu nastaven, tzn. vytvoření uživatelů, stažení a nakonfigurování pluginů. Pro předejití podobným problémům v budoucnu byl navíc

nainstalován plugin pro zálohování dat z Jenkins. První volbou byl Backup Plugin[12], ale není dále udržován a neumožňuje periodické zálohy dat. Z tohoto důvodu byl vybrán plugin thinBackup[13] umožňující jak ručně vyvolanou zálohu všech dat, tak automatické spuštění zálohování souborů.

[12] https://wiki.jenkins-ci.org/display/JENKINS/Backup+Plugin
[13] https://wiki.jenkins-ci.org/display/JENKINS/thinBackup

# 6 Implementace řešení

Pro implementaci navrženého řešení je používán Windows server s MySQL databází a Couchbase DB. Na serveru je nainstalován Apache Tomcat a pro build aplikace Apache Maven.

Serverová část aplikace je napsána v Javě verze 8 s pomocí IDE Eclipse neon. Klientská část aplikace je v Javě 7, jak je zdůvodněno dále, s rozšířením pro Android. Tato část je vyvíjena v Android studiu verze 2.3.

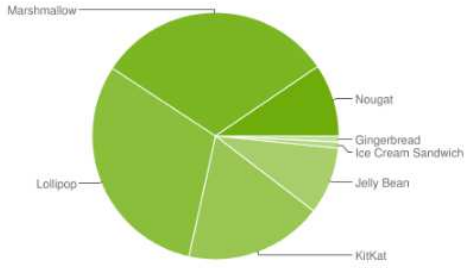
V této části budou pojmy související s vývojem (např. názvy tříd, anotací apod.) označeny *kurzívou*. Pokud se bude jednat o názvy vlastních tříd nebo metod, budou pro odlišení psány fontem `Consolas`.

## 6.1 Android

Android v současné době podporuje Javu 7 a novější verze systému i některé části Javy verze 8. Dle aktuálního podílu jednotlivých verzí systému stále značná část zařízení běží na Androidu v 4.4 (KitKat), jak ukazuje Obrázek 7. Proto musí být hra kompatibilní se zařízeními od této verze. Z tohoto důvodu není stále možné využívat pro klientskou část aplikace výhody Javy 8.

| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 0.8% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 0.8% |
| 4.1.x | Jelly Bean | 16 | 3.1% |
| 4.2.x | | 17 | 4.4% |
| 4.3 | | 18 | 1.3% |
| 4.4 | KitKat | 19 | 18.1% |
| 5.0 | Lollipop | 21 | 8.2% |
| 5.1 | | 22 | 22.6% |
| 6.0 | Marshmallow | 23 | 31.2% |
| 7.0 | Nougat | 24 | 8.9% |
| 7.1 | | 25 | 0.6% |

*Data collected during a 7-day period ending on June 5, 2017.*
*Any versions with less than 0.1% distribution are not shown.*




*Obrázek 7 Podíl používaných verzí Android*

*Zdroj: Android developers, 2017*

### 6.1.1 AndroidAnnotations

Pro zjednodušení a větší přehlednost kódu byl do projektu přidán framework AndroidAnnotations. Díky němu je možné se vyhnout stále se opakujícímu kódu (například při vytváření Activity, práce se SharedPreferences, vytváření listenerů apod.). To umožňuje rychlejší psaní kódu a jeho lepší čitelnost. Na Obrázku 8 je zobrazena ukázka použití anotací v Activity.

```
@EActivity(R.layout.translate) // Sets content view to R.layout.translate
public class TranslateActivity extends Activity {

    @ViewById // Injects R.id.textInput
    EditText textInput;

    @ViewById(R.id.myTextView) // Injects R.id.myTextView
    TextView result;

    @AnimationRes // Injects android.R.anim.fade_in
    Animation fadeIn;

    @Click // When R.id.doTranslate button is clicked
    void doTranslate() {
        translateInBackground(textInput.getText().toString());
    }

    @Background // Executed in a background thread
    void translateInBackground(String textToTranslate) {
        String translatedText = callGoogleTranslate(textToTranslate);
        showResult(translatedText);
    }

    @UiThread // Executed in the ui thread
    void showResult(String translatedText) {
        result.setText(translatedText);
        result.startAnimation(fadeIn);
    }

    // [...]
}
```

*Obrázek 8 Ukázka AndroidAnnotations*

*Zdroj: AndroidAnnotations, 2017*

Výhoda AndroidAnnotations spočívá také v tom, že neobsahují žádnou skrytou logiku, ale namísto toho z anotovaných tříd během kompilace vytváří potomky se stejným názvem zakončené podtržítkem. Pokud si tedy programátor není jistý, co daná anotace dělá, může se podívat do vygenerovaného kódu. Do manifestu se tedy pro správné

fungování aplikace přidávají potomci s podtržítkem. Stejně tak při spouštění nového Intentu se používají generované třídy. Pokud by se vývojář spletl a do manifestu aplikace vložil název původní Activity, během kompilace by na tuto chybu byl upozorněn.

Aby mohly být AndroidAnnotations použity, je třeba nejprve přidat gradle plugin a následně je možné celou třídu oanotovat jednou z následujících anotací (dle typu třídy):

- @EActivity
- @EApplication
- @EBean
- @EFragment
- @EProvider
- @EReceiver
- @EIntentService
- @EService
- @EView
- @EViewGroup

Už jen z názvů anotací lze vyčíst, které se kdy používají, a tedy že je možné je použít v takřka celé aplikaci. V těchto třídách je možné potom injectovat ostatní tímto způsobem vytvořené komponenty.

Dále lze přidávat anotace pro různé typy událostí, což velmi zpřehledňuje kód. Snadno lze také ovládat vlákna pomocí anotací *@UiThread* (pro metody na hlavním vlákne) a *@Background* (pro metody na pozadí). Usnadněná je také práce se *SharedPreferences* a Android resources.

Pravděpodobně největší zjednodušení přináší AndroidAnnotations v komunikaci s REST rozhraním, kde vývojář definuje pouze rozhraní pro jednotlivé metody. Ukázka takto vytvořeného klienta je na Obrázku 9. Bližší vysvětlení komunikace se serverem bude uvedeno v pozdější kapitole této práce.

```java
/**
 * Created by khanusova on 7.9.2016.
 * <p>
 * Rest client for communication with server
 */
@Rest(converters = {MappingJackson2HttpMessageConverter.class}, rootUrl = Constants.URL_BASE,
        interceptors = {AuthInterceptor.class, LogInterceptor.class})
public interface RestClient {

    @Get("/qr/{code}")
    Place getPlaceByCode(@Path String code);

    @Post("/activity/start?materialAmount={materialAmount}")
    AppUser startActivity(@Path Integer materialAmount, @Body Place place);

    @Put("/fingerprint/save")
    void sendFingerprint(@Body Fingerprint fingerprint);

    @Get("/activity/getItems")
    List<Item> getPossibleItems();

    @Post("/activity/buy")
    AppUser buyItem(@Body List<Item> items);

    @Get("/ranking/get")
    List<Ranking> getRankings();
}
```

*Obrázek 9 RestClient*

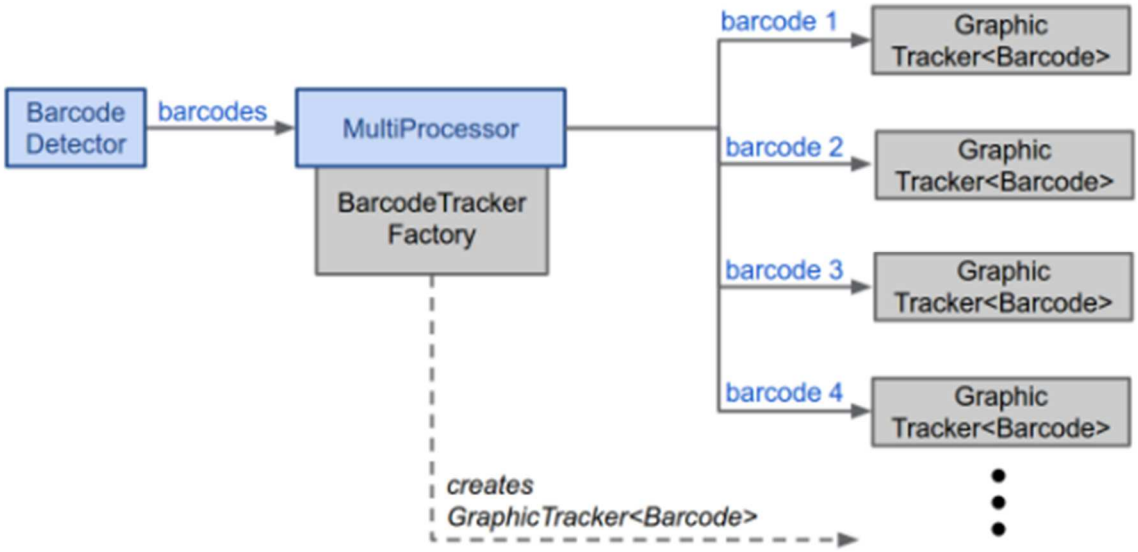
*Zdroj: Vlastní zpracování, 2017*

### 6.1.2 QR kódy

Návštěva jednotlivých míst ve hře probíhá pomocí skenování QR kódů prostřednictvím vytvořené aplikace. Tyto kódy jsou vždy ve tvaru http://beacon.uhk.cz/qr/{kód místa}. Z URL je získán kód místa, na které se aplikace dotáže na serveru. Tam se místo dohledá v tabulce Place podle sloupce code a je vráceno klientovi. Dle typu místa se zobrazí *SeekBar*, neboli vertikální ukazatel umožňující vybrat počet, který umožňuje uživateli nastavit, kolik surovin v místě použije.

Scanování QR kódů probíhá s použitím Google Mobile Vision API[14] starající se o detekci obrazu. Umožňuje rozpoznávat obličeje, čárové kódy a text. Pro načítání kódů je třeba nejprve vytvořit instanci třídy *BarcodeDetector*. Pro rychlejší práci detektoru je možné nastavit podporované formáty kódů (ve výchozím nastavení podporuje všechny formáty). Dalším krokem je vytvoření nové *TrackerFactory*, která umí vytvářet nové instance třídy *Tracker*.

[14] https://developers.google.com/vision/




*Obrázek 10 Průběh skenu pomocí Google Vision API*

*Zdroj: Android developers, 2017*

Obrázek 10 znázorňuje průběh skenování kódu. Je na něm vidět, že detektor vytváří kolekci instancí kódů. *MultiProcessor* sleduje všechny aktivní instance. Pomocí *TrackerFactory* pro každou instanci kódu vytvoří *GraphicTracker* starající se o sledování a grafickou reprezentaci kódu. *MultiProcessor* jim během skenování posílá aktualizace a *Tracker* se postará o aktualizaci.

### 6.1.3 Zobrazování údajů

Protože Android aplikace je to, co uživatelé uvidí, je třeba dobře promyslet, jak se jim budou zobrazovat informace. Aplikace by měla být snadno ovladatelná a pro přehlednost by měla dodržovat pravidla doporučená tvůrci androidu uvedená v dokumentaci Android Developers Design (2017).

Největší problémy s sebou nese zobrazení mapy, kde je potřeba vyřešit čtyři problémy – vykreslení ikon na mapě, možnost zvětšovat/zmenšovat mapu i s ikonami a posouvat ji, zobrazit tlačítka tak, aby nepřekrývala celou mapu a ukládání a zobrazování jednotlivých obrázků.

Protože hlavní aktivita aplikace zobrazuje mapu a ikony, je potřeba promyslet, jak budou tyto obrázky uloženy, získávány a zobrazovány. Možnostmi je uložení v zařízení a uložení na serveru. V případě stahování map ze serveru také přichází možnost vykreslit na nich jednotlivá místa už na serveru a klientovi posílat hotovou mapu. Pokud

by mapa byla uložena na klientovi, její zobrazení by bylo rychlejší a ušetřil by se přenos dat. Problém s takovýmto zobrazováním je však aktuálnost dat – kvůli nové mapě by si uživatelé museli aktualizovat celou aplikaci. Lepší je proto stahovat obrázky ze serveru. K řešení stále zůstává, zda kombinaci mapy a místa vytvářet už na serveru, nebo vykreslovat až na zařízení. Protože však není možné stahovat novou mapu při každém jejím zobrazení, je lepší vytvářet výsledné zobrazení až na klientovi. Navíc je potřeba vyřešit další problém – cachování. Obrázek ze serveru je stažen jako bitmapa. Tvůrci Androdiu v části dokumentace věnované cachování bitmap (2017) přímo popisují, jak s bitmapami pracovat. Rovnou ale doporučují použití knihovny třetí strany, kterou je Glide[15]. Ta umožňuje načíst obrázek z URL či ze souboru, zobrazit ho ve vybraném view a nastavit mu možnosti cachování.

Při prvním spuštění aplikace však stále trvalo, než se mapa stáhla a zobrazila. Proto byly jednotlivé prázdné mapy umístěny do aplikace a zobrazí se jako placeholder před tím, než se stáhne aktuální mapa ze serveru. Načtení mapy ze serveru včetně umístění placeholderu znázorňuje Obrázek 11.

```
String drawableName = "j" + currentFloor + "np";
int drawableId = context.getResources().getIdentifier(drawableName, "drawable", context.getPackageName());
Glide.with(context)
        .load(Constants.IMG_URL_BASE + currentFloor + "NP.jpg")
        .asBitmap()
        .override(MAP_WIDTH, MAP_HEIGHT)
        .diskCacheStrategy(DiskCacheStrategy.ALL)
        .placeholder(drawableId)
        .into(new SimpleTarget<Bitmap>() {
            @Override
            public void onResourceReady(Bitmap resource, GlideAnimation<? super Bitmap> glideAnimation) {
                view.setMap(resource);
                view.updateView();
            }
        });
```

*Obrázek 11 Načtení mapy*

*Zdroj: Vlastní zpracování, 2017*

Dalším problémem k řešení je, jakým způsobem zobrazit mapu a na ní ikony míst, která uživatel navštívil. Místa by se s mapou měla chovat jako jeden obrázek, ale zároveň by měla být samostatně klikatelná pro zobrazení dodatečných informací. Android pro zobrazování více objektů přes sebe poskytuje třídu *LayerDrawable*. Ta vykresluje několik objektů tak, že poslední přidaný je nahoře. Dohromady se pak chovají jako jeden obrázek. Problém však zůstává s umístěním ikon a nastavením jejich velikosti.

[15] http://bumptech.github.io/glide/

*LayerDrawable* sice má metody pro nastavení velikosti ikon (*setLayerSize*, *setLayerWidth*, *setLayerHeight*), ale ty jsou dostupné až od API 23 a úprava se tedy projeví jen na některých zařízeních. Minimální SDK je však nastaveno na 19 a proto je potřeba vymyslet jiné řešení. Nakonec byla zvolena metoda *setLayerInset*, která vrstvě nastaví, o kolik pixelů bude odsazena od každého okraje. Ve výpočtu je tedy odečítána velikost ikony od šířky mapy a dopočítáváno odsazení za pomoci souřadnic místa.

Po zobrazení mapy je potřeba, aby s ní uživatel mohl manipulovat, tj. ji zvětšovat, zmenšovat a posouvat. Android bohužel nenabízí žádné již hotové řešení na obrázky, se kterými by si uživatel mohl takto pohybovat. Je tedy nutné vytvořit vlastní view, které bude implementovat tuto funkcionalitu. Nová třída se jmenuje `TouchImageView` a pro detekci pohybu používá *onTouchListener* a *SimpleOnScaleGestureListener*. Pomocí těchto listenerů detekuje uživatelská gesta a na základě nich vypočítá, o kolik se view má pohnout, případně zvětšit/zmenšit a zavolá metodu *postTranslate* třídy *Matrix*. Ta provede translace zobrazovací matice a upraví zobrazení dle požadavku.

Uživatel se potřebuje přepínat mezi jednotlivými patry, jednoduše spouštět sken a zobrazovat si svůj stav. Protože se jedná o aktivity, které by mu měly být snadno dostupné, mělo by být jejich spouštění umístěno na mapě. Zároveň však nesmí zabírat velkou plochu mapy, aby zůstala přehledná a snadno se s ní manipulovalo. Kombinací těchto požadavků je rozbalovací menu na *FloatinActionButton* (tlačítko pro spuštění akcí umístěné na obrazovce překreslující její obsah). Takovou možnost nabízí *FloatingActionMenu* dostupné z knihovny FloatingActionButton[16] od Clans. Ostatní možnosti, které uživatel nevyužívá tak často, jsou umístěny v horním menu. Zobrazení jednotlivých uživatelských možností je vidět na Obrázku 12. Je na něm vidět i to, že i když uživatel nemůže už přepnout do vyššího patra, pro zachování konzistence je tato volba stále zobrazena, jen je tlačítko zobrazeno šedě, aby bylo vidět, že je nedostupné. Barevně je také odlišeno spouštění nových aktivit od pouhého přepínání mezi patry.

[16] https://github.com/Clans/FloatingActionButton


*Obrázek 12 Rozbalení menu*

*Zdroj: Vlastní zpracování, 2017*

## 6.2 Server

Serverová část aplikace je umístěna na školním serveru beacon.uhk.cz přístupném mj. prostřednictvím RDP[17]. Tam je k dispozici webový kontejner Apache Tomcat[18], na kterém je nasazena webová aplikace s názvem fingerprint-game. Z této aplikace je primárně využíváno vystavené REST rozhraní pro Android klienty, k dispozici je také část aplikace přístupná z internetového prohlížeče. V současnosti se jedná pouze o pravidla ochrany osobních údajů, ale je zde připraven prostor pro další rozšiřování aplikace i na webové rozhraní. Konkrétně aplikace používá Spring MVC[19] v kombinaci se Spring security[20] a Thymeleaf[21].

[17] Remote Desktop Protocol – síťový protokol umožňující ovládání vzdáleného počítače
[18] http://tomcat.apache.org/
[19] https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
[20] https://projects.spring.io/spring-security/
[21] http://www.thymeleaf.org/

Business logika aplikace je definována v samostatných servisních třídách. Tyto jsou tvořeny na rozhraní, aby v případě případných pozdějších změn bylo jednoduché vyměnit jen implementaci a nebylo potřeba zasahovat více do kódu. Pro logiku týkající se úpravy dat jsou používány anotace *@Transactional*, které umožňují spouštět transakce. V případě chyby při vykonávání transakce je proveden rollback a všechny změny jsou smazány a data jsou v původním stavu, jak byly před spuštěním transakce. Vzhledem k tomu, že Spring obaluje servisní třídy vlastní proxy, díky čemuž se anotace aktivuje, je potřeba, aby transakce byly vždy na public metodách a přistupovalo se k nim zvenčí.

### 6.2.1 DB vrstva

Na serveru je k dispozici MySQL server, na kterém je umístěna databáze fingerprint-game k uvedené aplikaci. Zde jsou uložena všechna data aplikace. Další používanou databází je objektová CouchbaseDB[22], do které se ukládají fingerprinty. Vzhledem k tomu, že je v tomto projektu používána jen k ukládání JSON souborů, tato kapitola se jí nebude věnovat a způsob ukládání bude popsán až v kapitole věnované sběru a ukládání fingerprintů.

Aplikace s databází komunikuje pomocí ORM[23] frameworku Hibernate[24] s použitím Spring data JPA[25]. Tato kombinace při práci programátora odstiňuje od starostí s připojováním k databázi a psaním SQL dotazů. Ty jsou vytvářeny pouze pomocí rozhraní rozšiřující *JpaRepository*. V něm jsou definovány metody podle toho, co je potřeba vyhledat. U jednodušších dotazů díky předdefinovaným názvům metod již není potřeba nic dalšího připisovat (např. metoda `public AppUser findByUsername(String username);` vyhledá uživatele dle uživatelského jména). Spousta metod je definovaná již v předkovi rozhraní, a tedy není třeba je znovu vytvářet (např. count, delete, findAll apod.).

Pro úpravu struktury databáze je do aplikace přidán Liquibase[26]. Všechny změny v databázi by měly probíhat přes něj, aby byly zpětně dohledatelné. Do předem připravené složky, konkrétně src/main/resources/liquibase jsou přidány SQL soubory, které se mají spustit. Nástroj si vytvoří v databázi tabulku s changelogy a spouští skripty,

[22] https://www.couchbase.com/
[23] Object Relational Mapping – mapování databázových tabulek a modelových objektů
[24] http://hibernate.org/orm/
[25] http://projects.spring.io/spring-data-jpa/
[26] http://www.liquibase.org/

které ještě nebyly spuštěny v pořadí, v jakém jsou definovány v souboru changelog-master.xml.

## Validace

Pro snadnější přenositelnost aplikace neprobíhá validace na straně databáze, ale na straně aplikace pomocí javax.validations spolu s Hibernate Validatorem[27].

Do pom.xml je potřeba přidat závislost na Hibernate Validator. Ten ke svému chodu potřebuje implementaci EL[28] pro možnost dynamického zpracování chybových hlášek (Redhat, 2017). Byly tedy přidány tyto dvě závislosti:

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>5.3.4.Final</version>
</dependency>
<dependency>
   <groupId>org.glassfish.web</groupId>
   <artifactId>javax.el</artifactId>
   <version>2.2.4</version>
</dependency>
```

Následně je potřeba, aby u všech modelových tříd proběhla validace před jejich uložením do databáze. Nejsou však třeba složitější validace, stačí výchozí. Pro zajištění této funkcionality je do aplikačního kontextu přidána bean validationFactory:

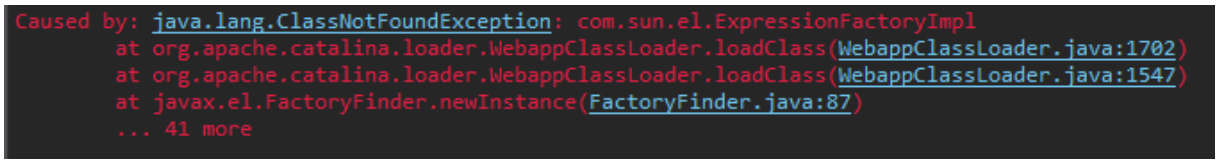
```
<bean id="validationFactory" class="javax.validation.Validation"
          factory-method="buildDefaultValidatorFactory" />
```

Tato konfigurace zajistí, že třídy opatřeny javax.validation anotacemi (např. @NotNull, @Min apod.) nemohou být uloženy do databáze v nevalidním stavu. V případě, že se o to aplikace pokouší, je vyhozena výjimka *ConstraintViolationException* s informacemi o příčině problému.

### Problémy se zavedením validace

Po přidání Hibernate Validatoru spolu s vytvořením bean v aplikačním kontextu nebylo možné spustit aplikaci, protože se nepodařilo vytvořit validationFactory (javax.validation.Validation). Důvodem byla tato chyba, která je vidět na Obrázku 13.

[27] http://hibernate.org/validator/
[28] Expression Language – jazyk umožňující vyjádření hodnoty jako odkazu




*Obrázek 13 Chyba při vytváření bean*

*Zdroj: vlastní zpracování, 2017*

Příčinou chyby byla špatná kompatibilita s javax.el-api verze 2.2.4, což byla první volba knihovny pro zpracování EL. Řešením bylo tuto knihovnu nahradit jinou, konkrétně el-impl ze skupiny org.glassfish.web verze 2.2.

## 6.3 REST

REST byl představen a definován v roce 2000 Royem Fieldingem v jeho disertační práci. Slouží k návrhu distribuovaných systémů. Nejedná se o standard, jako spíš o soubor pravidel (například bezstavovost, vztah klient-server). Principy této komunikace spočívají v následujících bodech:

- zdroj vystavuje srozumitelnou adresářovou strukturu URI
- klient zasílá JSON nebo XML reprezentující zasílané objekty a atributy
- zprávy definují HTTP metody (GET, POST apod.) explicitně
- bezstavová interakce si neukládá klientský kontext na serveru mezi requesty – o stav session se stará klient

Díky těmto pravidlům vždy všechny GET requesty se stejnými parametry vrací stejné výsledky. POST a PUT requesty jsou pak většinou používány pro insert nebo update entit. Rozdíl mezi nimi je ten, že PUT jen ukládájí/updatují objekty bez vedlejšího efektu, tzn. pokud je touto metodou zaslán dvakrát identický objekt, výsledek bude stále stejný. PATCH request lze použít k updatu pouze některých polí objektu a DELETE pro smazání objektu (Pivotal Software Inc., 2017).

### 6.3.1 Vystavení rozhraní

Rozhraní je na serveru vystaveno pomocí knihovny Spring. Třídy používané jako REST controllery jsou opatřeny anotací *@RestController*. Ta spolu s anotacemi

*@RequestMapping* zajistí vystavení metod a tím tvorbu REST rozhraní. Pro mapování objektů z/do JSONu byla použita knihovna Jackson[29].

Základní URL aplikace je *http://beacon.uhk.cz/fingerprint-game*. Pro REST metody Android clienta tato URL potom pokračuje */android/{verze}*, kde je určena verze rozhraní z důvodu zpětné kompatibility s klienty. Pokud tedy bude vydána nová major verze aplikace, kde bude rozhraní změněno, budou vystaveny nové metody s jinou verzí v URL, a tím nebudou všichni uživatelé nuceni si okamžitě updatovat verzi klientské aplikace. Seznam vystavených metod s popisem je uveden v Příloze 1.

### 6.3.2 Klientská část

Na klientovi je komunikace s vystaveným rozhraním implementována kombinací AndroidAnnotations a Spring for Android[30].

Spring for Android umožňuje používat Spring Framework při vývoji na zařízení s Androidem. V současné době se jedná o hlavně o možnost používání RestTemplate. Ta umožňuje vytváření HTTP requestů a přizpůsobuje nativní Android HTTP knihovnu pro vytváření těchto requestů (Clarkson, 2014).

Všechna data odeslaná a přijatá pomocí RestTemplate jsou kódovaná pomocí gzip komprese. Pro mapování z/do JSONu je potřeba využít knihovnu třetí strany. Spring for Android pro tyto účely podporuje Jackson a Gson[31] (Clarkson, 2014).

Pro přidání RestTemplate do Android projektu je třeba přidat do aplikačního build.gradle následující řádek:

```
compile('org.springframework.android:spring-android-rest-
template:${version}')
```

V projektu je použitá aktuální verze – 2.0.0.M3. Pro mapování objektů byl použit converter využívající Jackson ObjectMapper, tj. *MappingJackson2HttpMessagConverter*. Mapování je dále upravováno pomocí Jackson anotací, například *@JsonIgnore*. Hlubšími informacemi o použití RestTemplate se tato práce nezabývá z důvodu použití v kombinaci s AndroidAnnotations.

[29] https://github.com/FasterXML/jackson
[30] http://projects.spring.io/spring-android/
[31] https://github.com/google/gson

### 6.3.3 Spring RestTemplate + AndroidAnnotations

Při použití AndroidAnnotations není třeba psát zdlouhavý kód pro volání externí služby, ale stačí definovat pouze rozhraní a AndroidAnnotations potom na jeho základě potřebný kód vygenerují. Porovnání je možné vidět na http://androidannotations.org/. Ke každému rozhraní lze připojit libovolný počet interceptorů[32] (pre-/post-procesory) například pro logování nebo připojení hlaviček. Vytvořené rozhraní s interceptory je vidět na Obrázku 14.

```
/**
 * Created by khanusova on 7.9.2016.
 * <p>
 * Rest client for communication with server
 */
@Rest(converters = {MappingJackson2HttpMessageConverter.class}, rootUrl = Constants.URL_BASE,
        interceptors = {AuthInterceptor.class, LogInterceptor.class})
public interface RestClient {

    @Get("/qr/{code}")
    Place getPlaceByCode(@Path String code);

    @Post("/activity/start?materialAmount={materialAmount}")
    AppUser startActivity(@Path Integer materialAmount, @Body Place place);

    @Put("/fingerprint/save")
    void sendFingerprint(@Body Fingerprint fingerprint);

    @Get("/activity/getItems")
    List<Item> getPossibleItems();

    @Post("/activity/buy")
    AppUser buyItem(@Body List<Item> items);

    @Get("/ranking/get")
    List<Ranking> getRankings();
}
```

*Obrázek 14 Rest client*

*Zdroj: Vlastní zpracování, 2017*

Použití AndroidAnnotations pro komunikaci se serverem má však jednu nevýhodu – tím, že není nutné definovat samostatné AsyncTasky, může lehce dojít k tomu, že volání bude provedeno na hlavním vlákně. To však v Androidu způsobuje *NetworkOnMainThreadException*. Proto je nutné stále myslet na to, že každé použití třídy `RestClient` musí být v metodách opatřených anotací *@Background*, aby volání probíhalo na pozadí a nezasekla se aplikace.

[32] http://docs.spring.io/spring-android/docs/1.0.x/api/org/springframework/http/client/ClientHttpRequestInterceptor.html

## 6.4 Zabezpečení

Serverová část aplikace je zabezpečená pomocí Spring security[33] a všechny REST požadavky (kromě přihlášení) musí být volány uživatelem s rolí ROLE_USER. Pro budoucí použití je ještě připravena role ROLE_ADMIN.

Vzhledem k tomu, že je aplikace testována v prostředí FIM UHK, probíhá přihlašování přes IS STAG. K tomu byla použita dokumentace webových služeb IS STAG ([2017]). Zde je možné vidět, že vystavené metody se dělí na ty, které jsou autorizované pro různé typy rolí a na metody bez autorizace. U těch autorizovaných jsou kontrolovány uživatelské údaje zaslané v hlavičce jako basic authentication, tj. String ve tvaru uživatelské_jméno:heslo zakódované pomocí Base64. Ověření tedy probíhá tak, že ze serverové aplikace je volána zabezpečená služba systému STAG. Ta vrací buď kód 200 (OK) v případě úspěšného provolání, nebo 401 (unauthorized) v případě neplatných uživatelských údajů. Do budoucna je možné upravit serverovou část aplikace i pro použití vlastních uživatelů.

V případě úspěšného přihlášení je uživatel uložen v databázi aplikace a je mu přidělena role ROLE_USER. Další requesty jsou zabezpečeny pomocí Spring security. Pro ověření zpráv z klienta je vytvořena implementace rozhraní *ClientHttpRequestInterceptor*. Tato třída zachytí všechny requesty, kterým byl daný interceptor přiřazen a přidá k nim hlavičku pro autentizaci na serveru, jak je vidět na Obrázku 15.

[33] http://docs.spring.io/spring-security/site/docs/4.0.3.RELEASE/reference/htmlsingle/

```java
/**
 * Created by khanusova on 10.10.2016.
 * <p>
 * Interceptor for handling security requests <br />
 * Adds authentication information to request header
 */
@EBean(scope = EBean.Scope.Singleton)
public class AuthInterceptor implements ClientHttpRequestInterceptor {
    private static final String TAG = "AuthInterceptor";

    @Pref
    Preferences_ preferences;

    @Override
    public ClientHttpResponse intercept(HttpRequest request, byte[] body, ClientHttpRequestExecution execution) throws IOException {
        HttpHeaders headers = request.getHeaders();
        HttpAuthentication auth = new HttpBasicAuthentication(preferences.username().get(), preferences.password().get());
        headers.setAuthorization(auth);
        return execution.execute(request, body);
    }
}
```

*Obrázek 15 Implementace ClientHttpRequestInterceptor*

*Zdroj: Vlastní zpracování*

### 6.4.1 HTTPS

Jak je vidět v Zásadách pro vývojáře od Google (2017), u všech Android aplikací, které se přihlašují vzdáleně, by mělo být volání zabezpečeno pomocí HTTPS. Dalším důvodem pro použití tohoto protokolu je výše uvedené použití basic authentication.

Certifikát pro HTTPS komunikaci je vygenerován pomocí certifikační autority Let's Encrypt[34]. Protože nástroj je určen pro Linux servery, je v první řadě potřeba stáhnout program třetí strany pro použití Let's Encrypt na Windows server. Ten byl stažen z GitHubu Lone Coder[35]. Otevřením příkazové řádky a zadáním příkazu ve tvaru *letsencrypt.exe --manualhost <domain-name> --webroot <document-root>* jsou vygenerovány certifikáty pro server. Po jejich vytvoření bylo pak nutné přidat cesty k nim do konfiguračních souborů apache a po restartování serveru je možné volat jeho adresy přes https.

## 6.5 Sběr fingerprintů

Jak bylo řečeno v teoretické části, při sběru fingerprintů je třeba získat vektor naměřených hodnot na místě. Pro eliminaci extrémů probíhá měření deset sekund, během kterých je získáno více naměřených hodnot.

Sběr hodnot probíhá prostřednictvím třídy `Scanner` v klientské části aplikace. Ta obsahuje metody pro skenování dostupných Wi-Fi a GSM sítí a BLE zařízení v okolí

[34] https://letsencrypt.org/
[35] https://github.com/Lone-Coder/letsencrypt-win-simple/releases

(pokud zařízení nepodporuje BLE, ze skenu ho vynechá). Pomocí metod `startScan` a `stopScan` spustí/zastaví cyklické skenování sítí. Pro zjištění dostupných Wi-Fi sítí používá *BroadcastReceiver*[36] zachytávající *SCAN_RESULTS_AVAILABLE_ACTION*, který po obdržení výsledků přidá jednotlivé sítě do seznamu. Pro informace o GSM sítích je opakovaně spouštěna metoda *getNeighboringCellInfo* zjišťující informace o dostupných buňkách. Pro sken dostupných BLE zařízení sice existuje metoda *startScan* umožňující jednorázově najít zařízení v okolí a vrátit callback, ale Android neposkytuje žádnou alternativu pro nepřetržitý sken. V dokumentaci Android Developers (2017) pak přímo nabádají vývojáře k tomu, aby nikdy neskenovali zařízení ve smyčkách a vyhledávání nastavili timeout. Neumožňuje opakované vyhledávání zařízení, což je klíčový požadavek pro lokalizaci. Z těchto důvodů bylo třeba vybrat nějakou knihovnu třetí strany, která umožňuje skenovat zařízení několikrát během požadované doby, jak je pro tento projekt potřeba. Je proto vhodné vybrat knihovnu třetí strany, která zapouzdřuje požadovanou funkcionalitu a má již odladěné chyby a problémy s tím související. V tomto projektu byla vybrána knihovna Android Beacon Library[37] a to z důvodu konzistence s již existujícími projekty týkající se indoor lokalizace ve škole.

Po dokončení skenu je vytvořena nová instance třídy `Fingerprint`. Ta obsahuje množství atributů, z nichž některé jsou prázdné a existují pouze pro zachování konzistence s již existujícími záznamy v databázi. Oproti nim byly přidány dva nové parametry – `appName` a `clientVersion` určující, z které aplikace data pocházejí a v jaké verzi klienta byla nasbírána. Důležitými parametry jsou `level`, `x` a `y` udávající očekávanou polohu uživatele. V tomto případě se jedná o souřadnice místa reprezentovaného QR kódem na mapě. Neméně důležité jsou `cellScans`, `bleScans` a `wifiScans` nesoucí informace o načtených sítích. Parametr `createdDate` udává datum vytvoření fingerprintu. Další parametry se týkají informací o zařízení, jako je značka, model, informace o HW, jestli podporuje BLE apod. Poslední parametry jsou určeny pro informace z akcelerometru, gyroskopu a kompasu. Ty v současnosti nejsou používány, ale v budoucnu by mohly informovat o pohybu zařízení během skenu, což by mohlo upřesnit lokalizaci.

[36] https://developer.android.com/reference/android/content/BroadcastReceiver.html
[37] https://github.com/AltBeacon/android-beacon-library/

Pro získání dat do fingerprintu bylo potřeba přidat do android manifestu několik oprávnění, která jsou vidět na Obrázku 16. Hodnoty *BLUETOOTH_ADMIN* a *CHANGE_WIFI_STATE* jsou proto, aby mohla aplikace sama zapnout Bluetooth/Wi-Fi před začátkem skenu a po dokončení je zase vypnout.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

*Obrázek 16 Oprávnění aplikace*

*Zdroj: Vlastní zpracování, 2017*

Fingerprinty zaslané klientem jsou na serveru ukládány do objektové databáze Couchbase[38]. Pro práci s databází byl použit framework Spring data couchbase[39]. Nastavení připojení v aplikačním kontextu je vidět na Obrázku 17. Aplikace z databáze nečte, pouze zapisuje nová data a to tím způsobem, že po každém obdržení fingerprintu z klienta uloží výsledek jeho *toString()* metody do samostatného souboru do složky C:/beacon/fingerprint-game/fingerprints/. Následně ho pomocí Jackson ObjectMapperu převede do JSON Stringu a ten uloží do Couchbase databáze.

```
<couchbase:cluster>
  <couchbase:node>127.0.0.1</couchbase:node>
</couchbase:cluster>
<couchbase:clusterInfo login="${couchbase.login}" password="${couchbase.password}"/>
<couchbase:bucket bucketName="${couchbase.bucket}" bucketPassword=""/>
<couchbase:env />
```

*Obrázek 17 Připojení ke Couchbase DB*

*Zdroj: Vlastní zpracování, 2017*

[38] https://www.couchbase.com/
[39] https://docs.spring.io/spring-data/couchbase/docs/2.2.3.RELEASE/reference/html/

# 7 Testování

Důležitou součástí vývoje aplikace je i její testování. Mezi automatické testy patří jednotkové, integrační a UI testy. Důležité jsou i uživatelské testy. Jak bylo řečeno již dříve, veškerá logika je implementována na serveru. Klientská část aplikace pouze volá server a data z něj stažená prezentuje uživateli. Proto jsou testy umístěny pouze v serverové části aplikace.

## 7.1 Unit testy

Logika aplikace není nijak složitě rozmístěna napříč třídami, víceméně se jedná o CRUD[40] operace spojené s nějakou další operací. Z toho důvodu by byly zbytečné integrační testy a stačily tedy jednotkové (unit) testy. Pro psaní těchto testů byl použit framework JUnit[41].

Testovací třídy jsou umístěny ve složce src/main/test. Anotacemi *@Test* jsou v nich označeny testovací metody. Anotace *@Before* a *@After* pak označují metody, které musí být provedeny před každým testem, resp. po něm. Ve většině případů se jedná o přípravu dat pro testy.

Jednotkové testy by měly testovat vždy jednu metodu jedné třídy. Neměly by komunikovat s databází, ani s reálnými instancemi ostatních tříd, tzn. že se v nich nepoužívá Spring. Všechny objekty, které testovaná třída ke své funkcionalitě potřebuje, jí tedy musí být poskytnuty jinak. Jednou možností by bylo všechno vytvářet ručně, ale vzhledem k provázanosti tříd by to bylo dost pracné a ve výsledku by vlastně bylo potřeba pro každý test vytvářet prázdné instance velkého množství servisních tříd. Pro tyto účely existuje framework Mockito[42] umožňující vytvářet tzv. mocky. Jedná se o simulované objekty napodobující chování reálných objektů pomocí řízení jejich chování a hodnot. K mockovaným objektům lze přistupovat stejně jako k běžným objektům aplikace. Mimo to jim lze nastavit, jak se mají chovat v případě volání různých metod, pomoci metody *when*. Přípravu testu včetně nastavení mocků lze vidět na Obrázku 18.

[40] Create Read Update Delete – základní operace pro úpravu dat
[41] http://junit.org/junit4/
[42] http://site.mockito.org/

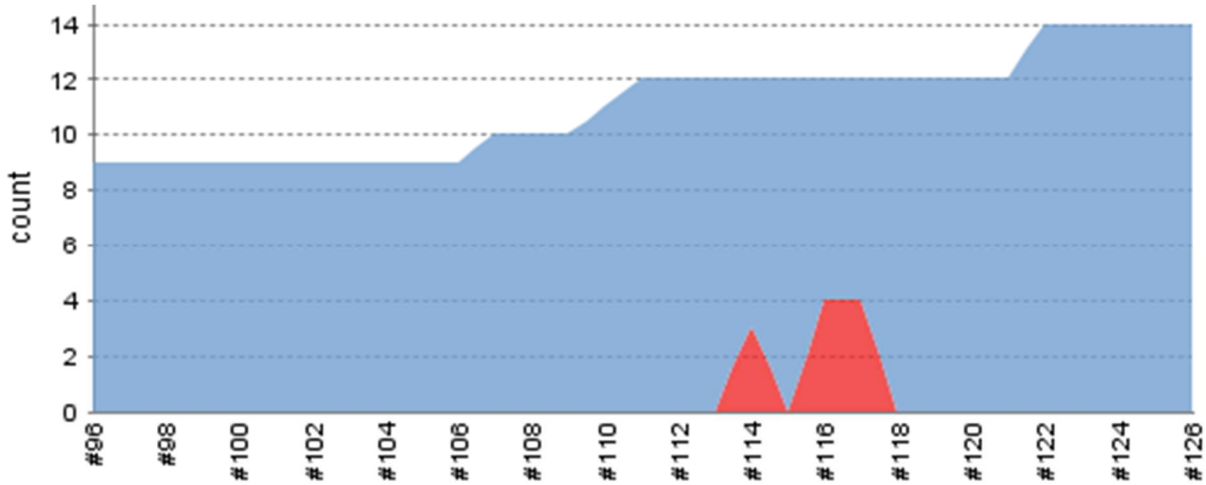
```
@Mock
UserRepository userRepositoryMock;
@Mock
MaterialRepository materialRepositoryMock;
@Mock
PlaceRepository placeRepositoryMock;

AppUser userMock;

@Before
public void init() {
    MockitoAnnotations.initMocks(this);
    userService = new UserServiceImpl(userRepositoryMock, materialRepositoryMock, placeRepositoryMock);
    userMock = userBuilder.build();
    Mockito.when(userRepositoryMock.findByUsername(Mockito.anyString())).thenReturn(userMock);
}
```

*Obrázek 18 Příprava mocků*

*Zdroj: Vlastní zpracování, 2017*

Na tomto obrázku je vidět, že pro jakýkoliv String zaslaný do metody `findByUsername` v `UserRepository` bude v testu vrácena vytvořená instance třídy `AppUser`. Modelové třídy za účelem testů by bylo obtížné konfigurovat pomocí mocků a rychlejší je si je instanciovat samostatně. K tomu byly vytvořeny buildery v balíku `cz.hanusova.fingerprintgame.builder`.

Tyto testy jsou automaticky spouštěny pomocí Jenkins po každém pushnutí do GitHub repository. V případě neúspěšného projití testů je rozeslán e-mail na určené adresy s informací o chybě. Pokud vše proběhne v pořádku, aplikace je nasazena na server. Z toho plyne, že testy by měly hlídat klíčovou funkcionalitu aplikace, aby nebyla na server nasazená nová verze v případě, že některá z důležitých funkcí není v pořádku. Jenkins umožňuje zobrazení průběhu výsledků testů i zpětně, jak je vidět na Obrázku 19. Na ose x je číslo buildu, osa y představuje počet testů. Modrá část grafu znázorňuje úspěšné testy, červeně jsou vyznačeny chybové testy. V případě ignorovaných testů jsou tyto vyznačeny žlutě.




*Obrázek 19 Automaticky spouštěné testy*

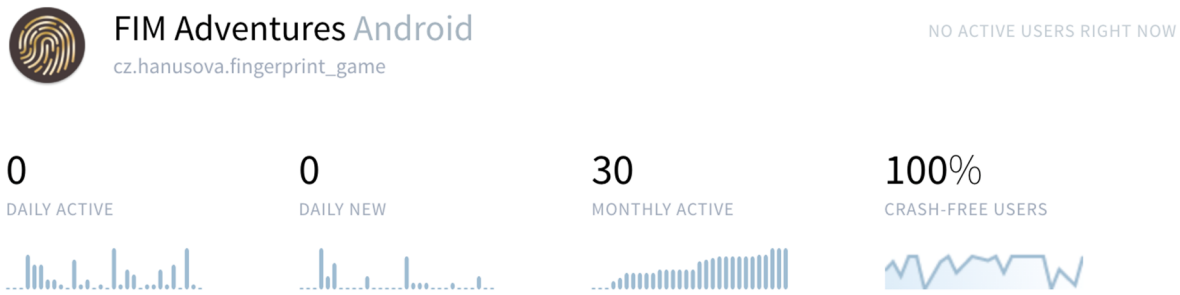
*Zdroj: Vlastní zpracování, 2017*

Výše uvedené však neznamená, že se před prvním spuštěním napíší testy pro klíčovou funkcionalitu a dál už se nové nevytváří. Je dobré také psát test pro každý bug, který se objeví v průběhu používání aplikace. Vývojář by neměl počítat s tím, že na první pokus napíše aplikaci, která se bude uživatelům líbit a nebude obsahovat žádné chyby. Vhodné je vytvořit projekt v nějakém ticketovacím systému (např. JIRA, Redmine), kam se sepisují chyby, připomínky a možná vylepšení. Na serverové části aplikace lze chyby vyčíst z logů. Pokud je vhodně nastavené logování, je z nich možné i vyčíst, za jakých podmínek k chybě došlo.

## 7.2 Fabric

U mobilní aplikace je odhalení chyb složitější – existuje více uživatelů a každý má jiné zařízení. Pokud se nejedná přímo o testera, vývojář se o chybě nemusí ani dozvědět. Nakonec ani nemusí vědět, na kolika a jakých zařízeních jeho aplikace běží, jak ji uživatelé využívají a podobně. K těmto účelům byl přidán Fabric for Android[43]. Ten přehledně zobrazuje aktuální stav aplikace – kolik má uživatelů, jak jsou aktivní, jaké verze používají apod. Základní přehled aplikace je vidět na Obrázku 20.

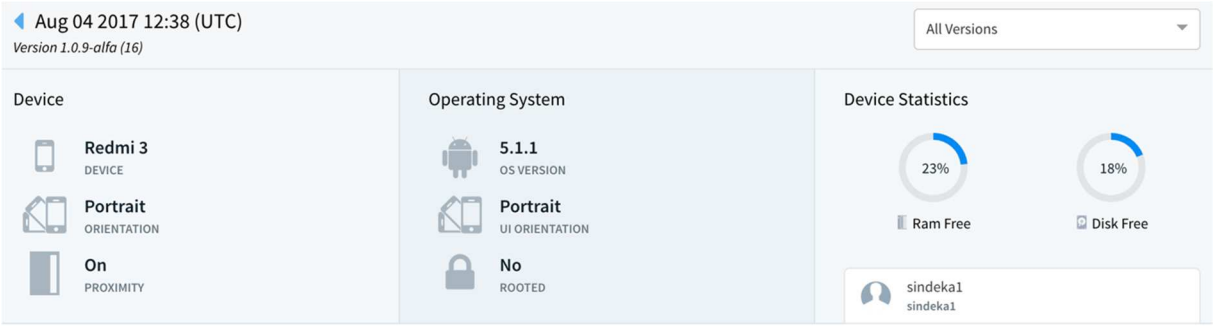
[43] https://docs.fabric.io/android/fabric/overview.html




*Obrázek 20 Souhrnný přehled ve Fabric*

*Zdroj: Vlastní zpracování, 2017*

Důležitou součástí Fabric je nástroj Crashlytics[44]. Ten přehledně zobrazuje chyby aplikace. Dělí je na *Crashes*, které způsobí pád celé aplikace a *Non-fatals*, což jsou chyby zachycené v aplikaci a zaslané pomocí metody *Crashlytics.logException(Throwable t)*. Chyby lze v přehledu filtrovat podle data nebo verze aplikace. V detailu chyby pak kromě konkrétního výpisu výjimky je možné vidět detaily týkající se zařízení a jeho stavu ve chvíli, kdy k chybě došlo. Tyto detaily ukazuje Obrázek 21. Na něm je vidět, že k chybě došlo na zařízení Redmi 3 s Androidem 5.1.1, které v době výskytu bylo orientováno na výšku, mělo 23 % volné RAM a 18 % volného disku. Dalšími údaji jsou uživatelské jméno, verze aplikace a čas chyby.




*Obrázek 21 Detaily zařízení v Crashlytics*

*Zdroj: Vlastní zpracování, 2017*

Fabric je také možné propojit s dalšími aplikacemi, např. Bitbucket, Asana, GitHub, Jira, Redmine, Slack apod. Díky tomu jsou vývojáři rychle informování o chybách v aplikaci, případně mají rovnou založené nové tickety v ticketovacím systému.

[44] https://fabric.io/kits/android/crashlytics

# 8 Zpracování výsledků

Pomocí nové aplikace bylo provedeno celkem 396 měření na jedenácti místech. Měření probíhalo v budově FIM UHK, kde je pro účely výzkumu umístěno množství beaconů. Do vytvořené hry byla zapojena pouze první tři patra budovy, protože ve čtvrtém BLE vysílače umístěny nejsou. Rozmístění BLE beaconů a míst ve hře je znázorněno na mapách v Příloze 2. Beacony jsou vyznačeny modře a označeny čísly, místa s QR kódy jsou zelená, označená písmeny. Tabulka 1 zobrazuje počty měření na jednotlivých místech. Dále je v ní vidět v kolika měřeních se na místě podařilo získat data s různými kombinacemi typů vysílačů. Na všech místech se podařilo naměřit data alespoň z jednoho typu vysílače (GSM, Wi-Fi, BLE). Maximálně u padesáti procent se však podařilo posbírat data ze všech typů vysílačů zároveň. V případě GSM sítě byl modus počtu naměřených hodnot vždy 0, jak je vidět v Tabulce 2. Medián s výjimkou tří míst také. Z tohoto důvodu byla dále sledována data pouze z Wi-Fi a BLE.

Na všech místech s výjimkou jednoho byla alespoň v 70 % měření získána data z obou typů těchto vysílačů zároveň. Na šesti místech se dokonce jednalo o více než 90 % měření.

| Místo | Počet měření | | Počet měření dle typu signálu | | |
|---|---|---|---|---|---|
| | | | BLE + Wi-Fi + GSM | BLE + Wi-Fi | Alespoň jedna z technologií |
| K | 36 | Celkem | 8 | 29 | 36 |
| | | Podíl v % | 22,22% | 80,56% | 100,00% |
| J | 18 | Celkem | 3 | 17 | 18 |
| | | Podíl v % | 16,67% | 94,44% | 100,00% |
| I | 17 | Celkem | 7 | 14 | 17 |
| | | Podíl v % | 41,18% | 82,35% | 100,00% |
| H | 37 | Celkem | 15 | 33 | 37 |
| | | Podíl v % | 40,54% | 89,19% | 100,00% |
| G | 45 | Celkem | 9 | 17 | 45 |
| | | Podíl v % | 20,00% | 37,78% | 100,00% |
| F | 31 | Celkem | 10 | 30 | 31 |
| | | Podíl v % | 32,26% | 96,77% | 100,00% |
| E | 42 | Celkem | 21 | 42 | 42 |
| | | Podíl v % | 50,00% | 100,00% | 100,00% |
| D | 39 | Celkem | 13 | 38 | 39 |
| | | Podíl v % | 33,33% | 97,44% | 100,00% |
| C | 42 | Celkem | 16 | 31 | 42 |
| | | Podíl v % | 38,10% | 73,81% | 100,00% |
| B | 48 | Celkem | 19 | 44 | 48 |
| | | Podíl v % | 39,58% | 91,67% | 100,00% |
| A | 41 | Celkem | 17 | 37 | 41 |
| | | Podíl v % | 41,46% | 90,24% | 100,00% |
| **Celkem** | **396** | | **138** | **332** | **396** |
| | | | **34,85%** | **83,84%** | **100,00%** |

*Tabulka 1 Souhrn sesbíraných dat*

*Zdroj: Vlastní zpracování, 2017*

Tabulka 2 zobrazuje modus a medián počtu měření, kdy se podařilo zachytit jednotlivé typy vysílačů na každém místě. Zejména kvůli GSM, kde modus i medián často nabývaly hodnoty 0, byly tyto ukazatele vypočítány ještě jednou s použitím pouze nenulových hodnot. Z nich je vidět, že u Wi-Fi a BLE se hodnoty moc neliší, ale u GSM jsou čísla dost odlišná, což znamená, že v mnoha případech se nepodařilo vůbec získat data z GSM sítě.

| | | | GSM | | | | BLE | | | | Wi-Fi | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Místo | Počet měření | | Modus | Modus bez nul | Medián | Medián bez nul | Modus | Modus bez nul | Medián | Medián bez nul | Modus | Modus bez nul | Medián | Medián bez nul |
| K | 36 | Celkem | 0 | 22,7 | 0 | 19 | 0 | 1 | 50 | 60 | 21 | 21 | 19,5 | 20 |
| | | Unikátní vysílače | 0 | 11 | 0 | 9 | 3 | 3 | 2 | 3 | 11 | 11 | 12 | 12 |
| J | 18 | Celkem | 0 | 30,8 | 0 | 30 | 0 | 6 | 86 | 89 | 42 | 42 | 41 | 41 |
| | | Unikátní vysílače | 0 | 11 | 0 | 10,5 | 4 | 4 | 3 | 3 | 8 | 8 | 8 | 8 |
| I | 17 | Celkem | 0 | 29,6 | 0 | 33 | 0 | 23 | 23 | 23 | 12 | 12 | 19 | 19 |
| | | Unikátní vysílače | 0 | 11 | 0 | 11,5 | 1 | 1 | 1 | 1 | 12 | 12 | 12 | 12 |
| H | 37 | Celkem | 0 | 27,5 | 5 | 29 | 0 | 67 | 74 | 79 | 5 | 5 | 7 | 7 |
| | | Unikátní vysílače | 0 | 9 | 3 | 9 | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 |
| G | 45 | Celkem | 0 | 16,8 | 12 | 16 | 0 | 1 | 0 | 4 | 27 | 27 | 27 | 27 |
| | | Unikátní vysílače | 0 | 5 | 4 | 7 | 0 | 1 | 0 | 1 | 16 | 16 | 14 | 14 |
| F | 31 | Celkem | 0 | 29,1 | 0 | 28 | 1 | 1 | 4 | 4 | 70 | 70 | 28 | 28,5 |
| | | Unikátní vysílače | 0 | 9 | 0 | 10 | 1 | 1 | 1 | 1 | 10 | 10 | 13 | 13 |
| E | 42 | Celkem | 0 | 22,5 | 6 | 19 | 4 | 4 | 6 | 6 | 10 | 10 | 19 | 19 |
| | | Unikátní vysílače | 0 | 8 | 3 | 9 | 2 | 2 | 2 | 2 | 11 | 11 | 11 | 11 |
| D | 39 | Celkem | 0 | 19,5 | 0 | 18 | 3 | 3 | 4 | 4 | 49 | 49 | 25 | 25 |
| | | Unikátní vysílače | 0 | 7 | 0 | 7,5 | 2 | 2 | 2 | 2 | 8 | 8 | 11 | 11 |
| C | 42 | Celkem | 0 | 12,4 | 0 | 11 | 0 | 4 | 4 | 5 | 40 | 40 | 40,5 | 40,5 |
| | | Unikátní vysílače | 0 | 8 | 0 | 7 | 3 | 3 | 2 | 3 | 25 | 25 | 20 | 20 |
| B | 48 | Celkem | 0 | 12 | 0 | 11 | 3 | 3 | 4 | 4 | 18 | 18 | 37 | 37 |
| | | Unikátní vysílače | 0 | 5 | 0 | 5 | 2 | 2 | 2 | 2 | 19 | 19 | 19 | 19 |
| A | 41 | Celkem | 0 | 21,3 | 0 | 16 | 0 | 1 | 5 | 6 | 32 | 32 | 33 | 33 |
| | | Unikátní vysílače | 0 | 8 | 0 | 10 | 2 | 2 | 2 | 2 | 17 | 17 | 15 | 15 |

*Tabulka 2 Modus a medián počtu měření dle míst*

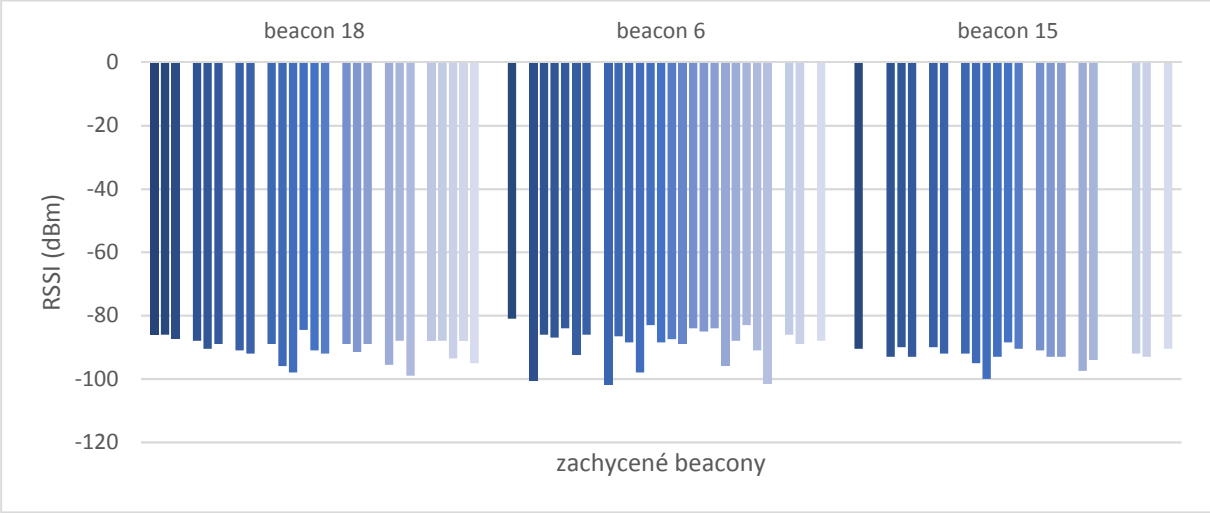
*Zdroj: Vlastní zpracování, 2017*

## 8.1 Výsledky z místa C

Z naměřených dat byly sledovány naměřené hodnoty RSSI a doba, za kterou je možné načíst vysílače. Pro sledování těchto hodnot bylo vybráno místo C, kde je největší medián nalezených unikátních Wi-Fi vysílačů během jednoho měření.

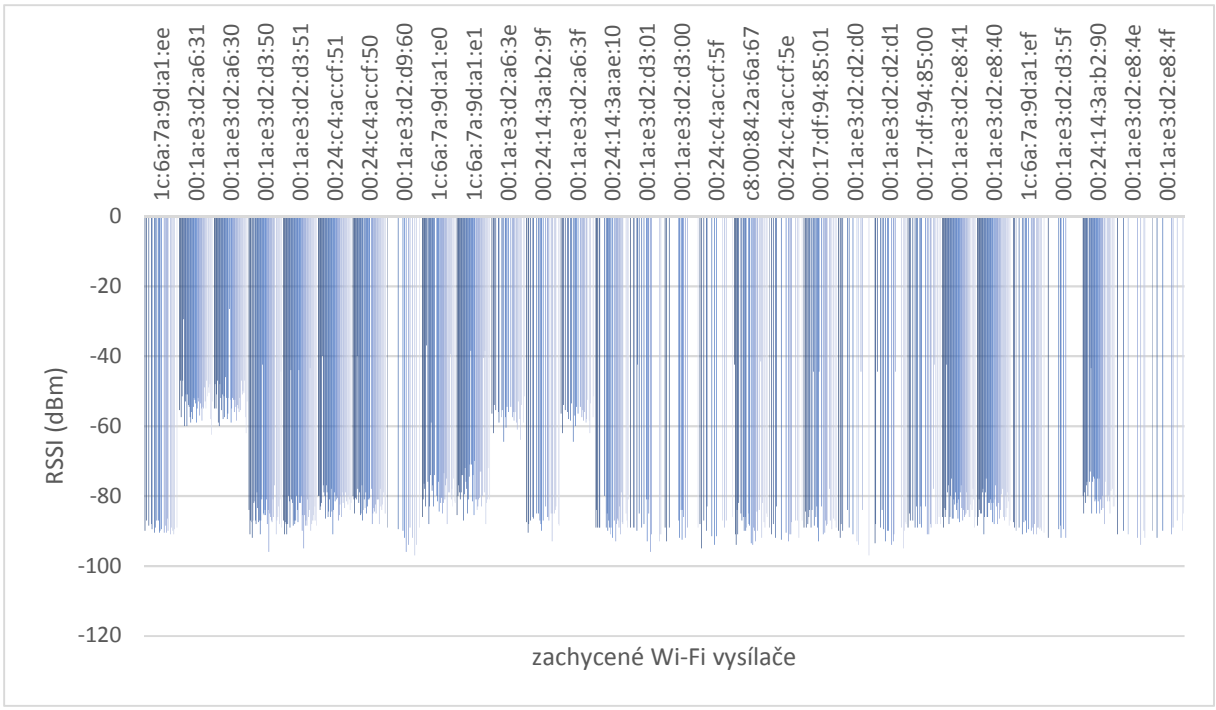
### 8.1.1 RSSI

Pro porovnání hodnot RSSI byly nejprve zjištěny všechny vysílače, ze kterých byl v daném místě získán signál. Pro každé měření byla pak pro eliminaci extrémů vypočítán medián naměřených hodnot z každého zařízení. Z výsledků byly odebrány vysílače, které se podařilo nasnímat pouze v pěti nebo méně měřeních, protože nemají dostatečně vypovídající hodnotu. Seznam všech zachycených beaconů a Wi-Fi včetně vypočítaných mediánů RSSI je vidět v Příloze 3. Důležité je pozorovat, jak se od sebe hodnoty liší mezi jednotlivými měřeními, což přehledně zobrazují následující sloupcové grafy. V ideálním případě, pokud by bylo měření zcela přesné, by všechny naměřené hodnoty pro jeden vysílač byly stejné.




*Graf 1 Měření BLE RSSI na místě C*

*Zdroj: Vlastní zpracování, 2017*



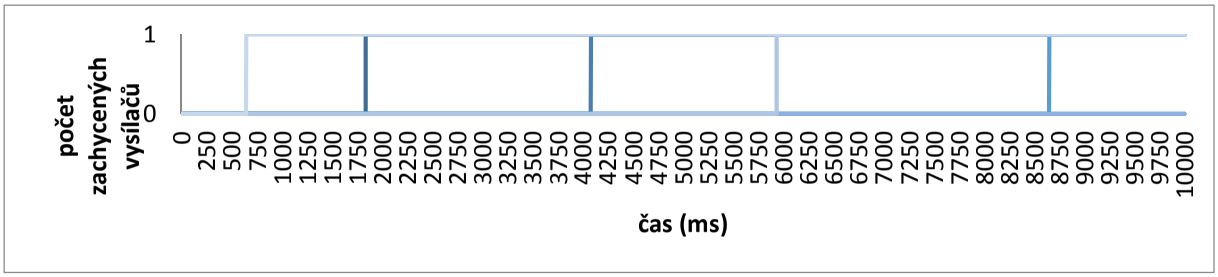

*Graf 2 Měření Wi-Fi RSSI na místě C*

*Zdroj: Vlastní zpracování, 2017*

Grafy 1 a 2 zobrazují zachycené RSSI z jednotlivých BLE a Wi-Fi vysílačů na místě. V rámci jednoho vysílače každý sloupec znázorňuje medián naměřených signálů z jednoho zařízení během jednoho měření. Pokud sloupec chybí, znamená to, že zařízením nebyl signál zachycen vůbec. Z Grafu 1 je zřejmé, že v případě BLE vykazuje měření jen drobné rozdíly. Pokud se podařilo daný vysílač zachytit, rozdíly v hodnotách nebyly velké. Zajímavé na tomto grafu je i to, že u všech vysílačů byly naměřeny přibližně stejné hodnoty RSSI, přestože beacon 6 je k místu C mnohem blíž než beacon 15. Graf 2 zobrazuje, že u Wi-Fi vysílačů však byly výkyvy v rámci každého vysílače větší.

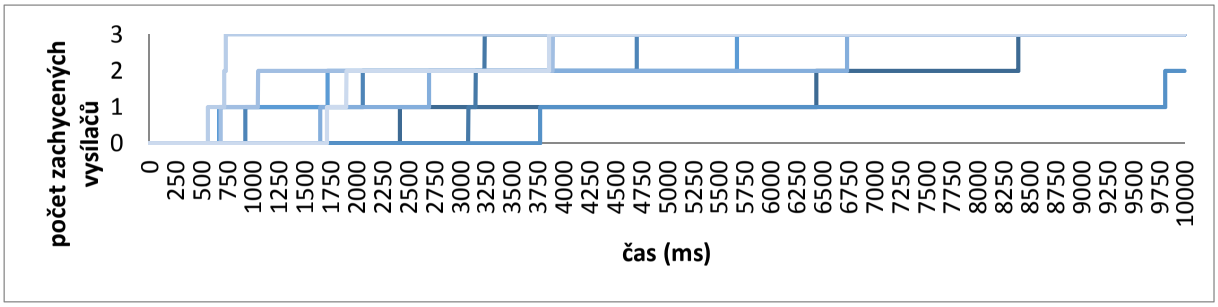
### 8.1.2 Čas

Dalším sledovaným údajem byl čas, za který jsou jednotlivá zařízení schopna přijmout signál od okolních vysílačů. K tomu byla vybrána zařízení, která na místě provedla alespoň pět měření. Na základě zkušeností z předchozích výzkumů probíhalo měření vždy deset sekund. Pro každé zařízení byl sledován počet Wi-Fi a BLE vysílačů, který se mu v tomto čase podařilo na místě najít. Grafy 3 – 10 tedy graficky znázorňují průběh několika měření jedním zařízením na jednom místě a jak v průběhu času narůstá počet zachycených vysílačů.



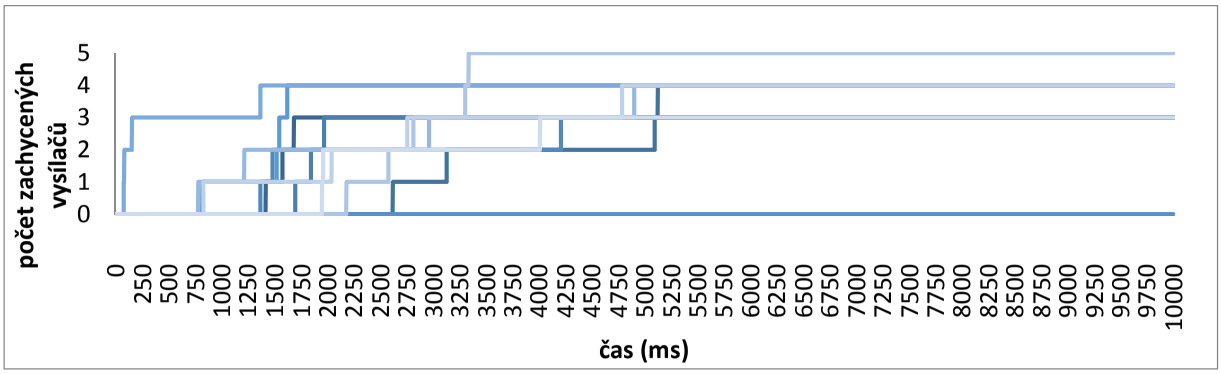

*Graf 3 Zachychení BLE vysílačů v čase – Asus Z00D, místo C*

*Zdroj: Vlastní zpracování, 2017*



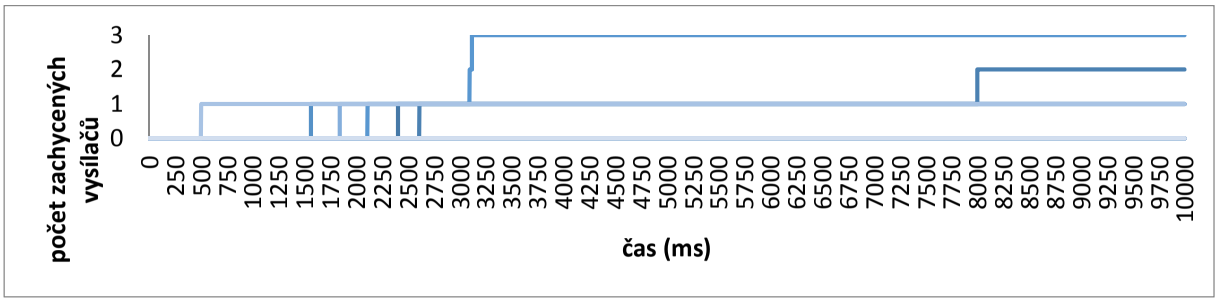

*Graf 4 Zachycení BLE vysílačů v čase – Xiaomi Redmi 3, místo C*

*Zdroj: Vlastní zpracování, 2017*



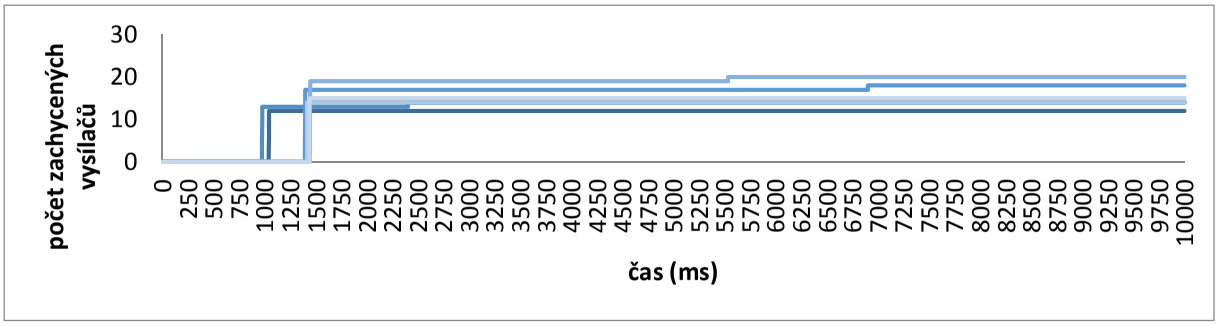

*Graf 5 Zachycení BLE vysílačů v čase – Sony F8331, místo C*

*Zdroj: Vlastní zpracování, 2017*



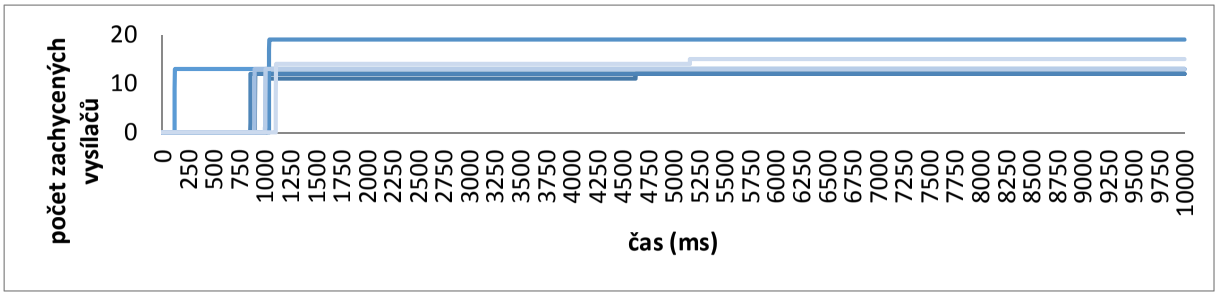

*Graf 6 Zachycení BLE vysílačů v čase – Lenovo YB1-X90L, místo C*

*Zdroj: Vlastní zpracování, 2017*



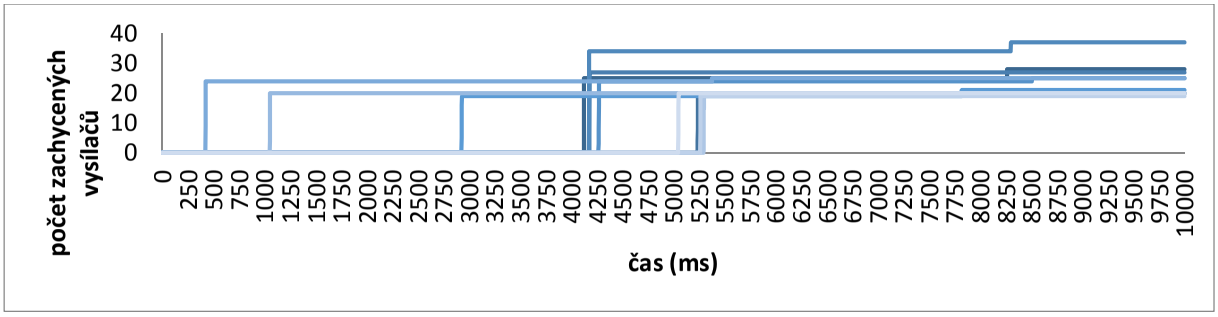

*Graf 7 Zachycení Wi-Fi vysílačů v čase – Asus Z00D, místo C*

*Zdroj: Vlastní zpracování, 2017*



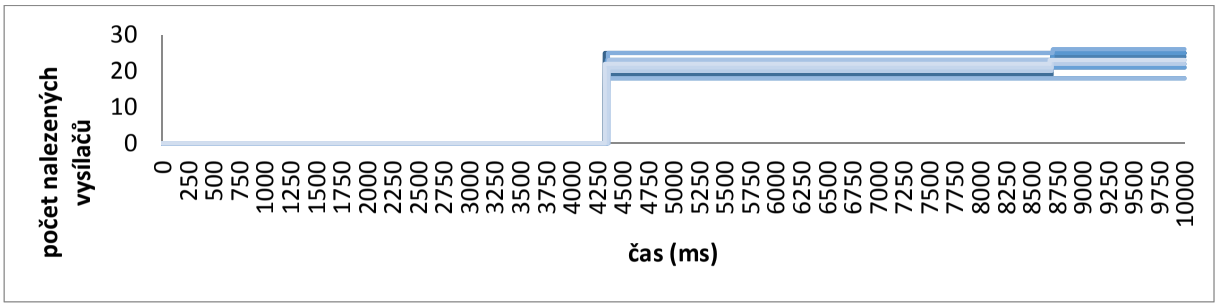

*Graf 8 Zachycení Wi-Fi vysílačů v čase – Xiaomi Redmi 3, místo C*

*Zdroj: Vlastní zpracování, 2017*




*Graf 9 Zachycení Wi-Fi vysílačů v čase – Sony F8331, místo C*

*Zdroj: Vlastní zpracování, 2017*




*Graf 10 Zachycení Wi-Fi vysílačů v čase – Lenovo YB1-X90L, místo C*

*Zdroj: Vlastní zpracování, 2017*

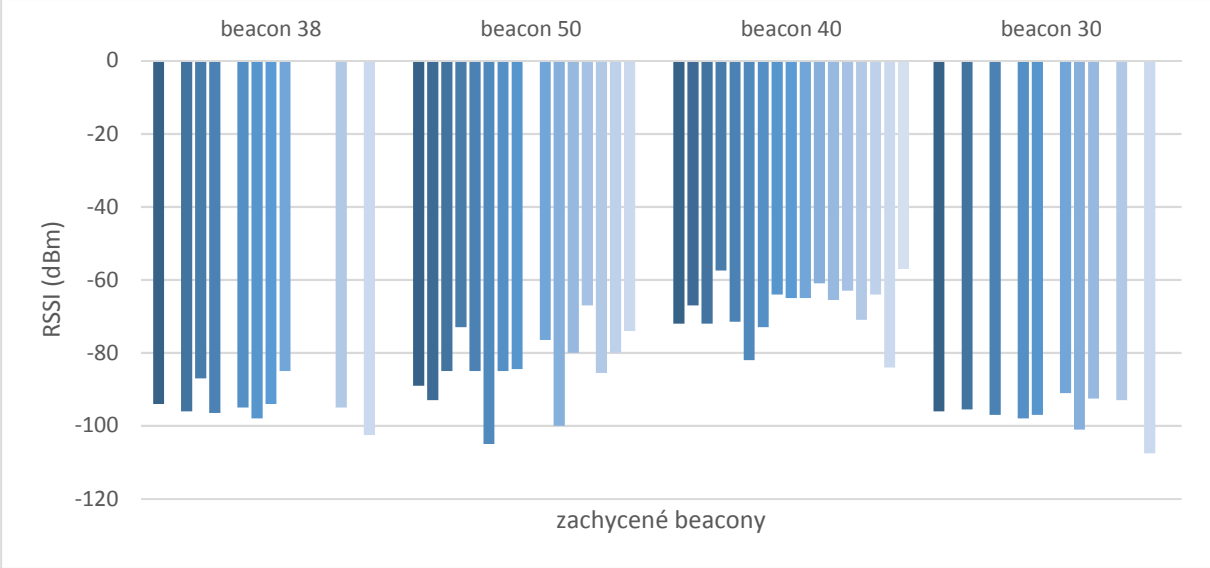
Z grafů je zřejmé, že čas nasnímání vysílačů se liší v závislosti na zařízení, a to co do počtu nalezených vysílačů, tak do času, který jim trvalo tyto vysílače nalézt. U Wi-Fi se podařilo najít většinu zařízení do šesti vteřin u BLE se to mezi zařízeními hodně lišilo. Stále však největší množství vysílačů se podařilo najít během první poloviny časového limitu.

## 8.2 Další místa

Pro potvrzení výsledků měření byla data graficky vyhodnocena ještě na dalších místech. To bylo vybráno ve třetím patře na místě J, protože je zde nejvyšší medián počtu naměřených hodnot BLE i Wi-Fi získaných během jednoho měření. Dále bylo vybráno místo E z důvodu nejvyššího podílu měření, kdy se podařilo získat data z Wi-Fi i BLE zároveň.

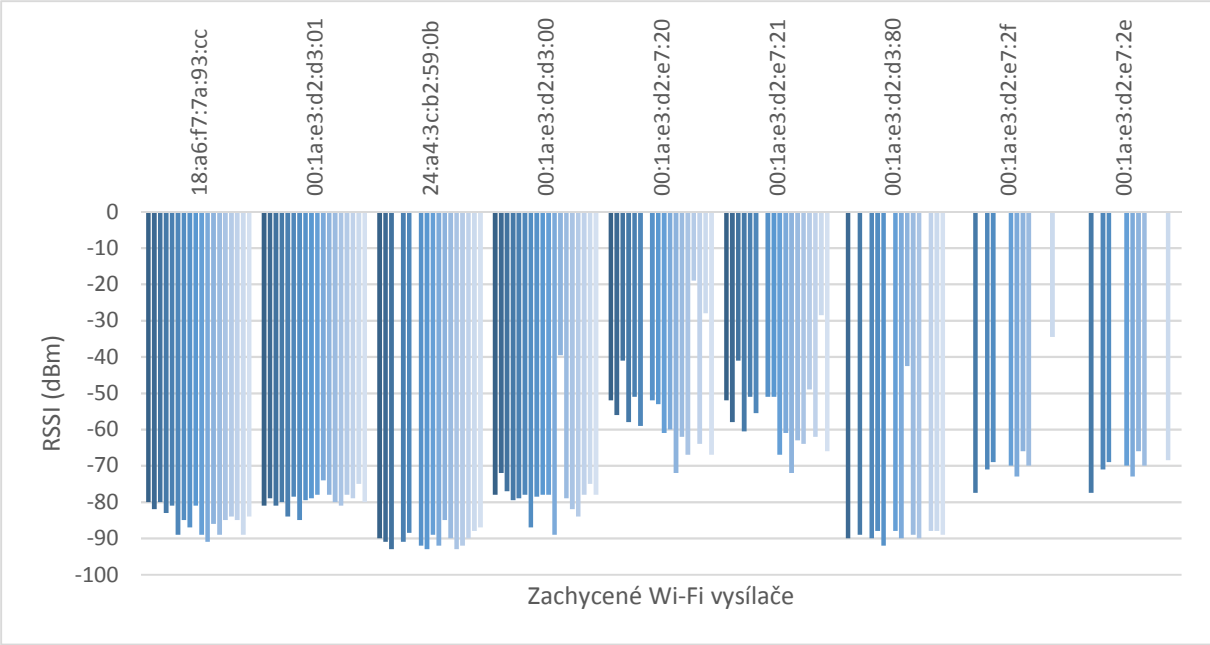
### 8.2.1 RSSI

Nejprve je třeba zjistit, jestli se hodnoty RSSI liší i na jiných místech. Dále bude u grafů sledováno, jak často dojde k tomu, že vysílač nebude při měření zachycen vůbec.



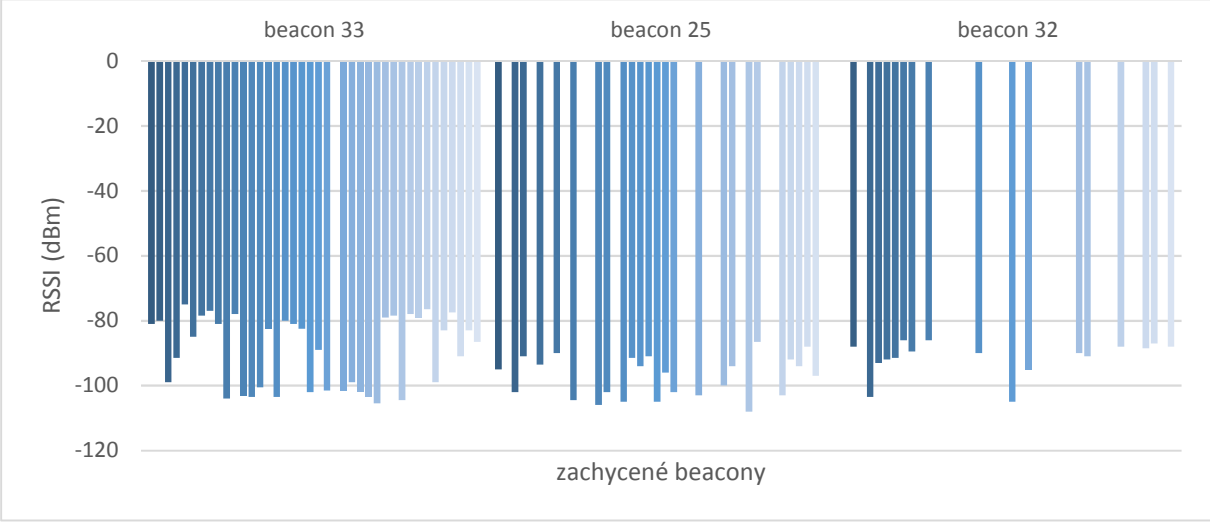

*Graf 11 Měření BLE RSSI na místě J*

*Zdroj: Vlastní zpracování, 2017*



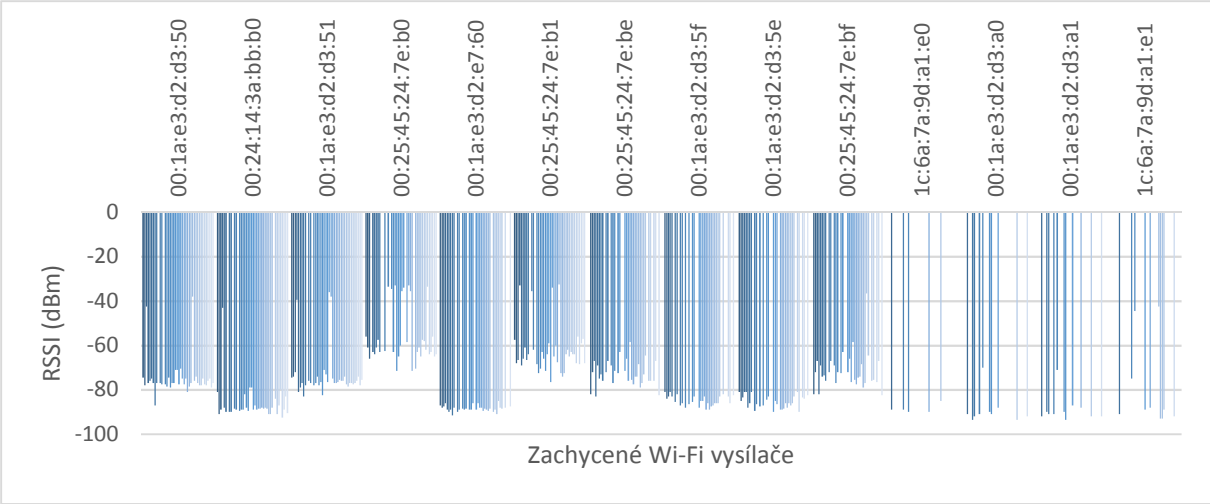

*Graf 12 Měření Wi-Fi RSSI na místě J*

*Zdroj: Vlastní zpracování, 2017*




*Graf 13 Měření BLE RSSI na místě E*

*Zdroj: Vlastní zpracování, 2017*



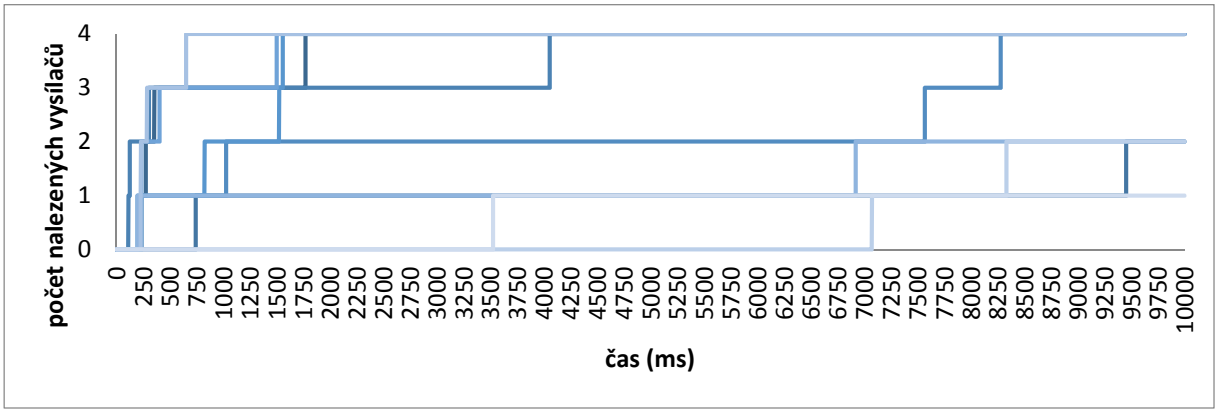

*Graf 14 Měření Wi-Fi RSSI na místě E*

*Zdroj: Vlastní zpracování, 2017*

U RSSI je z grafů vidět, že u Wi-Fi zůstává měření značně nepřesné. U BLE jsou větší výkyvy než v předchozím případě, stále však zůstává o dost stabilnější než Wi-Fi. Na těchto dvou místech je již vidět, že bližší BLE vysílače mají vyšší hodnoty RSSI a ve větším počtu případů se je podařilo při měření zachytit. Nicméně jak bylo uvedeno v teoretické části, pro účely lokalizace se očekává, že naměřená data budou podobná, což se zatím nedaří.

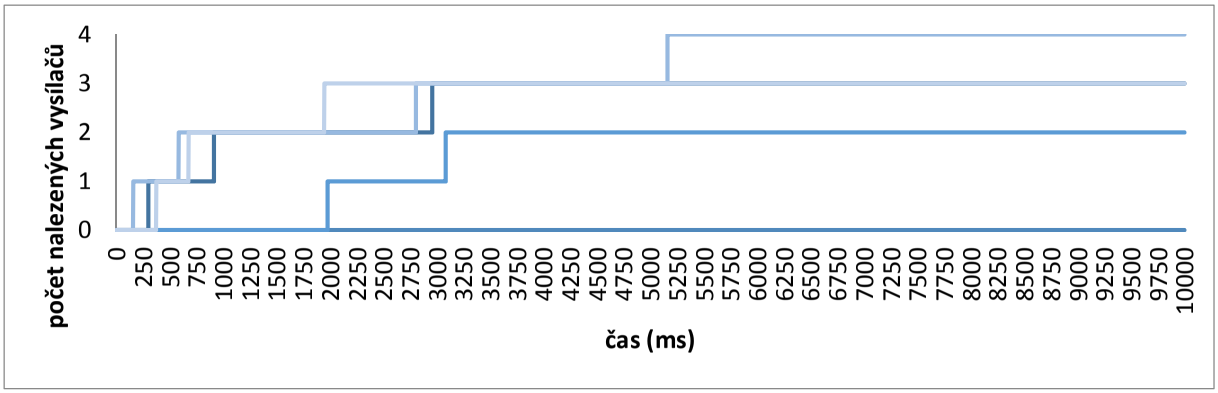
### 8.2.2 Čas

Pro načítání vysílačů v čase bude sledováno, jestli se i na jiných místech budou lišit v závislosti na zařízení. Dále bude zjištěno, jestli by bylo možné zkrátit čas skenu na méně než deset vteřin.



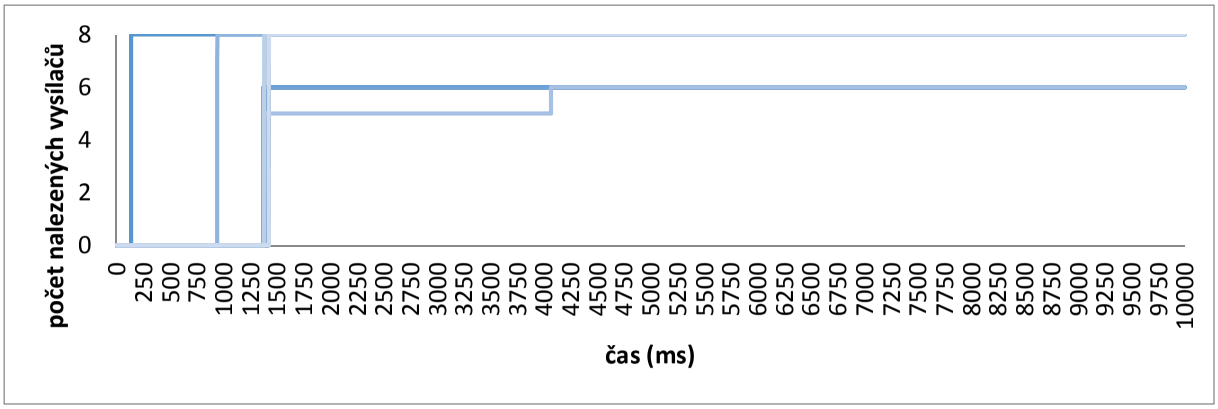

*Graf 15 Zachycení BLE vysílačů v čase – AsusZ00D, místo J*

*Zdroj: Vlastní zpracování, 2017*



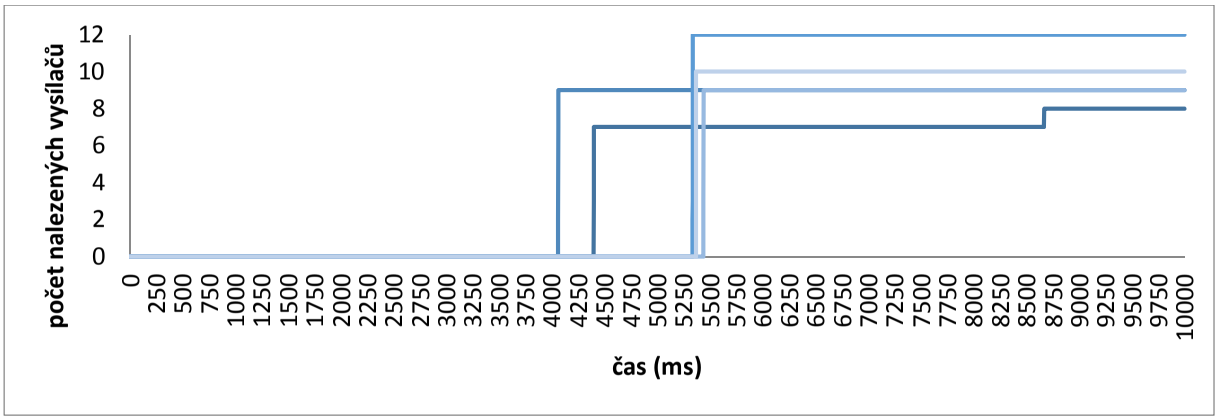

*Graf 16 Zachycení BLE vysílačů v čase – Sony F8331, místo J*

*Zdroj: Vlastní zpracování, 2017*



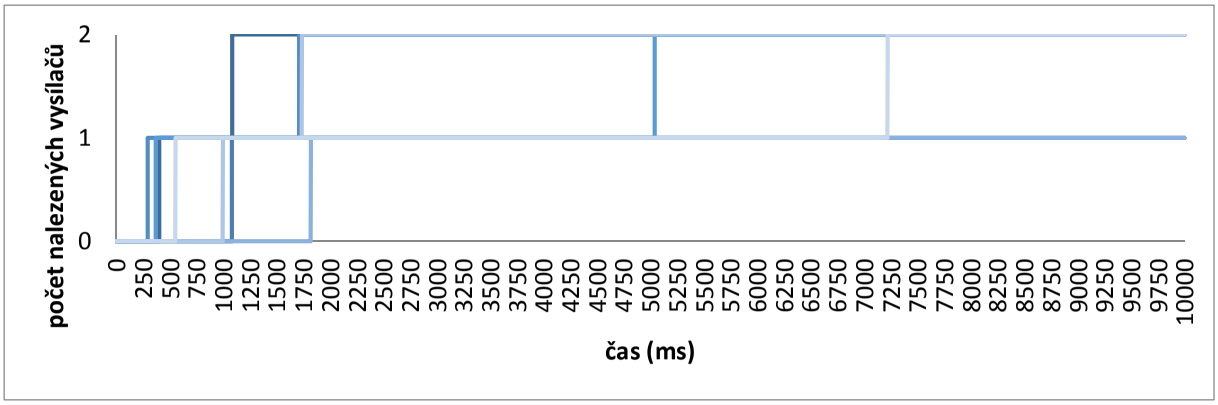

*Graf 17 Zachycení Wi-Fi vysílačů v čase – Asus Z00D, místo J*

*Zdroj: Vlastní zpracování, 2017*



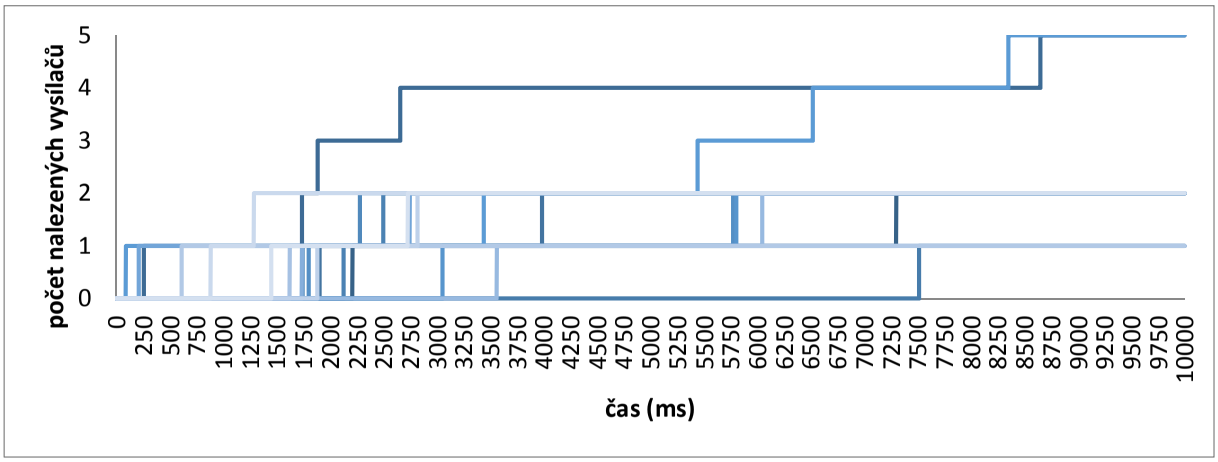

*Graf 18 Zachycení Wi-Fi vysílačů v čase – Sony F8331, místo J*

*Zdroj: Vlastní zpracování, 2017*



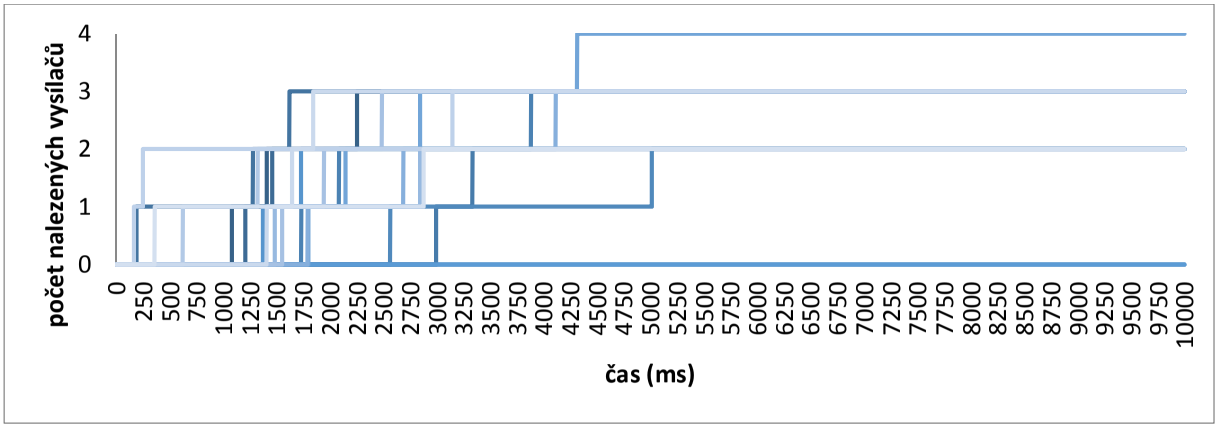

*Graf 19 Zachycení BLE vysílačů v čase – Xiaomi Redmi 3, místo E*

*Zdroj: Vlastní zpracování, 2017*



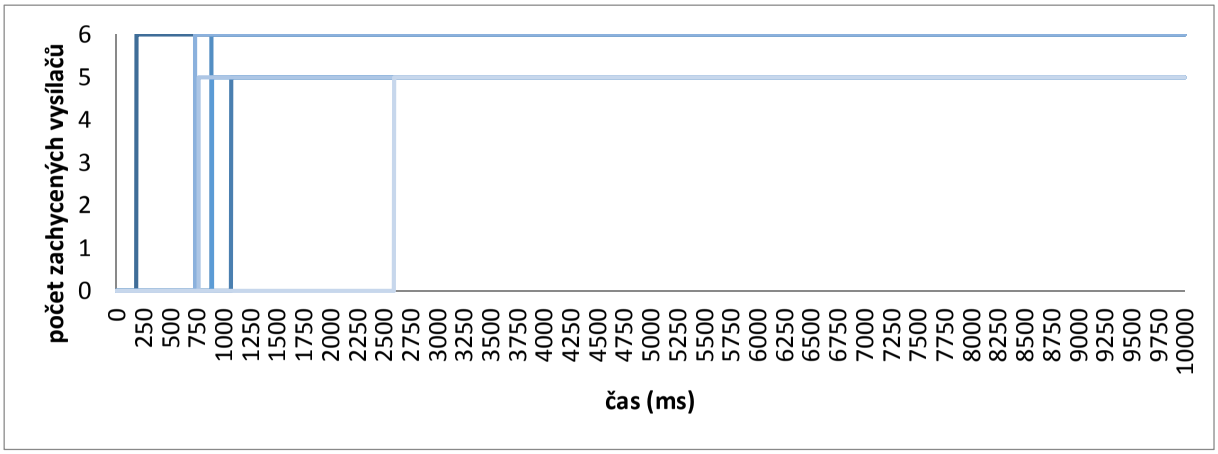

*Graf 20 Zachycení BLE vysílačů v čase – Lenovo YB1-X90L, místo E*

*Zdroj: Vlastní zpracování, 2017*



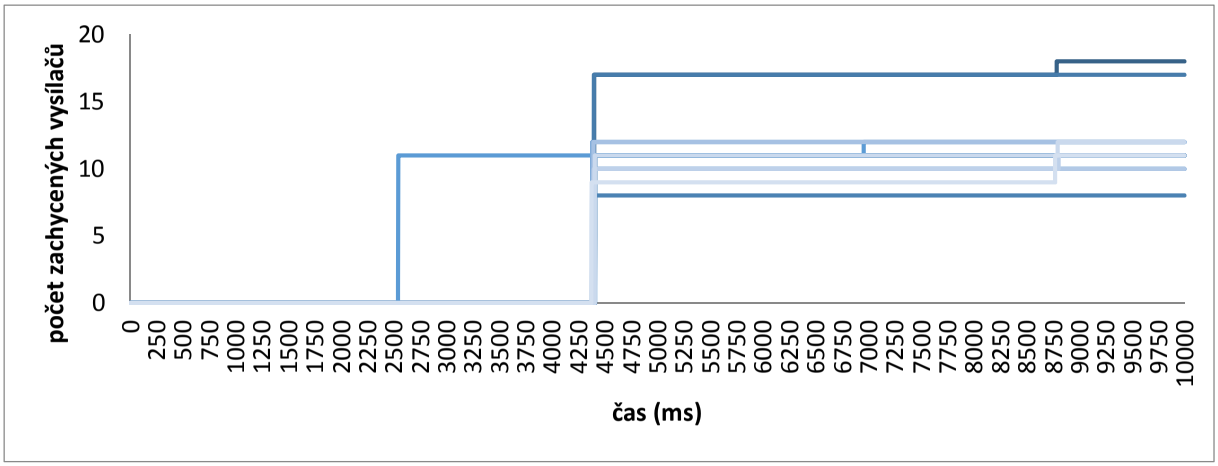

*Graf 21 Zachycení BLE vysílačů v čase – Sony F8331, místo E*

*Zdroj: Vlastní zpracování, 2017*



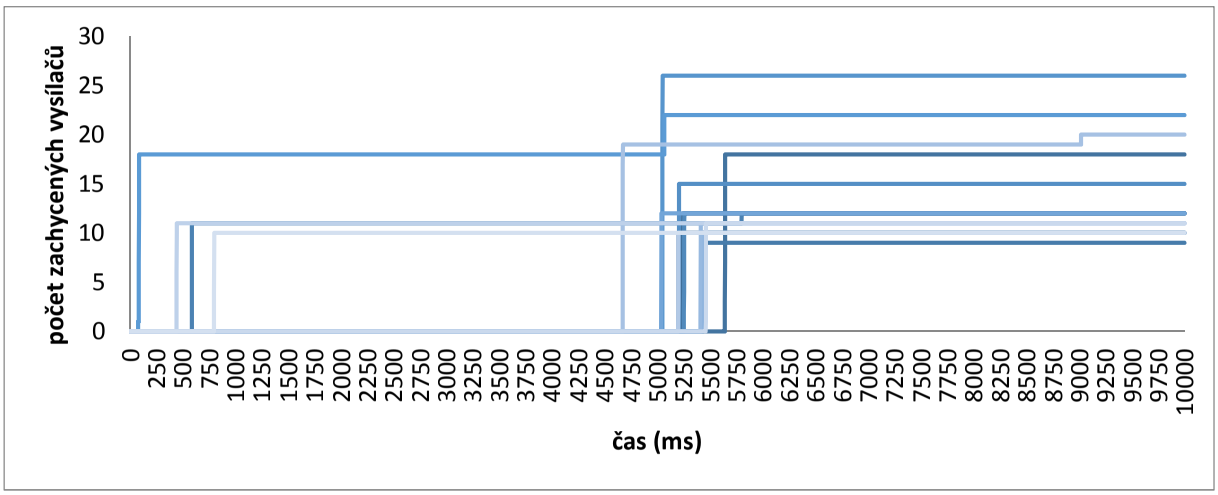

*Graf 22 Zachycení Wi-Fi vysílačů v čase – Xiaomi Redmi 3, místo E*

*Zdroj: Vlastní zpracování, 2017*




*Graf 23 Zachycení Wi-Fi vysílačů v čase – Lenovo YB1-X90L, místo E*

*Zdroj: Vlastní zpracování, 2017*




*Graf 24 Zachycení Wi-Fi vysílačů v čase – Sony F8331, místo E*

*Zdroj: Vlastní zpracování, 2017*

Z uvedených grafů je vidět, že doba, za kterou jsou vysílače nalezeny, skutečně závisí na zařízení. Znovu byly Wi-Fi nalezeny rychle, ale u BLE by mohlo hrozit, že při zkrácení doby skenování nebude nalezen požadovaný počet vysílačů. U Wi-Fi na rozdíl od BLE je navíc více skokové nalezení vysílačů, což je pravděpodobně způsobeno softwarovým nastavením zařízení.

# 9 Závěry a doporučení

V průběhu diplomové práce byla vytvořena aplikace za účelem sběru dat pro výzkum indoor lokalizace. Ta sestává ze dvou částí – serverové s vystaveným REST rozhraním a klientské pro Android zařízení.

Hlavním cílem bylo prostřednictvím této aplikace posbírat data potřebná pro lokalizaci a zjistit, jestli jsou pro ni vhodná. Bylo zjištěno, že není vhodné se spoléhat na data z GSM z důvodu, že jich je nedostatek. Je možné je použít pouze jako podporu, ale nelze na ně spoléhat. Data z Bluetooth a Wi-Fi vysílačů jsou v současnosti sbírána v dostatečném množství, ale hodnoty RSSI potřebné pro lokalizaci jsou různé. Dále byla sledována doba, za kterou jsou zařízení schopna získat signál od okolních vysílačů. U Wi-Fi vysílačů bylo jejich nalezení dost skokové, ale většinou byly nalezeny do šesti sekund. Bluetooth vysílače také většinou přibyly v první polovině časového limitu, ale často přibývaly i dále po celou dobu stanoveného časového limitu. Není proto vhodné zkracovat dobu skenování okolních sítí. U BLE i Wi-Fi byly rozdíly v počtu nalezených vysílačů mezi jednotlivými zařízeními. Pro další výzkum je jistě zajímavé i to, že častější bylo nalezení vzdálenějšího vysílače na chodbě než bližšího v místnosti, což znamená, že rušení signálu zdmi má velký vliv.

Aplikace je připravena pro další vývoj – je umístěna na sdílené repository, klientská část je připojena k systému pro monitorování chyb a webová část je připravena na vystavení webové aplikace jak pro uživatele, tak pro administrátory.

V dalším vývoji by bylo vhodné přidat rozpoznávání obrazu pro možnost využití více míst v budově a využití mimo budovu. Na serveru by pak za tímto účelem bylo dobré přidat uživatelské rozhraní pro administrátory, kde by se dala místa přidat ručně.

V případě dalšího vývoje na serverové části aplikace by bylo vhodné přidat gitflow pro lepší rozdělení práce a Jenkins pipeline, aby byla automaticky nasazována pouze ta část aplikace, která je umístěna v master branch.

# 10 Použité zkratky

BLE – Bluetooth Low Energy

BSSID – Basic Service Set Identifier, MAC adresa přístupového bodu

CI – Continuous Integration, způsob vývoje software

CRUD – Create Read Update Delete, základní operace pro úpravu dat

EL – Expression Language, jazyk umožňující vyjádření hodnoty jako odkazu

IDE – Integrated Development Environment, prostředí pro vývoj software

JVM – Java Virtual Machine, virtuální stroj ke spouštění Java souborů

MAC – Media Access Control, adresa síťového zařízení

ORM – Object Relational Mapping, mapování databázových tabulek na modelové třídy

REST – Representational State Transfer

RDP – Remote Desktop Protocol, protokol umožňující ovládat vzdálený počítač

RSSI – Received Signal Strength Indicator, indikátor síly přijímaného signálu

UUID – Universally Unique Identifier – jednoznačný identifikátor

# 11 Zdroje

*AndroidAnnotations* [online]. 2017 [cit. 2017-06-21]. Dostupné z: http://androidannotations.org/

APPLE INC. Getting started with iBeacon. *Apple Developer* [online]. 2014 [cit. 2017-07-25]. Dostupné z: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf

BERGMANN. Co je GPS? Historie a úvod do problematiky. *Svět mobilně* [online]. 12.12.2005 [cit. 2015-01-31]. ISSN 1214-2190. Dostupné z: http://www.svetmobilne.cz/co-to-je-gps-historie-a-uvod-do-problematiky/244

CLARKSON, Roy. *Spring for Android Reference Documentation* [online]. ©2010-2014 [cit. 2017-07-30]. Dostupné z: http://docs.spring.io/spring-android/docs/2.0.0.M3/reference/html/

COMDEAL, s.r.o. Mobilní datová spojení. *Comdeal - Software solutions* [online]. 2003-11 [cit. 2015-01-28]. Dostupné z: http://www.comdeal.cz/technologie-mobilni-datova-spojeni.php

FOWLER, Martin. Continuous integration. *Martin Fowler* [online]. 2006 [cit. 2016-09-18]. Dostupné z: http://martinfowler.com/articles/continuousIntegration.html

GOOGLE. *Android Developers* [online]. 2017 [cit. 2017-06-21]. Dostupné z: https://developer.android.com

GOOGLE. Caching Bitmap. *Android Developers* [online]. 2017 [cit. 2017-08-09]. Dostupné z: https://developer.android.com/topic/performance/graphics/cache-bitmap.html

GOOGLE. *Centrum zásad pro vývojáře* [online]. 2017 [cit. 2017-08-01]. Dostupné z: https://play.google.com/intl/cs_ALL/about/developer-content-policy/

GOOGLE. Design. *Android Developers* [online]. 2017 [cit. 2017-08-09]. Dostupné z: https://developer.android.com/design/index.html

JENKINS. *Jenkins Wiki* [online]. 2017 [cit. 2017-08-09]. Dostupné z: https://wiki.jenkins.io/display/JENKINS/Home

KOCMAN, Rostislav. Jak se zjišťuje poloha mobilního telefonu?. MAFRA, a.s. *Mobil: Vše o mobilech, operátorech a telekomunikacích* [online]. 23. července 2001 [cit. 2015-01-28]. Dostupné z:http://mobil.idnes.cz/jak-se-zjistuje-poloha-mobilniho-telefonu-fi3-/mob_tech.aspx?c=A010719_0036942_mob_tech

KŘÍŽ, Pavel, Filip MALÝ a Tomáš KOZEL. *Improving Indoor Localization Using Bluetooth Low Energy Beacons* [online]. Department of Informatics and Quantitative Methods, Faculty of Informatics and Management, University of Hradec Kralove, Rokitanskeho 62, 500 03 Hradec Kralove, Czech Republic, 2016 [cit. 2017-07-17]. Dostupné z: https://www.hindawi.com/journals/misy/2016/2083094/

LEON, Vojtěch. *Gamifikace sběru fingerprintů bezdrátových sítí*. Hradec Králové, 2016. Bakalářská práce. UHK, Fakulta informatiky a managementu. Vedoucí práce Ing. Pavel Kříž, Ph.D.

MATOULEK, Dominik. *Lokalizace uvnitř budov*. Hradec Králové, 2015. Bakalářská práce. UHK, Fakulta informatiky a managementu. Vedoucí práce Ing. Pavel Kříž, Ph.D.

ORLICH, M. Základní lokalizační metody v GSM. ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE, Fakulta elektrotechnická, Katedra telekomunikační techniky. Access server [online]. 28. 02. 2006 [cit. 2015-01-31]. Dostupné z: http://access.feld.cvut.cz/view.php?cisloclanku=2006022801

PIVOTAL SOFTWARE, INC. Understanding REST. *Spring* [online]. 2017 [cit. 2017-01-21]. Dostupné z: https://spring.io/understanding/REST

REDHAT. Getting started with Hibernate Validator. *Hibernate: Everything data* [online]. 2017 [cit. 2017-01-29]. Dostupné z: http://hibernate.org/validator/documentation/getting-started/

*Webové služby nad IS/STAG* [online]. 2017 [cit. 2017-08-09]. Dostupné z: https://stagws.uhk.cz/ws/web/

ZAVŘEL, Roman. IOS a Android vymazaly Windows Phone ze světa. ZAVŘEL MEDIA. *Letem světem Applem: Magazín o společnosti Apple a produktech Apple* [online]. 2016 [cit. 2017-06-05]. Dostupné z: https://www.letemsvetemapplem.eu/2016/08/19/ios-android-vymazaly-windows-phone-ze-sveta/

# Seznam obrázků

# Seznam tabulek

# Seznam grafů

# Seznam příloh

# Spuštění aktivity

Spustí aktivitu pro aktuálního uživatele

- **URL**

  /android/1.0/aktivity/start

- **URL parametry**

  `materialAmount=[integer]`

  - počet materiálu použitého na aktivitu (ve většině případů počet dělníků

- **Zaslaná data**

  place

  - Místo, na kterém má být aktivita spuštěna

- **Success Response:**
  - **AppUser** – vrací aktualizovaného uživatele

# Zakoupení předmětů

Přidá aktuálnímu uživateli předměty pro vylepšení těžby

- **URL**

  /android/1.0/aktivity/buy

- **Zaslaná data**

  List<Item>

  - Seznam předmětů, které uživatel zakoupil

- **Success Response:**
  - **AppUser** – vrací aktualizovaného uživatele

# Získání dostupných předmětů

Vrátí seznam předmětů, které si může aktuální uživatel koupit

- **URL**

  /android/1.0/activity/getItems

- **Success Response:**
  - **AppUser** – vrací aktualizovaného uživatele

## Uložení fingerprintu

Uloží zaslaný fingerprint do databáze

- **URL**

  /android/1.0/fingerprint/save

- **Zaslaná data**

  Nový fingerprint

## Přihlášení uživatele

Přihlásí uživatele do aplikace pomocí IS STAG

- **URL**

  /android/1.0/login

- **Metoda**

  `POST`

- **URL parametry**

  `username=[String]`

- **Hlavička**

  `authorization=[String]`

  - Hlavička obsahující uživatelské údaje k autorizaci – Base auth

- **Success Response:**
  - **AppUser** – vrací aktualizovaného uživatele

- **Error Response:**
    - **Code:** 401 UNAUTHORIZED

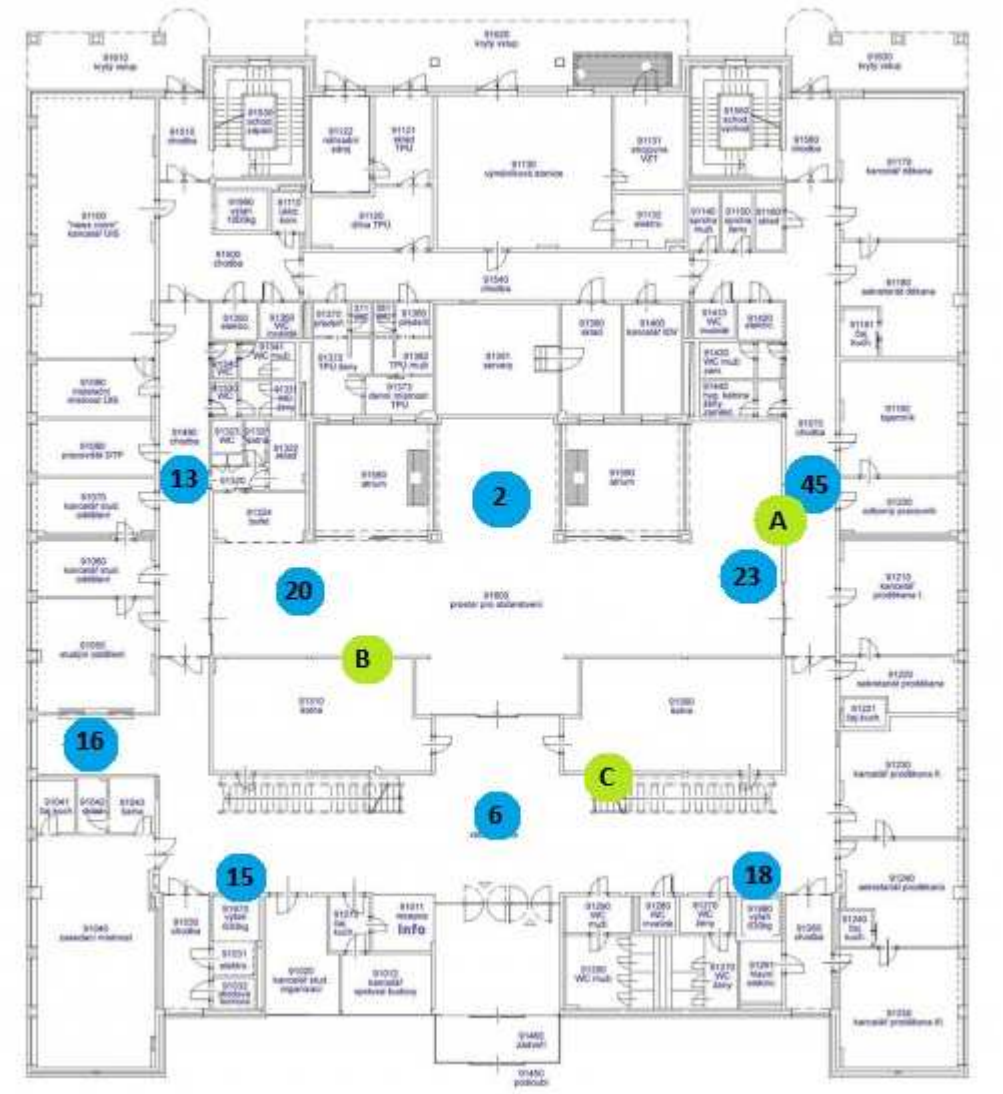
## Získání místa

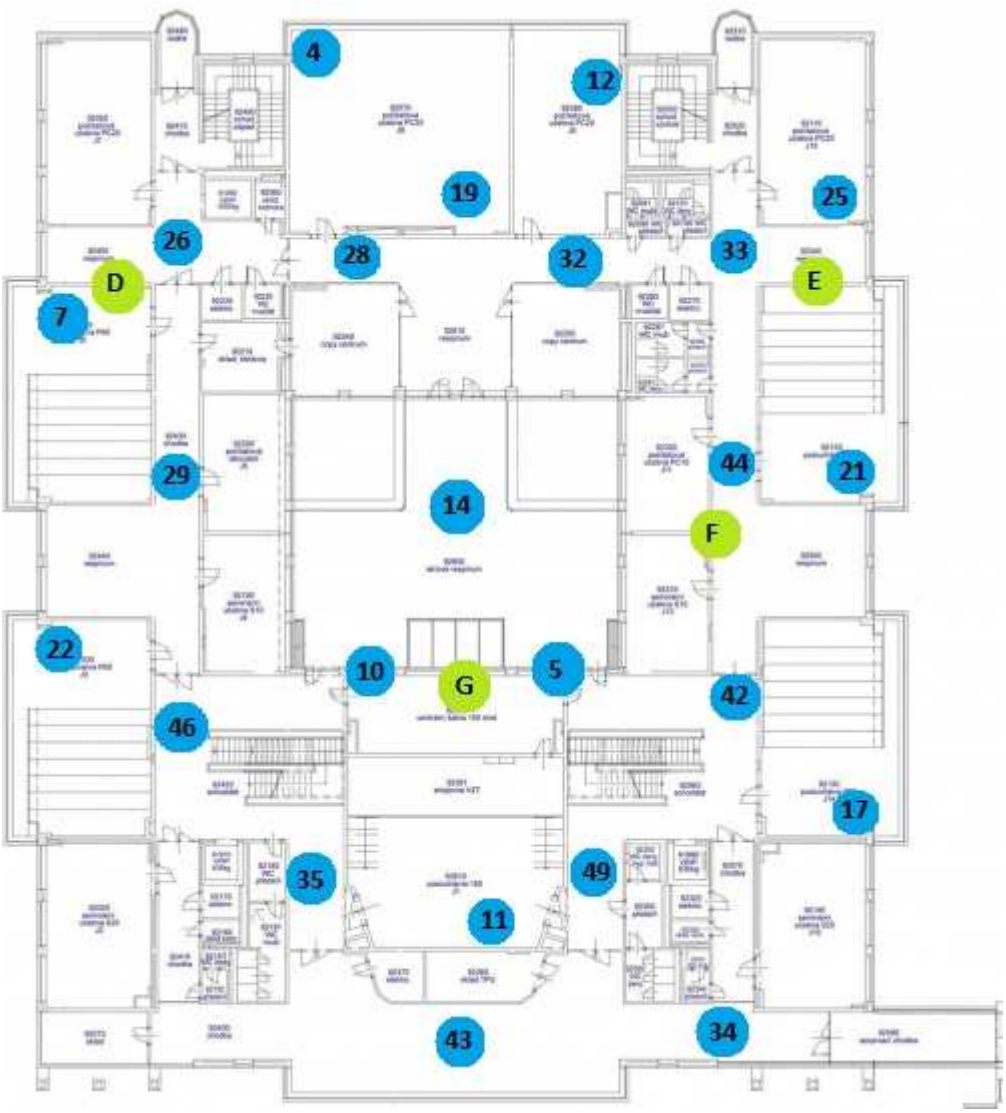
Vrátí místo dle zaslaného kódu

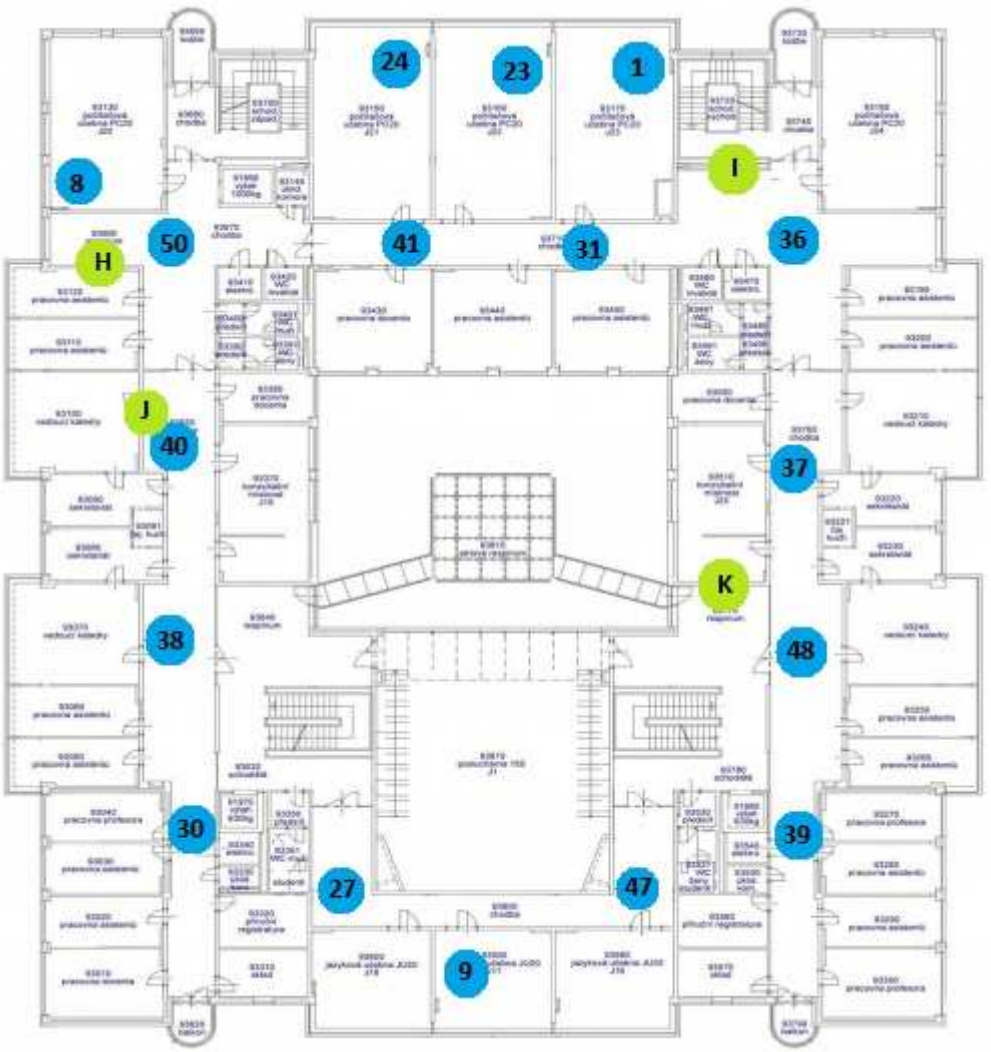
- **URL**

  /android/1.0/qr/{code}

- **Success Response:**
    - **Place** – místo se zadaným kódem (null v případě, že nic nenajde)


13
2
45
A
20
23
B
16
C
6
15
18
Info


4
12
19
25
26
28
32
33
D
E
7
29
44
21
14
F
22
10
G
5
42
46
17
35
49
11
43
34


24
23
1
I
8
50
H
41
31
36
J
40
37
K
38
48
30
39
27
47
9

/Zdrojové kódy

/fingerprint-game – *Zdrojové kódy serverové části aplikace*

/fingerprint-game-android – *Zdrojové kódy klientské části aplikace*

/Excel – *Přehled naměřených dat v .xlsx sešitech a seznam MAC adres beaconů dle čísel v mapě*

/Mapy – *Mapy s vyznačenými beacony a místy ve hře*

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/kombi: Aplikovaná informatika (a2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

| PŘEDKLÁDÁ: | ADRESA | OSOBNÍ ČÍSLO |
|---|---|---|
| Hamšová Kateřina | U Nuhena 330, Hradec Králové - Plotiště nad Labem | I1500684 |

TÉMA ČESKY:

Gamifikace sběru rádiových fingerprintů

TÉMA ANGLICKY:

Gamification of radio-fingerprinting acquisition

VEDOUCÍ PRÁCE:

Ing. Pavel Kříž, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Navrhnout a implementovat hru pro operační systém Android. Hra bude sloužit pro sběr obsahů rádiových sítí v okolí budů vyznačených v reálném prostoru pomocí QR kódů. Součástí řešení bude i serverová část.

Osnova:
1. Úvod
2. Analýza
3. Návrh
4. Implementace
5. Testování
6. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

Chris Schaefer, Clarence Ho, Rob Harrop: Pro Spring, 4th Edition
http://martinfowler.com/articles/continuousIntegration.html
https://www.hindawi.com/journals/wcis/2016/2083094/
Rudolf Pecinovský: Návrhové vzory

Podpis studenta: ..... Datum: 5. 10. 2016

Podpis vedoucího práce: ..... Datum: 5. 10. 2016