Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Vývoj mobilní aplikace s prvky gamifikace

Bakalářská práce

Autor: Pavol Podstrelený
Studijní obor: Aplikovaná informatika

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

Hradec Králové duben 2018

Prehlásenie:

Prehlasujem, že som bakalársku prácu spracoval samostatne a s použitím uvedenej literatúry.

V Hradci Králové dňa 25.4.2018 Pavol Podstrelený

Poďakovanie:

Ďakujem vedúcemu bakalárskej práce doc. Ing. Filipovi Malému, Ph.D. za metodické vedenie práce a hodnotné pripomienky k tejto práci.

Anotácia

Táto bakalárska práca je zameraná na vývoj mobilnej aplikácie s prvkami gamifikácie. V prvej časti práce je stručne predstavená gamifikácia a s ňou spojená motivácia a typológia hráčov v gamifikovanom systéme. Táto časť sa taktiež zaoberá základnými hernými mechanikami gamifikovaného systému. Ďalšiu časť práce tvorí predstavenie a porovnanie základných architektonických vzorov využívaných pri návrhu aplikácie v operačnom systéme Android. Nasledujúca časť práce sa venuje návrhu aplikácie s použitím android architecture components. Predstaveniu mobilnej aplikácie s prvkami gamifikácie je venovaná posledná časť práce.

Hlavným výsledkom tejto bakalárskej práce je vytvorenie mobilnej aplikácie s prvkami gamifikácie pre žiakov základných škôl v predmete matematika.

Annotation

Title: Development of a mobile application with gamification elements

This bachelor thesis focuses on the development of a mobile application with gamification elements. The first part of the thesis briefly presents gamification, associated motivation and typology of players in a gamified system. In addition, this section also deals with basic game mechanics of this system. Following part of the work consists of an introduction to and a comparison of basic architectural patterns used in the process of designing an application in Android operating system. The fourth section is dedicated to designing an application using Android Architecture Components. The final part of the work is devoted to introducing a mobile application with gamification elements.

The main result of this bachelor thesis is the creation of a mobile application with gamification elements for elementary school mathematics class.

Obsah

Zoznam obrázkov

1 Úvod

Počet mobilných zariadení za posledné desaťročia enormne vzrástol. V roku 2017 operačný systém Android (ďalej len OS Android) prekonal hranicu dvoch biliónov aktívnych zariadení a viac ako 50% vyhľadávaní na Google je realizovaných práve týmito zariadeniami. Okrem toho je možné sledovať každoročný nárast mobilných aplikácií vyvíjaných pre tieto zariadenia. Podľa (Google developers, 2016) bolo v roku 2015 z oficiálneho zdroja Google Play nainštalovaných viac ako 65 biliónov aplikácii. Z týchto informácií plynie, že ľudia využívajú mobilné telefóny v každodennom živote na komunikáciu, učenie sa nových vecí, zábavu, online nákupy a na mnoho ďalších aktivít.

Pre vývojára, ktorý vyvíja aplikácie pre OS Android to predstavuje obrovský potenciál no taktiež riziko. Na Google Play je možné nájsť množstvo navzájom podobných aplikácií (vytvorených rôznymi vývojármi a ponúkajúcich rovnaké využitie), no je len na užívateľoch mobilných zariadení, ktorú aplikáciu sa rozhodnú vybrať. Vývojár sa tak dostáva do situácie, keď musí ponúknuť užívateľovi nie len predpokladanú funkčnosť aplikácie, ale aj jednoduchú ovládateľnosť, užívateľsky prívetivý design a taktiež zaručiť, aby bol užívateľ motivovaný, lojálny (ostal „verný“ danej aplikácii) a strávil čo najviac času využívaním ich aplikácie. Jednou z možností, ktorá môže prispieť k dosiahnutiu tohto cieľa je využitie gamifikácie.

Gamifikácií sa venuje prvá kapitola práce, ktorá stručne zoznamuje čitateľa s týmto pojmom. Dôležitou úlohu pri aplikovaní gamifikovaného systému hrá motivácia a typologické rozloženie hráčov. V závere kapitoly sú predstavené herné mechaniky, ktoré pri správnom implementovaní poskytujú želané emocionálne reakcie vyvolané u hráča pri interakcii s gamifikovaným systémom.

Okrem ponúknutia jedinečnej skúsenosti pri užívaní aplikácie je nutné venovať pozornosť technickej stránke vývoja. Pri raste užívateľskej základne musí vývojár reagovať na skutočnosť, že títo užívatelia využívajú zariadenia s rôznou pamäťou, výpočtovou silou a rozdielnou verziou operačného systému. Okrem tejto fyzickej rôznorodosti zariadení musí taktiež počítať so zmenami požiadaviek na systém

v priebehu času. Použitie správneho architektonického vzoru sa tak stáva kľúčovým faktorom, ktorý môže kladne alebo záporne ovplyvniť ďalší vývoj aplikácie.

Časť bakalárskej práce je venovaná predstaveniu architektonických vzorov využívaných pri vývoji Android aplikácií. Hlavným účelom tejto kapitoly je zoznámiť čitateľa s predstavenými architektonickými vzormi a poukázať na ich využitie.

Bližšia pozornosť je venovaná návrhu aplikácie s použitím android architecture components, ktoré boli predstavené v roku 2017 na I/O konferencii spoločnosťou Google. Týmto komponentom je taktiež venovaná samostatná kapitola práce.

Záver bakalárskej práce je venovaný predstaveniu android aplikácie s prvkami gamifikácie. Táto časť stručne popisuje účel aplikácie, využitie herných mechaník, použité technológie a architektúru aplikácie.

Hlavným cieľom bakalárskej práce je navrhnúť a implementovať mobilnú aplikáciu s prvkami gamifikácie pre žiakov základných škôl v predmete matematika.

2 Gamifikácia

(Werbach et. al, 2012) definujú gamifikáciu ako *„využitie herných elementov a techník herného dizajnu v nehernom prostredí“*. (Zichermann et. al, 2011) vo svojej knihe definujú gamifikáciu ako *„proces herného myslenia a herných mechanik, ktoré angažujú užívateľa a riešia problémy“*.

Aj keď sa pojem gamifikácia môže zdať nový, myšlienka využitia herného myslenia a herných mechaník siaha niekoľko storočí do histórie ľudstva . (Zichermann et. al, 2011) uvádzajú, že armáda používa hry a simulácie stovky rokov, pričom základy pre pochopenie motivácie hráča boli položené už v 18 storočí škótskym filozofom Davidom Humeom. Taktiež poukazujú na to, že už od roku 1960 spisovatelia píšu knihy o „hernej“ stránke života.

Aj v súčasnosti sa prirodzená nákladnosť k hrám a zábave objavuje v každodennom živote ľudí. Gamifikácia sa stáva súčasťou množstva systémov a služieb, ktoré spoločnosti poskytujú. V každodennom živote sa stretávame s rôznymi príkladmi využitia gamifikácie. Jedným z príkladov sú vernostné programy, ktoré poskytujú svojim zákazníkom na základe získaných bodov rôzne produkty a zľavy, ktoré im boli pridelené pri nákupe tovaru.

Pre správne pochopenie a aplikovanie gamifikácie je nutné sa zoznámiť s jej základnými pojmami. V nasledujúcom texte sú predstavené typológie motivácie a hráčov, ktoré nám pomáhajú poznať zákazníkov. Následne sú predstavené herné mechaniky, set nástrojov, ktoré pri správnom použití poskytujú želané emocionálne reakcie vyvolané u hráča pri interakcii so systémom. Pri správnom aplikovaní týchto techník je možné urobiť vytváraný systém viac zábavný, zmysluplný a orientovaný na zákazníkov.

2.1 Motivácia a typológia hráčov

Rôznorodosť charakteristík hráčov, ktorí prichádzajú s gamifikovaným systémom je obrovská. Hráči vstupujú do gamifikovaného systému s rozličnou úrovňou motivácie a majú rozličné očakávania a požiadavky na tento systém. Preto je

dôležité pri tvorbe gamifikovaného systémov vedieť čo najviac o motivácii hráčov a ich typologickom rozložení.

2.1.1 Rozdelenie motivácie

Jedným z dôležitých aspektov pri tvorbe úspešného gamifikovaného systému je správna motivácia hráča. Motivácia predstavuje základný „hnací motor“ ľudí pri akejkoľvek vykonávanej činnosti. Pokiaľ je osoba dobre motivovaná, činnosť ktorú vykonáva sa stáva pre ňu zmysluplnou, nie je jej ľahostajná a vidí v nej vyšší význam. Na druhej strane, nepochopenie významu resp. nedostatočný zmysel vytvára u danej osoby nezáujem a ľahostajnosť k činnosti a tým hráča demotivuje.

Z psychologického hľadiska je motivácia rozdelená na vonkajšiu a vnútornú. Vnútorná motivácia vychádza z vlastného záujmu o danú činnosť, pričom motivátorom nie je žiadny externý faktor. Naproti tomu vonkajšia motivácia vychádza z prostredia okolo nás.

(Zichermann et. al, 2011) poukazujú na to, že sa môžeme stretnúť s troma základnými myšlienkami (pokiaľ ide o porovnanie vonkajšej a vnútornej motivácie v gamifikovanom, motivačnom prostredí).

Prvá myšlienka poukazuje na to, že pri plnení komplexných úloh sú peniaze veľmi slabou motiváciou. Pokiaľ ľudia dostávajú úlohy kreatívneho typu, alebo pri ktorých musia nájsť originálny spôsob riešenia, peniaze, ako druh vonkajšej motivácie, nefungujú.

(Zichermann et. al, 2011) vo svojej knihe uvádzajú ďalšiu myšlienku, ktorú prezentuje Dr. John Houston, vedúci výskumu v oblasti konkurencieschopnosti. Houston zistil, že mimoriadne súťaživí ľudia môžu byť seba-deštruktívni. Poukazuje na to, že títo ľudia súťažia aj keď neexistuje žiadna odmena, ktorú by mohli v tejto súťaži získať, pričom do každej činnosti dávajú všetku svoju energiu.

Posledná myšlienka vychádza z náhrady vnútornej motivácie za vonkajšie odmeny. (Zichermann et. al, 2011) popisujú, že pokiaľ dieťa hrá na klavíri preto, lebo mu to prináša radosť a „baví ho to“, môže po začlenení do súťaživej skupiny (súťaživého prostredia) zmeniť svoje správanie. Ak vyhrávalo súťaže, no neskôr dôjde

k niekoľkým prehrám za sebou, je veľká pravdepodobnosť, že s hraním úplne skončí. Vonkajšia motivácia teda „rozdrví“ vnútornú motiváciu a tá sa už nikdy nemusí vrátiť.

Veľkou výzvou pri návrhu gamifikovaného systému je preto spoznanie motivácie zákazníka a navrhnutie správneho resp. dostatočne motivujúceho systému odmeňovania.

2.1.2 Motivácia spojená s hraním hier

Náklonnosť k hrám možno pozorovať v každodennom živote. Ľudia sa s hraním stretávajú už od skorého detstva – najmä v rodinnom prostredí. Zmyslom týchto hier nie je len zábava. Hry pomáhajú napríklad aj pri zlepšení motoriky dieťaťa. Čím je dieťa staršie, tým viac objavuje ich krásu a rôznorodosť reálnych či virtuálnych hier. Hry nachádzame v dnešnej modernej dobre na každom kroku, sú neoddeliteľnou súčasťou nášho života.

Prečo hrajú ľudia radi hry? Pri hraní zažíva hráč emócie ako je radosť, šťastie ale hlavne zábavu. Nicole Lazzaro (2004) poukazuje na to, že hra nám poskytuje 4 druhy zábavy, bližšie špecifikované emóciou, ktoré definuje nasledovne:

- **Ťažká zábava** (hard fun): pre mnohých hráčov je prekonávanie prekážok hlavný dôvod hry. U hráčov sú vzbudzované emócie, ktoré prichádzajú s triumfom či frustráciou. Hráči väčšinou tento typ hry obľubujú. Je to kvôli tomu, že môžu zistiť, aký dobrý v hre naozaj sú. Emócie vyplývajúce z ťažkej zábavy sú zamerané na zmysluplné výzvy, stratégie a hádanky.
- **Ľahká zábava** (easy fun): skupina hráčov, ktorí preferujú ľahkú zábavu sa zameriava na plný pôžitok prežívania aktivít. Pocity úžasu či tajomstva môžu byť pri hraní ľahkých hier veľmi intenzívne. Typ týchto hier sa často odohráva vo svete, ktorý treba preskúmať a je plný záhad.
- **Zábava ako terapia**: (altered state): jeden z hlavných dôvodov, prečo hrajú hráči tento typ hry je zmena pocitov, ktoré pri hraní

prežívajú. Tešia sa svojim vnútorným zmenám počas a po skončení hry, kde často dochádza k pocitom uvoľnenia a odreagovania sa.

- **Sociálna zábava**: (people factor): mnoho hráčov sa zameriava na zábavu s ostatnými hráčmi. V niektorých prípadoch hráči v skutočnosti hru radi nemajú, no hrajú ju len kvôli vzájomnej interakcii s ostatnými hráčmi. Pri plnení spoločných cieľov v hre sa interakcia medzi hráčmi zvyšuje a narastá tímová práca a priateľstvo medzi hráčmi.

2.1.3 Typy hráčov

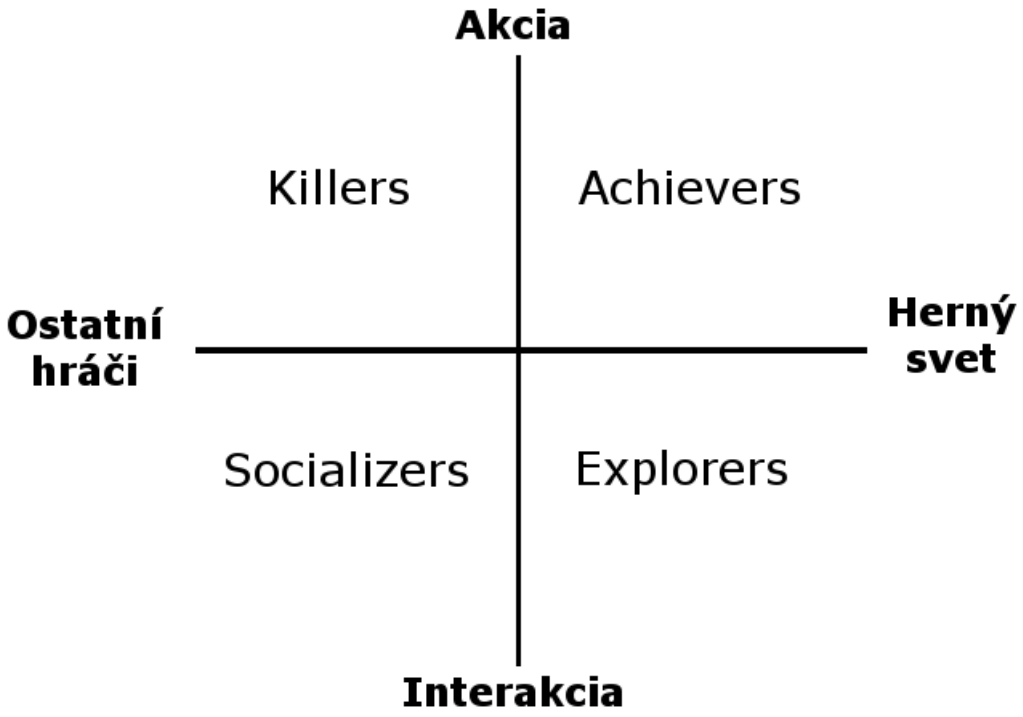
Britský profesor Richard Bartle prezentoval v roku 1996 článok s názvom "Hearts, Clubs, Diamonds, Spades: Players who suit MUDS". V tomto článok definuje a popisuje 4 prístupy hrania MUDS (multiplayer real-time virtual world) hier a predstavuje základnú typológiu hráčov. Bartle (1996) založil svoj výskum na pozorovaní hráčov online hier, ktoré viedlo k definovaniu štyroch, medzi ľuďmi obľúbených, herných kategórií:

- **Úspech v kontexte hry:** hráči si dávajú vlastné ciele a dôrazne sa usilujú o ich dosiahnutie.
- **Preskúmanie hry:** hráči sa snažia zmapovať topológiu virtuálneho sveta a dozvedieť o ňom čo najviac.
- **Socializovanie s ostatnými hráčmi:** hráči často využívajú hru ako komunikačný nástroj, pomocou ktorého môžu hovoriť alebo inak komunikovať so svojimi spoluhráčmi.
- **Vyvyšovanie sa nad ostatnými:** hráči používajú nástroje, ktoré hra poskytuje na spôsobenie utrpenia, alebo v zriedkavom prípade pomoc ostatným hráčom.

Bartle (1996) zistil, že je možné rozdeliť hráčov podľa dvoch základných kritérií. Najprv definoval dve skupiny hráčov. Prvá skupina sa zaujíma o herný svet a druhá skupina skôr o hráčov. Taktiež zistil, že niektorí hráči preferujú vykonávanie individuálnych činností voči ostatným hráčom či hernému svetu (akcia), zatiaľ čo

ďalší uprednostňujú činnosti, do ktorých sú zahrňovaní spolu s ostatnými hráčmi alebo prostredím (interakcia). Bartle (1996) na základe tohto rozdelenia definoval 4 typológie hráčov, ktoré definuje a popisuje nasledujúcim spôsobom:

- **Achievers**: (šplhúni), zameriavajú sa na akcie spojené s herným svetom.
- **Explorers:** (prieskumníci), špecifikujú sa na interakcie spojené s herným svetom.
- **Socializers:** (socializátori), upriamujú sa na interakcie, ktoré sú spojené s ostatnými hráčmi.
- **Killers:** (zabijaci), orientujú sa na akcie voči ostatným hráčom.



Obrázok 1: Graf typológie hráčov

Prevzaté z Bartle(1996), upravené autorom

2.1.3.1 Achievers

Tento typ hráčov je zameraný na zhromažďovanie bodov a zvyšovanie svojej úrovne v hre, pričom celú svoju aktivitu podriaďujú dosiahnutiu týchto cieľov. Nezameriavajú sa na porážku ostatných hráčov, skôr im ide o sebarealizáciu. Medzi ich hlavné aktivity patrí zbieranie odznakov, vyznamenaní, cenných predmetov či zvyšovanie úrovní. Záujem o spoznávanie herného sveta sa prejaví iba v prípade, že im to pomôže pri zlepšení ich úrovne, poprípade pri získaní bodov. Svoju interakciu s ostatnými hráčmi využívajú skôr na získanie informácií a vedomostí, pomocou ktorých môžu zvýšiť svoje bodové postavenie v hre. Títo hráči sa uchyľujú k zabitiu iba v prípade, že im ich hlavný rival stojí v ceste pri dosiahnutí svojho cieľa alebo v prípade, že zabitie druhého hráča je bodovo ohodnotené. Hráči často analyzujú pravidlá a herné mechaniky, ktoré hra prináša, za cieľom vytvorenia stratégie. Tá im môže pomôcť k efektívnemu dosiahnutiu bodov.

2.1.3.2 Explorers

Hráči typu „explorers“ sa zameriavajú hlavne na dôkladné skúmanie, objavovania a prezeranie herného sveta. Zaujímajú sa o dejové línie hier, hľadajú zaujímavé funkčnosti sveta a radi zisťujú fungovanie vecí do detailu. Často získavajú body len za účelom odomknutia uzamknutého prostredia, ktoré môžu prebádať. Zabíjanie protihráčov sa u tohto typu vyskytuje zriedka, pretože by im to mohlo priniesť veľa ťažkostí spojených s budúcou odplatou protihráča. S ostatnými hráčmi sa občas schádzajú za účelom získania informácií či nápadov, ktoré by mohli v hernom svete vyskúšať, no často je táto informácia irelevantná alebo danú skutočnosť už poznajú. Pri zdieľaní svojich poznatkov nečakajú od ostatných hráčov uznanie. Svoju pozornosť venujú hlavne skúmaniu sveta a o statným cieľom venujú len malú, poprípade žiadnu pozornosť. Medzi základné aktivity týchto hráčov patrí objavovanie histórie, mytológie, tajomstiev sveta, poprípade miest, o ktorých nikto nevie.

2.1.3.3 Socializers

Socializers hrajú hry predovšetkým kvôli sociálnej interakcii s ostatnými hráčmi. Hru vnímajú iba ako miesto, kde dochádza k tejto interakcii. Vzťahy medzi hráčmi sú pre nich veľmi dôležité. Ich aktivita je zameraná na empatiu, sympatizovanie s ostatnými hráčmi či vytváraní uvoľnenej atmosféry a zábavy. Mnohokrát sú títo hráči nútení dostatočne preskúmať herný svet, aby mohli pochopiť, o čom diskutujú ostatní hráči a neskôr sa zapojiť do konverzácie. Zabíjanie využívajú ako odplatu predovšetkým v prípadoch, keď niekto spôsobil nepríjemnú bolesť ich priateľom. Ich hlavným cieľom v hre nie je získavanie bodov, zabíjanie či objavovanie sveta, aj keď si dokážu užívať samotný priebeh hry. Snažia sa hlavne o nadviazanie kontaktu s ostatnými hráčmi a vytvorenie dôvery a priateľstva. Radi organizujú spoločenské aktivity, spoznávajú nových ľudí a získavajú priateľov.

2.1.3.4 Killers

Tento typ hráčov je zameraný na získanie bodov a na zvyšovanie svojej hernej úrovni. Okrem tohto im však ide o zničenie protihráča, ktorému môže spôsobiť úzkosť (čim väčšia je úzkosť, tým väčšiu radosť prežívajú). Najväčšou motiváciou týchto hráčov je získanie dostatočného množstva bodov. Tie slúžia na zvýšenie a umocnenie sily spôsobujúcej zmätok a frustráciu u ostatných hráčov. Rovnako využíva aj skúmanie a objavovanie sveta a nových vecí. Dômyselnými spôsobmi sa potom zameriava na to, ako zabije protihráča. Tento typ hráčov medzi sebou diskutuje o rôznych spôsoboch zabíjania či taktike. Často vyhľadávajú adrenalín a ostatní hráči majú pred nimi rešpekt a boja sa ich.

2.1.3.5 Bartle test

Na základe hráčskej typológie je zostrojený Bartlov test skladajúci sa z 30 otázok, ktoré percentuálne určujú, ako veľmi osoba spadá pod danú typológiu hráča.

Tento test môže byť veľmi praktický pri testovaní vyčlenenej skupiny zákazníkov využívajúcich gamifikovaný systém. Na základe tohto testu je možné pochopiť do akej kategórie hráčskych typov naši zákazníci spadajú a na základe toho upraviť stratégiu gamifikovaného systému.

2.2 Herné mechaniky

Podľa (Zichermann et. al, 2011) sa *„herné mechaniky (mechanics) gamifikovaného systému skladajú zo série nástrojov, ktoré pri správnom použití sľubujú poskytnutie zmysluplnej odozvy od hráčov“*. Pod touto odozvou chápeme estetika (aesthetics) systému. (Hunicke et. al, 2004) popisujú estetika, ako *„želané emocionálne reakcie vyvolané u hráča pri interakcií so systémom“.* Dynamika (dynamics) určuje správanie hráča ako jednotlivca, či ako súčasť väčšej skupiny hráčov, v reakcii na mechaniky systému. Herné mechaniky, estetika a dynamika tvoria takzvané MDA rozhranie, ktoré predstavuje jeden z najčastejšie využívaných rozhraní herného dizajnu. (Hunicke et. al, 2004) predstavili toto rozhranie za účelom zmenšenia priepasti medzi herným dizajnom a vývojom hier a medzi kritikou hry a jej výskumom.

V nasledujúcom texte budú popísané herné mechaniky, ktoré podľa (Zichermann et. al, 2011) patria medzi najdôležitejšie a najpoužívanejšie. Sú to: body, odznaky, rebríčky, levely, úlohy/výzvy a onboarding.

2.2.1 Body

(Werbach et. al, 2015) definujú body ako *„numerickú reprezentáciu herného progresu“*. Pri definovaní bodového systému sa množstvu ľudí vybaví ohodnotenie vo videohrách, či iných aktivitách. Tie sú však dôležitým aspektom taktiež aj pri monitorovaní hráčov gamifikovaného systému. Pri správnom použití môžu slúžiť na povzbudenie súťaživosti medzi hráčmi. Body sa tak stávajú prirodzenou súčasťou a základom každého gamifikovaného systému.

Body môžu byť používané rôznym spôsobom v gamifikovanom prostredí. (Werbach et. al, 2012) popisujú nasledujúcich 6 spôsobov:

- **Body efektívne ukazujú skóre**: toto je typický spôsob, ktorým sú body využívané. Sú užitočným prostriedkom, pri porovnávaní medzi hráčmi (úspešnejší hráč dosahuje väčšie skóre ako menej úspešný hráč). Body môžu taktiež slúžiť na vymedzenie hráčskeho levelu. Napríklad, pri dosiahnutí určitého počtu bodov, je možné získať rôzne ocenenia, odznaky.

- **Body môžu určiť stav výhry gamifikovaného procesu (pokiaľ nejaký obsahujú):** v niektorých prípadoch je možné použiť body na vytvorenie výhernej podmienky, pričom po jej splnení môže byť užívateľ nejakým spôsobom odmenený.
- **Body vytvárajú spojenie medzi pokrokom v hre a vonkajším odmeňovaním:** množstvo gamifikovaných systémov ponúka pri dosiahnutí určitého počtu bodov vo virtuálnom svete rôzne odmeny z reálneho sveta. Napríklad môže spoločnosť ponúknuť program, kde je pri nazbieraní 25 000 bodov užívateľ odmenený zájazdom.
- **Body poskytujú spätnú odozvu:** explicitná a častá spätná odozva je kľúčovým elementov vo väčšine dobrých herných dizajnoch, kde body poskytujú túto odozvu rýchlo a jednoducho. Každý bod poskytuje užívateľovi malý kúsok spätnej odozvy, ktorý hovorí o určitom postupe užívateľa v hre.
- **Body môžu byť externe viditeľné**: body môžu byť viditeľné pre hráčov, ktorí porovnávajú svoje bodové hodnotenie s ostatnými hráčmi.
- **Body poskytujú informácie pre herných dizajnérov**: získané body môžu byť jednoducho uložené a následne použité pri analýze systému.

Podľa (Werbach et. al, 2012) sú body veľmi obmedzené, a preto sa často používajú v spojení s odznakmi.

2.2.2 Odznaky

(Werbach et. al, 2012) definujú odznaky ako *„vizuálnu reprezentáciu úspechu v gamifikovanom procese“*. Jednou z dôležitých vlastností, ktoré odznaky majú, je ich flexibilita. Odznaky majú široké využitie a ich obmedzenosť je definovaná herným dizajnérom, poprípade firemným plánom. Často sa odznaky využívajú na vymedzenie dosiahnutia určitého počtu bodov. Napríklad, užívateľ môže byť

ocenený odznakom pokiaľ využíva gamifikovaný systém každý deň počas jedného týždňa.

(Churchill et. al, 2011) poukazujú na to, že dobre navrhnutý systém odznakov by mal obsahovať nasledujúcich 5 charakteristík, ktoré definujú nasledovne:

- **Stanovenie cieľov:** odznaky sú využívané na stanovenie cieľov a vyzývajú užívateľa, aby tieto ciele dosiahol. Stanovanie cieľa slúži ako efektívna motivácia pre hráča.
- **Inštrukcie:** odznaky môžu odhaľovať inštrukcie, s akými aktivitami sa môže užívateľ stretnúť v gamifikovanom systéme. Užívateľ si na základe preskúmania odznakov vytvára dostačujúci obraz o systéme. Toto preskúmanie môže inšpirovať, motivovať nových užívateľov systému.
- **Reputácia:** odznaky odzrkadľujú užívateľský záujem a akcie, ktoré vykonal v minulosti. Na základe vzhliadnutia dosiahnutých odznakov je možné zistiť pohľad na reputáciu či aktivitu hráča.
- **Status:** odznaky slúžia ako pripomienky a potvrdenie toho, že užívateľ v minulosti dosiahol a prekonal určité výzvy. Užívatelia, ktorí dosiahli veľmi jedinečné a cenné odznaky, môžu byť považovaní za veľmi schopných v očiach druhých hráčov.
- **Identifikácia v rámci skupiny:** odznaky slúžia na identifikovanie skupiny hráčov. Hráči, ktorí dosiahli rovnakú úroveň, cítia identitu s ostatnými hráčmi v skupine.

Okrem sledovania pokroku a dosiahnutých úspechov pomocou bodov a odznakov, má často užívateľ tendenciu porovnávať svoje úspechy s ostatnými hráčmi. Toto porovnanie je možné znázorniť pomocou ďalšej hernej mechaniky nazývanej rebríčky.

2.2.3 Rebríčky

(Werbach et. al, 2015) definujú rebríčky ako *„vizuálne znázornenie hráčskeho pokroku a úspechov v zoradenom poradí v určitej skupine hráčov“*. Hlavný význam

rebríčku je jednoducho znázorniť postavenie hráčov v skupine. Rebríčky môžu slúžiť v gamifikovanom prostredí ako veľmi efektívny motivátor. Na druhej strane so sebou môžu prinášať určitú mieru demotivácie z pohľadu hráčov, ktorí sa márne snažia o dosiahnutie určitého skóre. (Werbach et. al, 2012) poukazujú na to, že *„rebríčky nemusia byť len statickou tabuľkou, ktorá ukazuje skóre, a taktiež nemusia sledovať len jeden atribút"*.

(Zichermann et. al, 2011) predstavujú 2 typy rebríčkov, ktoré sú široko používané.

Prvý typ umiestňuje hráča presne do stredu rebríčka a sprístupňuje mu informácie len o hráčoch, ktorý sú blízko jeho dosiahnutej úrovne. Hráč tak presne vidí, koľko bodov potrebuje získať na predbehnutie hráča v jeho blízkosti. (Zichermann et. al, 2011) poukazujú na to, že *„pokiaľ sa hráč nachádza v rebríčku na prvých priečkach, rebríček by mal túto skutočnosť priamo zobraziť a ukázať hráčovi skutočné poradie".*

Druhý typ rebríčka umožňuje hráčovi zobraziť jeho postavenie v rôznych kategóriách. Napríklad, rebríček môže byť filtrovaný na základe miesta, kde sa hráč nachádza, či na základe priateľov, ktorých má hráč v gamifikovanom systéme.

Pri vytváraní rebríčku je dôležité narábať opatrne so senzitívnymi, poprípade súkromnými informáciami ostatných hráčov. Taktiež je potrebné zamyslieť sa nad atribútmi na základe ktorých bude tento rebríček zostavený.

2.2.4 Levely

(Werbach et. al, 2015) popisujú levely ako *„definované kroky v hráčskom pokroku"*. Level slúži na určenie úrovne, v ktorej sa hráči nachádzajú v určitom čase v gamifikovanom systéme. V niektorých systémoch môžu levely definovať obtiažnosť hry. Podľa (Zichermann et. al, 2011) by mali byť levely logické (jednoduché na pochopenie), rozšíriteľné (pridanie ďalších levelov v budúcnosti) a flexibilné. Okrem toho odporúčajú, aby ich bolo možné testovať a podľa výsledku testov následne upravovať.

(Zichermann et. al, 2011uvádzajú, že v hernom dizajne nie je hráčsky level popísaný lineárne. To znamená, že pre dosiahnutie ďalšieho levelu je nutné vynaložiť viac úsilia, ako pri získaní predošlého levelu.

Pre získanie levelu je potrebné často nazbierať určitý počet bodov, poprípade splnenie iných aktivít. Tento herný mechanizmus je možné využiť ako súčasť motivácie – pri dosiahnutí určitého levelu je možné sprístupniť hráčovi uzamknuté časti hry.

Podľa (Zichermann et. al, 2011) sa v dnešnej dobe stretávame s hrami, ktoré začínajú s ľahkou úrovňou obťažnosti a táto úroveň postupom času narastá. Úroveň obťažnosti hry sa tak zvyšuje s každým získaným levelom, ktorý užívateľ dosiahne.

Pre zobrazenie úrovne levelu užívateľa využívajú niektoré gamifikované systémy takzvaný ukazovateľ postupu (progress bar), ktorý určuje koľko bodov, príp. percent je nutné získať na postup do ďalšieho levelu.

2.2.5 Onboarding

Onboarding (nalodenie) predstavuje proces oboznámenia sa užívateľa s gamifikovaným systémom pri prvom použití. (Werbach et. al, 2015) poukazujú na to, že *„pri návrhu gamifikovaného systému musíme predpokladať že užívateľ nevie, ako s daným systémom narábať, a preto je nutné ho s tým oboznámiť"*. Úspešné gamifikované systémy využívajú onboarding veľmi efektívne, kde pri prvom spustení prevedú užívateľa a ukážu mu dôležité aspekty systému hravou formou.

Podľa (Zichermann et. al, 2011) popisujú, že *„prvá minúta strávená hráčom v systéme je kľúčová, pretože v priebehu tejto minúty sa väčšina hráčov rozhoduje o ďalšom pôsobení v systéme".* Ďalej popisujú, že v priebehu tejto minúty nie je veľa času na podrobné vysvetlenia a pozornosť by mala byť venovaná angažovanosti hráča.

Počas prvej minúty by mal hráč zažiť, dosiahnuť, poprípade získať určitú spätnú väzbu, ktorá by ho vtiahla do systému. Môže to byť napríklad vo forme získania odznaku, levelu, poprípade ďalšie veci, ktoré ho motivujú k aktivite.

2.2.6 Úlohy a výzvy

(Werbach et. al, 2015) vymedzuje úlohy ako *„konkrétne príklady výziev, ktoré sú definované dopredu pre hráčov".* Úlohy majú vo väčšine prípadov dopredu stanovenú odmenu, čo znamená, že hráč vidí, akú hodnotu mu môže splnenie tejto úlohy

priniesť. Pomocou úloh a výziev môžu herní dizajnéri vytvoriť príbeh, ktorý dáva gamifikovanému systému väčší význam.

V typológií hráčov existujú skupiny, ktoré sú hnané dopredu na základe prekonávania prekážok a výziev. Využitie úloh pri týchto skupinách je opodstatnené. Gamifikovaný systém by mal obsahovať dostatok zaujímavých úloh a výziev, ktoré sa hráči usilujú splniť.

3 Architektonické vzory využívané v OS Anrdoid.

Výber správneho architektonického vzoru pri vývoji aplikácie je kľúčový, pretože môže zásadne ovplyvniť budúci vývoj aplikácie.

Prvým vzorom predstaveným v práci je Model-View-Controller, ktorý patrí medzi najstaršie architektonické vzory.

3.1 Model-View-Controller

Muntenescu (2016a) uvádza, že *„logika užívateľského rozhrania (UI) sa v minulosti menila častejšie ako biznis logika a vývojári desktopových, ale aj webových aplikácií hľadali spôsob, ako oddeliť funkcionalitu užívateľského rozhrania od zvyšku aplikácie“*. Vzor Model-View-Controller (MVC) sa stal ich riešením. Tento architektonický vzor pozostáva z 3 častí: model, view, controller.

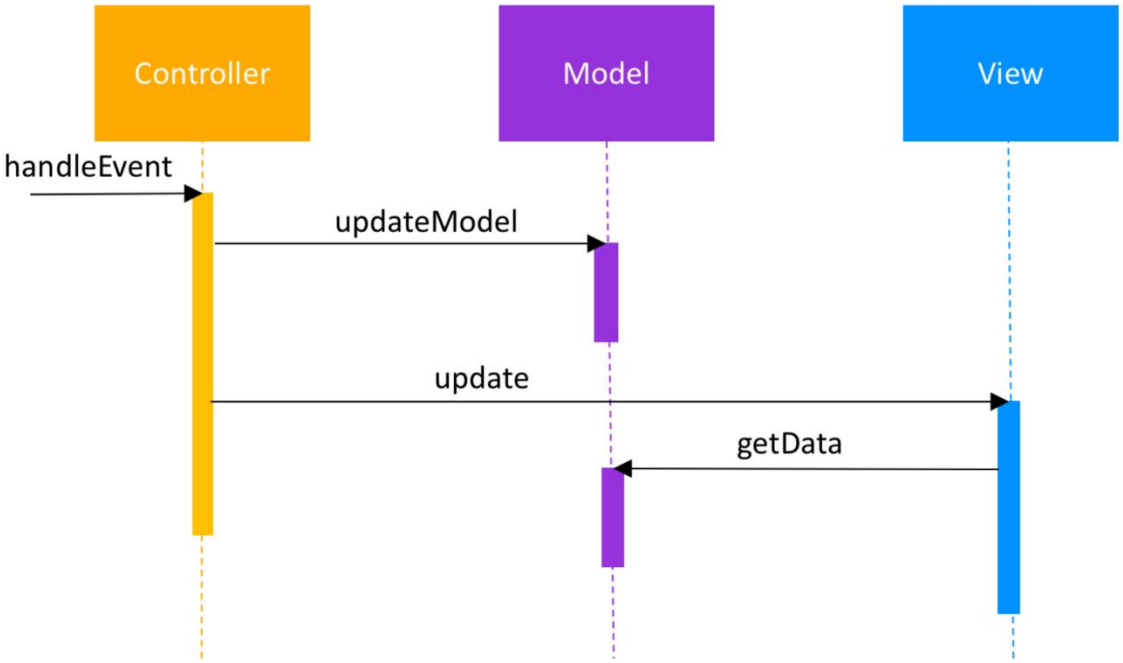
- **Model**: reprezentuje množinu tried, ktoré popisujú biznis logiku a sú taktiež zodpovedné za spracovanie sieťových alebo databázových volaní.
- **View**: je zodpovedný za zobrazovanie dát z modelu.
- **Controller**: je logická vrstva, zodpovedná za spracovanie prichádzajúcich žiadostí. Dostáva informácie o správaní používateľa a podľa potreby aktualizuje model.

V priebehu vývoja sa objavilo niekoľko variant MVC vzoru. Medzi najznámejšie varianty patrí aktívny a pasívny model.

3.1.1 Pasívny model

Muntenescu (2016a) popisuje pasívny model ako *„model, v ktorom je controller jediná trieda, ktorá s ním manipuluje“.* Implementáciu tohto modelu je možné znázorniť za pomoci sekvenčného diagramu.

Na základe užívateľskej akcie controller modifikuje model. Pokiaľ je aktualizovaný, controller upozorní view a ten požiada o dáta z modelu.



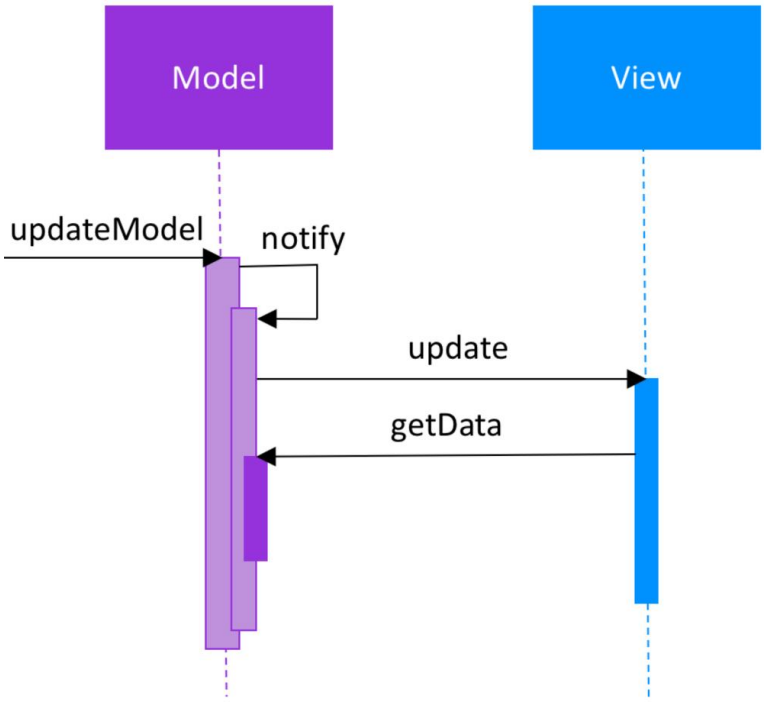
Obrázok 2: Pasívny model v MVC - sekvenční diagram

Prevzaté z Muntenescu (2016a)

3.1.2 Aktívny model

Muntenescu (2016a) poukazuje na to, že *„v aktívnom modeli nie je controller jediná trieda, ktorá modifikuje model, a preto je potrebné nájsť spôsob ako upozorniť view a ďalšie triedy o tom, že bol model aktualizovaný“*. Ďalej predstavuje návrhový vzor observer (pozorovateľ), pomocou ktorého sa dá tento problém vyriešiť. Implementovanie tohto modelu je možné znázorniť pomocou sekvenčného diagramu.

Model obsahuje kolekciu pozorovateľov, ktorí sa zaujímajú o jeho aktualizácie. View implementuje rozhranie pozorovateľa a registruje sa ako pozorovateľ modelu. Zakaždým, keď sa model aktualizuje, iteruje cez kolekciu pozorovateľov a volá metódu `update()`. View následne zavolá požiadavku na získanie aktuálnych dát z modelu.



Obrázok 3: Aktívny model v MVC - sekvenční diagram

Prevzaté z Muntenescu (2016a)

3.1.3 Model-View-Controller v OS Android

Popularita OS Android narastá každý rok a otázky architektúry sa objavujú čoraz častejšie. Muntenescu (2016a) uvádza, že *„MVC bol jeden z najpopulárnejších návrhových vzorov v roku 2011, a preto sa ho vývojári snažili implementovať do operačného systému OS Android“*. Ďalej Poukazuje na prvé pokusy, ktoré navrhovali implementovať triedu `Activity` ako view a controller zároveň. V dnešnej dobre odporúča implementovať triedy `Activity`, `Fragment`, `View` ako view. Controller a model by mali byť separátne triedy, ktoré nededia zo žiadnej android framework triedy.

3.1.3.1 Spracovanie logiky užívateľského rozhrania

Z definície view plynie, že je zodpovedný za zobrazovanie dát, ktoré získa z modelu. Komu patrí úloha rozhodnúť akým spôsobom zobraziť dáta? Pokiaľ je úlohou modelu poskytnúť “originálne” dáta, tak to znamená, že view je zodpovedný za manipuláciu logiky užívateľského rozhrania.

```
String firstName = userModel.getFirstName();
String lastName = userModel.getLastName();
nameTextView.setText(lastName + ", " + firstName)
```

Pokiaľ model sprostredkováva dáta, ktoré musia byť zobrazené a zároveň schováva biznis logiku pred view, tak to znamená, že manipuluje ako s biznis, tak aj s užívateľskou logikou a je tak implicitne závislý na view.

```
String name = userModel.getDisplayName();
nameTextView.setText(name);
```

3.1.3.1.1 Výhody a nevýhody

Podľa Karpouzis (2015) *„MVC podporuje oddelenia zodpovednosti, čo znamená, že model a controller sú oddelené od užívateľského rozhrania“*. Výsledkom tohoto prístupu je jednoduchšie testovanie kódu a údržba aplikácie.

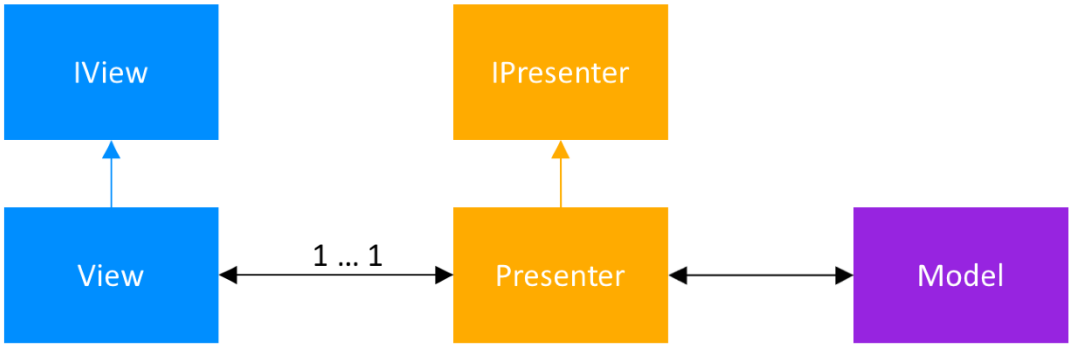
Keďže je view závislý na modeli a controller-u, zmeny v užívateľskom rozhraní môžu vyžadovať aktualizáciu v niekoľkých triedach, čo znižuje flexibilitu tohoto vzoru.

3.2 Model-View-Presenter

Potel (2011) definuje Model-View-Presenter (MVP) ako *„adaptáciu MVC vzoru pre moderné potreby systémov riadených udalosťami (event-driven systems)“*. MVP pozostáva z 3 základných častí: model, view, presenter

- **Model**: dátová vrstva zodpovedná za spracovanie biznis logiky a taktiež za komunikácie s databázovými a sieťovými vrstvami.
- **View**: úlohou tejto vrstvy je zobrazenie dát a upozorňovanie presenter-u na akcie užívateľa.
- **Presenter**: získava dáta z modelu, rozhoduje o tom, čo má zobrazovať.

V MVP vzore je vzťah medzi view a presentérom jasne definovaný. View ako aj presenter obsahujú referencie jeden na druhého, čo umožňuje úzku spoluprácu medzi týmito komponentami. Muntenescu (2016b) odporúča definovať spolupráca pomocou kontrakt interfejsu, ktorý umožňuje lepšie pochopenie interakcií medzi týmito komponentami a sprehľadňuje kód.



Obrázok 4: MVP model - diagram tried

Prevzaté z Muntenescu (2016b)

3.2.1 Model-View-Presenter v OS Android

Vývojári postupom času prichádzali na nedostatky a obmedzenosť MVC vzoru, čo viedlo k pokusom implementovania MVP vzoru.

V nasledujúcom texte sú bližšie predstavené časti MVP vzoru a ich použitie v OS Android.

3.2.1.1 Model

Model predstavuje vrstvu aplikácie, ktorá pracuje s lokálnymi a vzdialenými zdrojmi. Triedy, predstavujúce model obsahujú referencie na tieto zdroje a sprostredkovávajú ich dáta. Podľa Muntenescu (2016b) by nemal model dediť žiadne vlastnosti a funkcie z android framework tried, čo umožňuje jeho jednoduché testovanie. Model sa tak stáva vrstvou, ktorá spracováva biznis logiku a obstaráva dáta pre presenter.

3.2.1.2 View

Úlohou view vrstvy je zobrazovať dáta a upozorňovať presenter o užívateľských akciách vo view. View je v tomto vzore zastúpené triedami `Activity`, `Fragment`, `View`. Muntenescu (2016b) odporúča, aby všetky view implementovali rovnaké `BaseView` rozhranie umožňujúce nastaviť presenter a taktiež odporúča, aby každý presenter obsahoval metódy (`subscribe()`, `unsubscribe()`) pomocou ktorých dá view vedieť, že je pripravený alebo nepripravený prijímať aktualizácie.

```
public interface BaseView<T> {
      void setPresenter(T presenter)
}
```

3.2.1.3 Presenter

Úlohou presenter vrstvy je získavať dáta z modelu a rozhodovať o tom, čo zobraziť vo view. To znamená, že presenter obsahuje referenciu ako na view, tak na model. Pri vytvorení inštancie presenter-u sa nastaví view daný presenter. Muntenescu (2016b) definuje spoločné `BasePresenter` rozhranie pre všetky presentre a okrem tohto rozhrania odporúča, aby každý presenter definoval metódy, ktoré odpovedajú akciám používateľa vo view. Tieto metódy definuje v kontrakt interfejsu.

```
public interface BasePresenter {
      void subscribe()
      void unsubscribe()
}
```

```
public interface UserContract {

    interface View extends BaseView<Presenter> {
        void showUserName();
        …
    }

    interface Presenter extends BasePresenter {
        void saveUser(String firstname, String lastname);
        …
    }

}
```

3.2.1.4 Výhody a nevýhody MVP vzoru

MVP podporuje oddelenie zodpovednosti, čo prináša jednoduchšie testovanie a údržbu aplikácie. Pri vývoji aplikácie s malým rozsahom, sa tento prístup môže zdať zbytočne zdĺhavý a komplikovaný

Muntenescu (2016b) poukazuje na nástrahy spojené s presúvaním logiky užívateľského rozhrania do presentru, čo vedie k výsledku, kde presenter obsahuje niekoľko tisíc riadkov kódu a je neprehľadný.

3.3 Model-View-ViewModel

Gossman (2005) definuje MVVM ako *„variant MVC vzoru, ktorý je prispôsobený pre moderné vývojové platformy, kde view je zodpovednosťou skôr dizajnéra ako klasického vývojára“*. MVVM pozostáva z nasledujúcich častí:

- **Model**: biznis vrstva aplikácie.
- **View**: informuje ViewModel o užívateľskom správaní a zobrazuje dáta.
- **ViewModel**: poskytuje dáta relevantné pre view.

Obrázok 5: MVVM model - diagram tried

Prevzaté z Muntenescu (2016c)

3.3.1 Model-View-ViewModel v OS Android

MVVM bol vytvorený za účelom zjednodušenia programovania užívateľských rozhraní riadených udalosťami. Implementácia MVVM sa v porovnaní s MVP zásadne odlišuje. V MVP vzore presenter udržiaval referenciu na view a priamo mu určoval čo zobrazovať. Naproti tomu MVVM vystavuje dáta pomocou ViewModel vrstvy, a preto nepotrebuje udržiavať referenciu na view vo vnútri tejto vrstvy. Muntenescu (2016c) poukazuje na to, že *„tento prístup vedie k odstráneniu všetkých interfejsoch ktoré sa definujú v MVP vzore“*.

V nasledujúcom texte je bližšie popísaná úloha jednotlivý vrstiev MVVM vzoru v androide.

3.3.1.1 Model

Úlohou model vrstvy je získavanie dát z rôznych zdrojov, ako napríklad databázová, sieťova vrstva či zdieľané preferencie a následne tieto dáta vystaviť spotrebiteľom. Táto vrstva je zodpovedná za biznis logiku aplikácie a Muntenescu (2016c) odporúča vytvárať model pre každú funkciu aplikácie. Napríklad úlohou užívateľského modelu je zastrešovať biznis logiku užívateľa a získavať dáta o užívateľovi, ktoré následne sprístupní pozorovateľom.

3.3.1.2 ViewModel

Úlohou ViewModel vrstvy je získavať dáta z modelu, následne na nich aplikovať logiku užívateľského rozhrania a vystaviť ich pre view vrstvu. Muntenescu (2016c) vo svojom článku odporúča, aby každá užívateľská akcia prechádzala cez ViewModel a všetka logika užívateľského rozhrania bola presunutá práve tu.

3.3.1.3 View

View je vrstva, ktorá predstavuje užívateľské rozhranie aplikácie. Primárnou úlohou tejto vrstvy je zobrazovať dáta, ktoré získava z ViewModel-u. Zvyčajne býva zastúpená triedami `Activity`, `Fragment`, alebo `View`.

3.3.1.4Výhody a nevýhody MVVM vzoru

MVVM kombinuje výhody oddelenia zodpovednosti spolu s výhodami data binding. Muntenescu (2016c) poukazuje na to, že *„view je iba konzumentom dát z ViewModel-u, a preto je jednoduché nahradiť rôzne UI elementy s minimálnymi a občas žiadnymi zmenami v ostatných triedach“*.

4 Android Architecture Components

Podľa (Cleron et. al., 2017) je *„android založený na malej stabilnej súdržnej zostave základných primitívov, ktoré umožňujú používať spoločný programovací model naprieč rozličným spektrom zariadení od hodiniek, cez telefóny a tablety až po televízory, autá a ďalšie zariadenia“.* Ďalej poukazujú na to, že tento model dáva vývojárom aplikácií slobodu vnútorného framework-u používaného v aplikácií. To znamená, že vývojári android frameworku sa nemuseli zapájať do diskusií porovnania rôznych architektonických vzorov (frameworkov) ako sú MVC, MVP, MVVM a ďalšie. Aj keď sú silné základy framework-u a sloboda výberu skvelé, android tím dostával čoraz častejšie otázky od vývojárov, ktoré smerovali na správny návrh aplikácie. (Cleron et. al., 2017) vysvetľujú, že práve toto bol podnet k tomu, aby sa zapodievali názormi na skúsenosti s vývojom Android aplikácií od developerov vo vnútri Google, ale aj mimo spoločnosti, a uvedomili si, že niektoré aspekty vývoja aplikácie sú spracované lepšie ako ostatné a došli k riešeniu s nasledujúcimi vlastnosťami:

- **Riešenie správnych problémov**: potreba riešenia problémov, s ktorými sa stretáva každý vývojár a ktoré sú ťažké na realizáciu.
- **Adaptácia s ostatnými**: vytvorenie API, ktoré môže byť postupom času adaptované a je interpolované s ostatnými knižnicami.
- **Jasnejší postoj**: vytvorenie silnejšieho a jasnejšieho postoja k tomu, ako navrhovať aplikácie správnym spôsobom.
- **Škálovateľnosť**: využitie riešení, ktoré sú priemyselnou silou a ktoré sa prispôsobia skutočným požiadavkám reálneho sveta- fungujú v chaotickej zložitosti reality .
- **Dostupnosť**: uľahčenie vývoja aplikácií pre android.

Výsledkom tejto snahy sa stala podrobná dokumentácia správneho návrhu aplikácií, ktorá je postavená na sade android architecture components (komponenty). Medzi tieto komponenty patrí: Lifecycle, LiveData, ViewModel a Room.

Podľa Fujiwara (2017a) nám *„tieto komponenty pomáhajú perzistovať dáta, spravovať životný cyklus, zabraňujú únikom pamäte, predchádzajú písaniu štandardizovanému kódu a robia aplikáciu viac modulárnou“.*

V nasledujúcom texte sú bližšie predstavené jednotlivé android architecture components.

4.1.1 Lifecycle

Každý android vývojár musí rešpektovať životný cyklus aplikácie. Triedy `Activity` a `Fragment` poskytujú základnú sadu šiestich spätných volaní (`onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()`), ktoré dovoľujú ovládať logiku vlastného kódu vo vyššie spomenutých metódach a zabraňujú únikom pamäte. Tieto metódy často obsahujú vlastné listener-e, či mnoho ďalších komponentov, ktoré musí vývojár registrovať a neskôr od registrovať. Aktivity a fragmenty tak často obsahujú v týchto metódach desiatky až stovky riadkov kódu. Pri takomto množstve kódu môže vývojár ľahko zabudnúť na od registrovanie jedného listener-a, poprípade komponenty, čo môže viesť k úniku pamäte a nadmernému využívaniu batérie. Google v roku 2017 na I/O konferencii predstavuje lifecycle-aware komponenty, ktoré sa dokážu vysporiadať so životným cyklom.

Google (2017a) definuje lifecycle-aware komponenty ako *„komponenty, ktoré vykonávajú činnosť v reakcii na zmenu stavu UI komponentov“*.

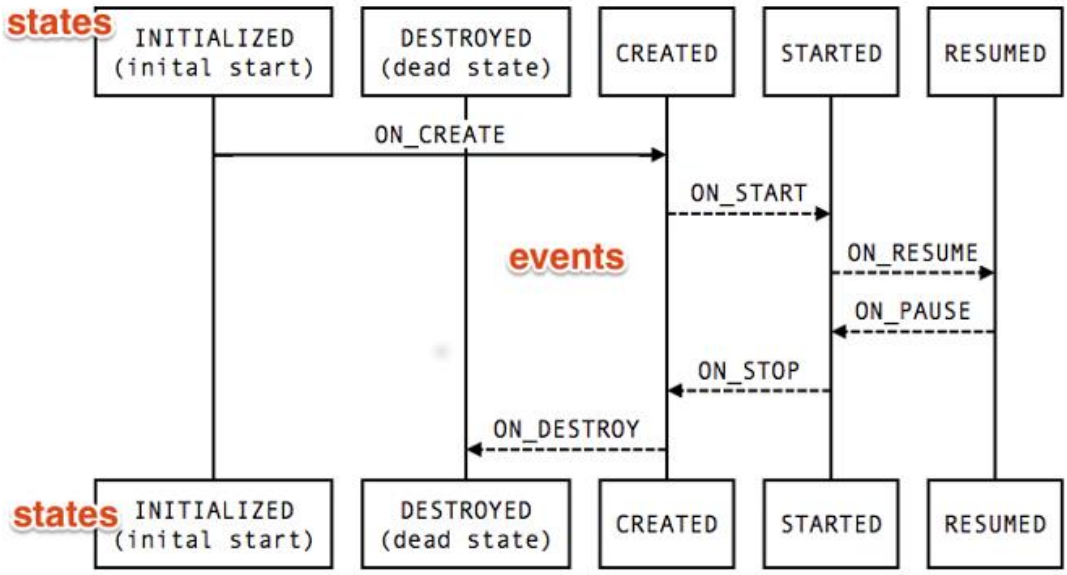
Pre lepšie pochopenie lifecycle-aware komponentov je nutné zoznámiť sa s pojmami: `Lifecycle`, `LifecycleOwner`, `LifecycleObserver`.

4.1.1.1 Lifecycle

Fujiwara (2017b) definuje `Lifecycle` ako *„objekt, ktorý definuje android životný cyklus“*. Tento objekt udržiava informáciu o stave životného cyklu komponentu (napríklad aktivity) a umožňuje ďalším objektom tento stav pozorovať. Používa stavy (states) a udalosti (events) na sledovanie statusu životného cyklu súvisiaceho komponentu.

- **Stav**: predstavuje súčasný stav komponentu.

- **Udalosti**: mapujú udalosti spätného volania v aktivitách a fragmentoch.



Obrázok 6: Stavy a udalosti životného cyklu

Prevzaté z Google(2017b)

Google (2017b) stavy prirovnáva k vrcholom grafu a udalosti k jeho hranám.

4.1.1.2 LifecycleOwner

`LifecycleOwner` je rozhranie s jedinou `getLifecycle()` metódou. Návratová hodnota tejto metódy predstavuje `Lifecycle`. Toto rozhranie abstrahuje vlastníctvo životného cyklu z jednotlivých tried, ako napríklad `Fragment` a `AppCompactActivity`, a umožňuje vytvorenie komponentov, ktoré s týmito cyklami pracujú.

4.1.1.3 LifecycleObserver

`LifecyclerObserver` predstavuje prázdne rozhranie, ktoré umožňuje pozorovať triedy, implementujúce `LifecycleOwner` rozhranie.

Google (2017c) poukazuje na to, že komponenty implementujúce `LifecycleObserver` spolupracujú bezproblémovo s komponentami implementujúcimi `LifecycleOwner` rozhranie, pretože vlastník poskytuje životný cyklus, ktorý pozorovateľ môže sledovať.

4.1.1.4 Tvorba Lifecycle-aware komponentu

Z predchádzajúceho textu plynie, že vývojár môže vytvoriť vlastné komponenty, ktoré budú závislé na životnom cykle UI komponenty.

Trieda implementujúca `LifecycleObserver` rozhranie môže monitorovať stav životného cyklu UI komponenty pridaním anotácie k jej metódam.

```
public class MyOwnObserver implements LifecycleObserver {
    …
   @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
   public void startListening() { … }

   @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
   public void stopListening() { … }

}
```

Každá trieda implementujúca `LifecycleOwner` rozhranie obsahuje `getLifecycle()` metódu, ktorá vracia `Lifecycle` objekt. Tento objekt obsahuje metódu `addObserver()` pomocou ktorej je možné pridať pozorovateľa pre daný životný cyklus.

```
myLifecycleOwner.getLifecycle().addObserver(new MyOwnObserver());
```

4.1.2 ViewModel

Jednou z nástrah prinesených vývojom Android aplikácií je zmena konfigurácie. Medzi akcie prinášajúce zmenu konfigurácie patrí napríklad rotácia telefónu, pri ktorej dochádza k zničeniu aktivity a následne k jej opätovnému vytvoreniu. To prináša radu výhod, ako je napríklad možnosť celkovej zmeny usporiadania grafických elementov na obrazovke na základe režimu výšky či šírky (landscape, portrait mode). Na druhej strane, pri zmene konfigurácie je nutné zaistiť, aby boli dáta správne uložené a následne obnovené. Môže totižto dôjsť k strate dát, neočakávaným chybám a v najhoršom prípade k zrúteniu aplikácie.

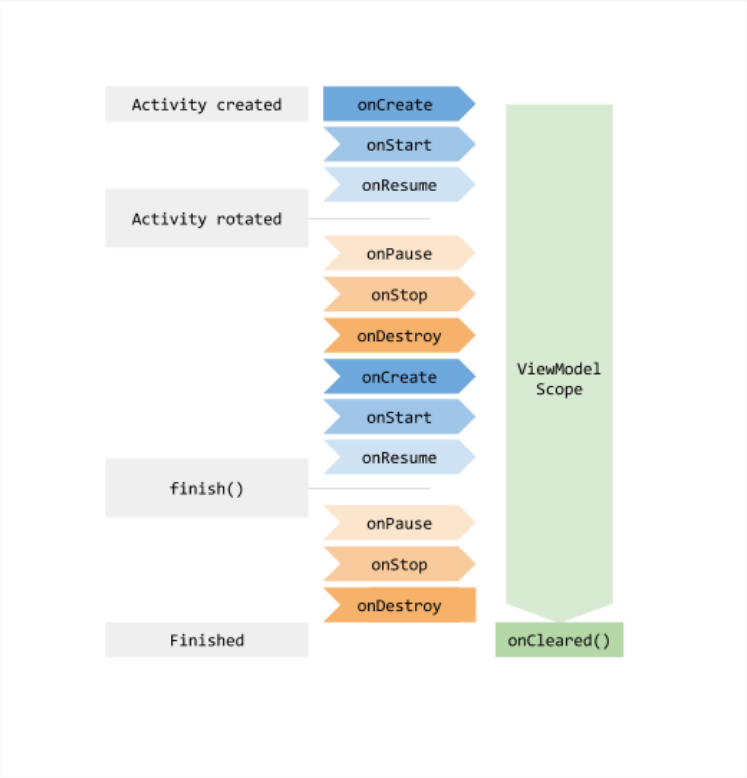
Úskalia, ktoré prináša zmena konfigurácie je možné vyriešiť pomocou `ViewModel`, patriaci medzi Android Architecture komponenty.

Primárne sú aktivity a fragmenty určené na zobrazenie údajov, reagovanie na užívateľské akcie, alebo na komunikáciu s operačným systémom. Google (2017d) neodporúča, aby boli aktivity a fragmenty zodpovedné za načítanie dát z databázy alebo zo siete. Upozorňuje na to, že tento prístup vedie k výsledku, kde je jedna trieda zodpovedná za príliš mnoho akcií a obsahuje tisíce riadkov kódu, ktorý je neprehľadný.

Namiesto ukladania UI dát do aktivít, fragmentov odporúča Fujiwara (2017c) ich vloženie do ViewModelu, čo pomáha vyriešiť problém so zmenou konfigurácie a taktiež je to považované za dobrý software-ový návrh.

Fujiwara (2017c) definuje ViewModel ako *„objekty, ktoré poskytujú dáta pre UI komponenty a dokážu prežiť zmeny konfigurácie".* Hlavnou úlohou ViewModel-u je tak poskytnúť dáta pre tieto komponenty, ktorých úlohou je len vykreslenie týchto dát.

Referencie na objekty `Activity`, `Fragment`, `View` alebo `Context` by nemali byť uložené vo ViewModel-u, pretože ten sa nezaujíma o špecifický životný cyklus aktivity alebo fragmentu. Fujiwara (2017c) upozorňuje na to, že aktivity, či fragmenty môžu byť niekoľkokrát zničené a následne znova vytvorené počas životného cyklu ViewModel-u.



Obrázok 7: ViewModel - životný cyklus

Prevzaté z Google (2017d)

Pokiaľ by ViewModel obsahoval referenciu na aktivitu a dôjde k rotácií obrazovky, aktivita by bola zničená a zároveň by mal na ňu ViewModel uloženú neplatnú referenciu, čo je považované za únik pamäte.

Slúži ViewModel ako náhrada `onSavedInstanceState` metódy? Z textu plynie, že si ViewModel dokáže poradiť so zmenami konfigurácie, no android framework má právo ukončiť proces pokiaľ potrebuje uvoľniť zdroje. ViewModel je v takejto situácií zničený a pri opätovnom vytvorení aktivity znovu vytvorený. Z toho plynie, že nie je schopný uchovať dáta pri zničení aplikácie a preto je nutné využívať naďalej `onSavedInstanceState` metódu. Muntenescu (2017d) odporúča v tejto metóde uložiť malé dáta (napríklad ID užívateľa), tieto dáta pri vytvorení aktivity získať a požiadať ViewModel aby nám poskytol svoje dáta na základe dát (ID užívateľa), ktoré sme získali pri vytvorení aktivity.

4.1.2.1 Tvorba ViewModel komponentu

Prvým krokom pri tvorbe vlastného ViewModel-u je vytvorenie triedy, ktorá dedí metódy a vlastnosti z `ViewModel` triedy. Následne presunúť všetky dáta spojené s UI do tejto komponenty. Fujiwara (2017c) odporúča sprístupniť tieto dáta pomocou prístupových metód, aby nebol porušený princíp zapúzdrenia.

```
public class UserModel extends ViewModel {
    private User mCurrentUser;
    …
    public User getCurrentUser() {
        return mCurrentUser;
    }
}
```

Ďalším krokom je získanie referencie z aktivity, fragmentu na práve vytvorený ViewModel. Toto je možné dosiahnuť pomocou `ViewModelProvider` utility triedy, ktorá obsahuje inštanciu danej aktivity, fragmentu. Fujiwara (2017c) vysvetľuje, že *„práve toto je mechanizmus pomáhajúci získať technicky novú inštanciu triedy pri rotácií telefónu, ale zaručuje, že práve táto inštancia je vždy spojená s rovnakým ViewModel-om.“*

```
public class UserActivity extends AppCompactActivity {
    public void onCreate(Bundle savedInstanceState) {
        …
        UserModel model = ViewModelProviders.of(this).get(UserModel.class);
        User user = model.getCurrentUser();
    }
}
```

ViewModel môže byť použitý na vytvorenie reaktívnych UI (Reactive UI) v spolupráci s ďalším android architecture komponentom nazývaným LiveData.

4.1.3 LiveData

Fujiwara (2017d) definuje LiveData ako *„triedu, ktorá drží dáta (data holder class) hocijakého objektového typu a je si vedomá životného cyklu (lifecycle aware)“*. Ďalej poukazuje na to, že LiveData uchováva iba aktuálny stav dát a nepozná koncept histórie.

Google (2017e) poukazuje na výhody, ktoré so sebou tento komponent prináša:

- **Zabezpečuje, aby UI zodpovedalo stavu dát**: LiveData využívajú návrhový vzor observer a upozorňujú pozorovateľa pri každej zmene stavu životného cyklu. Namiesto ručnej aktualizácie UI, môže pri každej zmene údajov aplikácie pozorovateľ aktualizovať UI.
- **Žiadne úniky pamäte**: pozorovatelia sú viazaní na `Lifecycle` objekty a upracú po sebe, keď sa odstráni ich priradený životný cyklus.
- **Žiadne pády z dôvodu zastavených aktivít**: ak je životný cyklus pozorovateľa neaktívny, tak neprijíma žiadne LiveData udalosti.
- **Žiadne ručné spravovanie životného cyklu:** UI komponenty pozorujú dáta a LiveData automaticky spravuje všetky nástrahy spojené s pozorovaním, ako je napríklad pozastavenie a následné pokračovanie pozorovania. Toto je dosiahnuté za pomoci poznania životného cyklu UI komponenty.
- **Vždy aktuálne údaje:** aktivita, ktorá bola na pozadí dostane najnovšie údaje hneď po návrate do popredia.
- **Správne zmeny konfigurácie:** ak je aktivita alebo fragment znovu vytvorený z dôvodu zmeny konfigurácie, okamžite dostane najnovšie dostupné dáta.

4.1.3.1 Implementácia LiveData

Z predchádzajúceho textu plynie, že LiveData drží dáta objektového typu. Tieto dáta musí byť možné nejakým spôsobom aktualizovať. Pre tieto účely sa používa metóda `postValue()`, ktorá je vykonávaná na UI vlákne a `setValue()` vykonávaná na vlákne v

pozadí. MutableLiveData je potomkom triedy LiveData a verejne sprístupňuje metódy postValue() a setValue(). Na druhej strane, trieda LiveData slúži iba na pozorovanie a nemá tieto metódy verejne prístupné.

Google (2017e) odporúča, aby bol ViewModel jedinou triedou, ktorá priamo aktualizuje LiveData. To znamená, že vo vnútri triedy je odporúčané využívať MutableLiveData inštancie a pre okolité objekty definovať getre, ktoré sprístupnia LiveData na pozorovanie.

```
public class UserViewModel extends ViewMode {
    private MutableLiveData<User> mUser;
    ….
    public LiveData<User> getUser() {
        return mUser;
    }
}
```

LiveData je možné pozorovať za pomoci metódy observe(), ktorá obsahuje dva parametre. Dátový typ prvého parametru je LifecycleOwner, pomocou ktorého je možné získať aktuálny stav životného cyklu UI komponenty. Ako druhý parameter je vložený objekt Observer. Observer je rozhranie s jedinou metódou onChanged(), ktorá je volaná pri každej zmene dát.

Fujiwara (2017d) poukazuje na to, že *„pozorovateľ je spojený so životným cyklom a notifikuje UI komponenty iba v prípade, že sa LifecycleOwner nachádza v aktívnom stave (STARTED alebo RESUMED), čo zabraňuje, aby bol UI komponent aktualizovaný, pokiaľ sa nachádza na pozadí."*

```
public class UserActivity extends AppCompactActivity {
   …
   void onCreate(Bundle savedInstanceState) {
      …
      mViewModel = ViewModelProviders.of(this).get(UserViewModel.class);
      mViewModel.getUser().observe(this,new Observer<User>() {
         @Override
         public void onChanged(User user) {
            updateUserInterface(user);
         }
      });
   }
}
```

4.1.4 Room

Práca s SQLite databázou v androide prináša niekoľko nevýhod a problémov. Muntenescu (2017e) medzi tieto problémy zaraďuje:

- písanie veľkého množstva štandardizovaného kódu;
- nutnosť implementácie mapovanie objektov pre každý dotaz (query);
- náročná implementácia migrácie databáz;
- náročné testovanie databáz;
- vykonanie databázových operácií na hlavnom (UI) vlákne pri zlej implementácií;

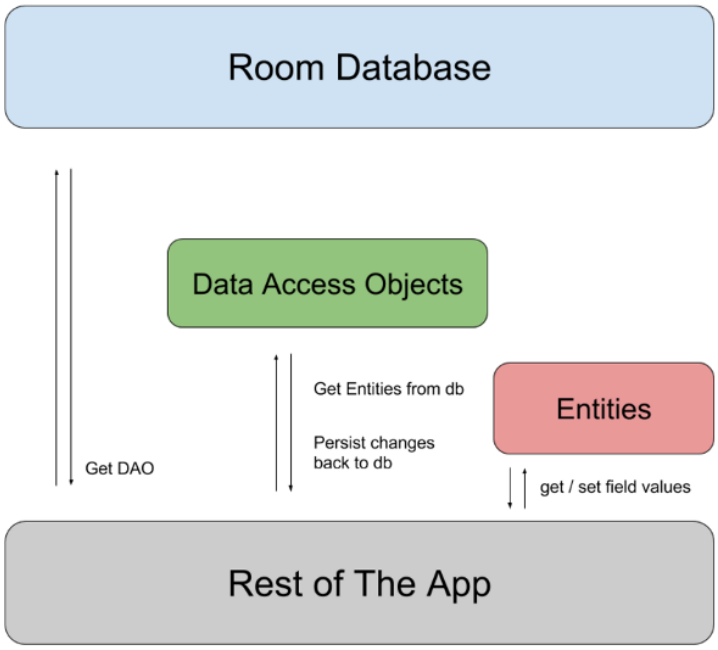
Riešením vyššie spomenutých problémov môže byť použitie Room komponenty.

Muntenescu (2017e) definuje Room ako „*knižnicu, ktorá perzistuje dáta a poskytuje vrstvu abstrakcie nad SQLite“.*

Tento komponent pozostáva z 3 základných častí:

- **Database**: jedná sa o hlavný prístupový bod k perzistovaným relačným údajom databázy.

- **Entity**: reprezentuje databázovú tabuľku.
- **DAO**: obsahuje metódy, ktoré pracujú s databázou.



Obrázok 8: Architektúra Room komponentu

Prevzaté z Google (2017f)

4.1.4.1 Entity

Entity (entita) predstavuje časť Room komponentu, ktorý slúži na definovanie databázových tabuliek.

Pre vytvorenie tabuľky je nutné použiť `@Entity` anotáciu nad deklaráciou triedy. Okrem tejto anotácie, je možné pridať ďalšie anotácie nad názov triedy alebo atribútov, ktoré bližšie špecifikujú elementy a funkcionalitu tabuľky:

- `@Entity(tableName = "")`: nastavenie názvu tabuľky;
- `@ColumnInfo(name = "")`: zmena názvu stĺpca tabuľky;
- `@Ignore`: atribút nebude zahrnutý ako stĺpec tabuľky;
- `@PrimaryKey`: definovanie primárneho kľúča tabuľky;
- `@Index`: nastavenie indexu pre rýchlejšie dotazy;
- `@ForeignKey`: nastavenie cudzieho kľúča tabuľky;

```
@Entity(indicies = {@Index(“name”), @Index(value  = {“last_name”,”address”})})
class User {
   @PrimaryKey
   public int id;
   public String firstname;
   public String address;

   @ColumnInfo(name=”last_name”)
   public String lastname;

   @Ignore
   Bitmap picture;
   …
}

@Entity(foreignKeys = @ForeignKey(entity = User.class, parentColumns = ”id”,
         childColumns = “user_id”))
class Book {
   @PrimaryKey
   public int bookID;
   public String title;

   @ColumnInfo(name=”user_id”)
   public int userID;
}
```

4.1.4.2 DAO

Na získanie dát z databáze je potrebné vytvoriť rozhranie s `@DAO` anotáciou. V tomto rozhraní sa deklarujú metódy a SQL dotazy, ktoré pracujú s databázou a Room sa postará o implementáciou týchto metód. Podporuje dotazy typu `@Insert`, `@Update`, `@Delete` a `@Query`, ktoré sú kontrolované v čase kompilácie, čo zabraňuje vytvorenie neplatných dotazov. Muntenescu (2017e) upozorňuje na to, že všetky dotazy sú realizované na rovnakom vlákne z ktorého boli spustené a nie je možné realizovať dotaz na UI vlákne.

```
@Dao
public interface UserDao {

  @Delete
  public void deleteUsers(User…users);

  @Update
  public void updateUsers(User… users);

  @Insert(onConflict = OnConflictStategy.REPLACE)
  public void insertUsers(User… users);

  @Query("SELECT first_name, last_name FROM users WHERE region IN (:regions)")
  public LiveData<List<User>> loadUsersFromRegions(List<String> regions);

}
```

4.1.4.3 Database

Pri vytvorení vlastnej databáze je nutné definovať abstraktnú triedu, ktorá dedí z `RoomDatabase` a nad deklaráciou triedy uviesť `@Database` anotáciu, ktorá obsahuje všetky entity databáze spolu s jej aktuálnu verziou.

```
@Database(entities = {User.class}, version = 1)
abstract class MyDatabase extends RoomDatabase {
  public abstract UserDao getUserDao();
  ...
}
```

5 Návrh android aplikácie pomocou android architecture components

Typická android aplikácia je postavená z viacerých komponentov vrátane activities, fragments, services, content providers a broadcast receivers. Táto štruktúra je oveľa zložitejšia v porovnaní s desktopovými aplikáciami, ktoré majú vo väčšine prípadov

jediný vstupný bod a bežia ako jediný monolitický proces. Užívatelia mobilných zariadení môžu prepínať medzi rôznymi aplikáciami a tie musia tieto prechody správne zvládať. Vývojár aplikácie tak musí počítať s týmito prechodmi a taktiež si musí byť vedomý toho, že operačný systém môže ich aplikáciu zastaviť v ktoromkoľvek okamihu, pokiaľ potrebuje uvoľniť zdroje pre nové alebo práve spustené aplikácie.

V roku 2017 Google na IO konferencii predstavil príručku k architektúre android aplikácií. Táto príručka pozostáva z postupov a princípov, ktoré Google odporúča dodržiavať pri návrhu android aplikácií.

5.1 Základné architektonické princípy

Pri návrhu aplikácie Google (2017g) odporúča vývojárom zamerať pozornosť na princíp oddelenia zodpovednosti (separation of concerns). Základná chyba, na ktorú upozorňuje, je vkladanie celého kódu do `Activity` a `Fragment` tried. V týchto triedach by sa mal nachádzať iba kód, ktorý pracuje s užívateľským rozhraním alebo s operačným systémom. Poukazuje na to, že dodržanie tohto princípu sa umožňuje vyhnúť mnohým problémom, ktoré súvisia so životným cyklom aplikácie.

Ďalší princíp, ktorý Google (2017g) odporúča dodržiavať je, aby UI získavalo dáta z modelu, v najlepšom prípade z perzistentného modelu. Modely predstavuje ako *„komponenty, ktoré sú zodpovedné za spracovanie dát pre aplikáciu“*. Sú nezávislé na View a na komponentoch aplikácie (activities, services, etc), čo znamená, že sú izolované od životných cyklov týchto komponentov. Využitie perzistovaného modelu so sebou prináša dve hlavné výhody:

- užívatelia nestratia údaje, ak operačný systém zničí aplikáciu;
- aplikácia funguje aj so slabým, poprípade žiadnym pripojením k internetu.

Založenie aplikácie na triedach modelov s presne vymedzenou zodpovednosťou podľa Google(2017g) umožňuje ich jednoduché testovanie a aplikácia sa tak stáva konzistentná.

5.2 Odporúčaná architektúra aplikácie

V tejto sekcii je predstavená štruktúra android aplikácie, ktorá je založená na android architecture komponentoch. Táto štruktúra vychádza z oficiálnej príručky napísanej android framework tímom.

V nasledujúcom texte je predstavený spôsob, akým Google odporúča vytvárať užívateľské rozhranie, získavať dáta z internetu, spravovať závislosti medzi komponentami aplikácie a perzistovať dáta.

5.2.1 Importovanie komponentov

Prvým krokom, ktorý je potrebné vykonať, je importovať android architecture komponenty do projektu. Tieto komponenty sú dostupné z Google Maven zdroja.

Pridanie týchto závislostí umožňuje využívať android architecture komponenty v aplikácií.

5.2.2 Vytvorenie užívateľského rozhrania

Vizuálna štruktúra užívateľského rozhrania je definovaná pomocou layout-u (rozloženia). Tento layout môže byť deklarovaný v OS Android dvoma spôsobmi:

- deklarovanie UI elementov v XML súbore;
- vytvorenie UI elementov programovo;

Výhodou využitia XML súboru je oddelenie prezentačnej vrstvy aplikácie od kódu, ktorý riadi jej správanie.

Po vytvorení UI komponenty (activity, fragment) s jej zodpovedajúcim layout-om je nutné získať dáta a následne ich zobraziť. Google (2017h) odporúča využiť ViewModel, ktorý poskytuje dáta pre tieto komponenty a je zodpovedný za komunikáciu s biznis logikou aplikácie.

Vytvorenú aktivitu a ViewModel je potrebné prepojiť a nájsť spôsob ako informovať aktivitu o nastávajúcich zmenách vo ViewModel-i. Google(2017h) pre tieto účely odporúča využiť LiveData komponent. Ďalej odporúča vývojárom, ktorí používajú

knižnice ako RxJava alebo Agera namiesto LiveData, aby zamerali pozornosť na správne zachádzanie so životným cyklom aplikácie.

5.2.3 Získavanie dát z internetu

Google (2017h) odporúča pre komunikáciu s back-endom (poskytujúcim REST API) použiť Retrofit knižnicu alebo hocijakú inú knižnicu, ktorá slúži rovnakému účelu.

ViewModel môže priamo komunikovať s touto knižnicou, ktorá mu poskytuje dáta. Táto implementácia je možná, ale prináša so sebou nevýhody, medzi ktoré Google (2017h) zaraďuje:

- porušenie princípu oddelenia zodpovednosti;
- náročnosť udržiavania aplikácie pri jej náraste;

Naproti tomu odporúča využitie Repository modulov, ktoré sú zodpovedné za spracovanie dát. Je možné ich považovať za sprostredkovateľov dodávajúcich dáta z rôznych zdrojov (cache, perzistované dáta, atď.).

5.2.3.1 Spravovanie závislosti medzi komponentami

Kvôli tomu, aby mohli Repository triedy komunikovať s back-endom, vyžadujú inštanciu triedy, pomocou ktorej je komunikácia vykonávaná. Táto inštancia by mohla byť vytvorená vo vnútri každej Repository triedy, ale na vytvorenie je nutné poznať závislosti triedy, čo komplikuje kód a predstavuje veľkú záťaž na zdroje. (každá Repository trieda potrebuje poznať závislosti na vytvorenie komunikačnej inštancie).

Google (2017h) poukazuje na to, že existujú dva vzory, ktoré môžu byť použité na vyriešenie tohto problému:

- **Dependency Injection**: (vkladanie závislostí) umožňuje triedam definovať ich závislosti bez toho, aby ich skonštruovali. Google(2017h) odporúča pre tieto účely využívať Dagger 2 knižnicu.
- **Service locator**: poskytuje registre, kde triedy môžu získať svoje závislosti, namiesto toho, aby ich skonštruovali. Google(2017h)

odporúča použiť Service locator, pokiaľ nie je programátor oboznámený s vkladaním závislostí.

5.2.4 Využitie cache

Pokiaľ Repository modul získava dáta iba z webovej služby (web service), nepredstavuje ideálne riešenie. Google(2017h) poukazuje na to, že toto riešenie nie je vhodné z 2 dôvodov:

- mrhanie šírkou pásma siete (network bandwidth);
- núti užívateľa čakať na dokončenie nového dotazu;

Po získaní dát z webovej služby, by mali byť tieto dáta nejakým spôsobom perzistované. Jedným spôsobom ako tieto dáta perzistovať je využitie cache.

Vytvorený Repository modul tak obsahuje 2 zdroje dát (webovú službu, pamäť cache), čo zabraňuje tomu, aby došlo k mrhaniu šírkou pásma siete a k čakaniu na tie isté dáta, ktoré sa už raz načítali.

5.2.5 Perzistencia dát

Repository, získavajúci dáta z webovej služby a z cache, sa môže javiť ako postačujúce riešenie. Pokiaľ však OS Android ukončil príslušný proces, tak Repository nebude obsahovať žiadne dáta v cache a bude musieť získať znova tie isté dáta zo siete, čo je považované za nie veľmi prívetivú užívateľskú skúsenosť.

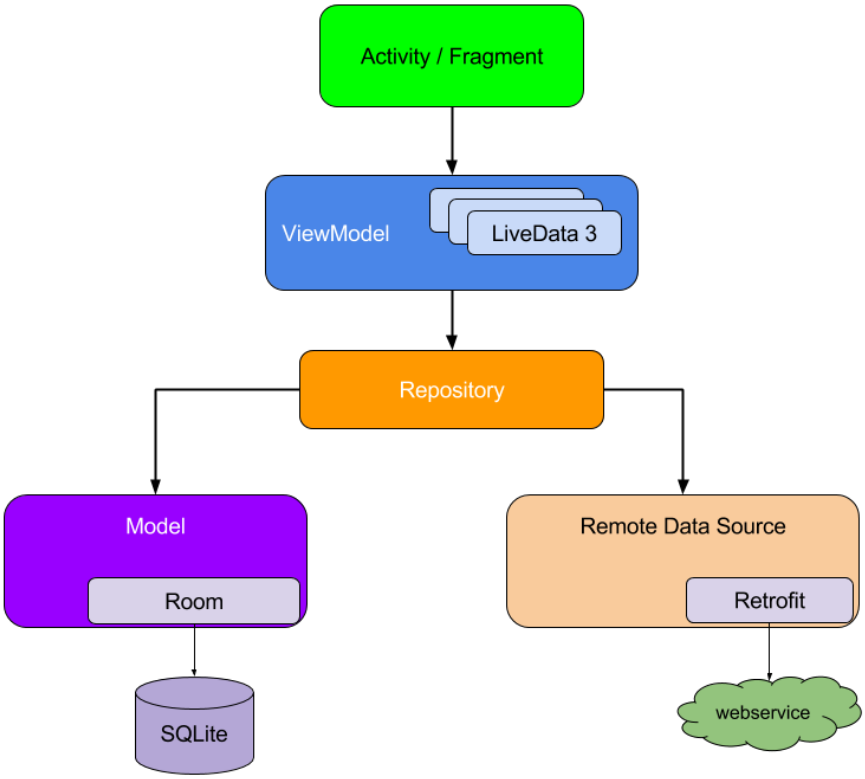
Správny spôsob vyriešenia tohto problému je využitie perzistovaného modelu. Google (2017h) pre tieto účely odporúča využiť Room knižnicu.

Repository tak získa dáta z webovej služby, ktoré sú následne uložené do perzistovaného modelu a sprístupnené pomocou LiveData, čo zaručuje aktualizáciu užívateľského rozhranie zakaždým, keď sa aktualizujú dáta v databáze.

Pokiaľ sú perzistované dáta zastaralé, môže Repository zavolať webovú službu a po získaní dát aktualizovať databázu.

5.2.6 Konečná architektura

V nasledujúcom obrázku sú znázornené všetky vrstvy architektúry, ktoré Google odporúča



Obrázok 9: Konečná architektúra

Prevzaté z Google(2017h)

6 Predstavenie android aplikácie s prvkami gamifikácie

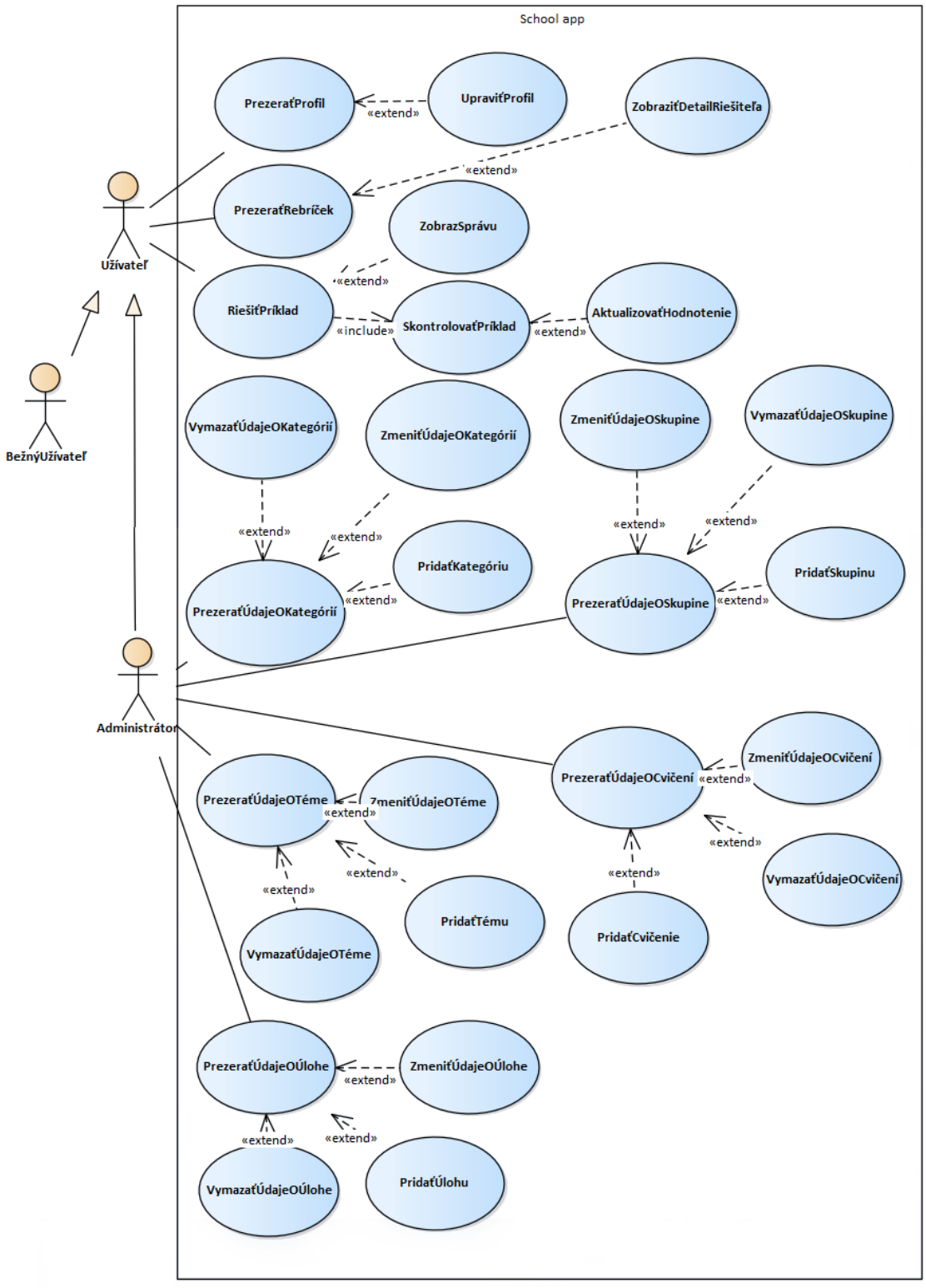
Táto časť je zameraná na predstavenie praktickej časti bakalárskej práce, ktorá sa snaží previesť vyššie spomenutú teóriu gamifikácie do praxe a implementovať aplikáciu na základe postupov a princípov, ktoré odporúča spoločnosť Google. V nasledujúcom texte je popísaný účel aplikácie, využitie gamifikácie v aplikácií a v neposlednom rade, použité technológie a architektúra aplikácie.

6.1 Predstavenie účelu

Aplikácia bola vytvorená za účelom precvičovania príkladov z predmetu matematika pre základné školy. Táto aplikácia môže byť použitá ako podporné médium, kde si môžu žiaci precvičiť látku preberanú v škole.

Aplikácia rozlišuje dva typy užívateľov, ktorí prichádzajú do kontaktu s aplikáciou. Prvý typ užívateľa je administrátor, ktorý má možnosť upravovať, pridávať a mazať obsah hierarchie úloh. Tento typ užívateľa predstavuje kompetentnú osobu (učiteľa na základnej škole), ktorá môže vykonávať spomínané akcie. Okrem toho má administrátor rovnaké možnosti ako bežný užívateľ. Bežný užívateľ prichádza do systému za účelom riešenia a precvičovania úloh, ktoré boli vytvorené administrátorom.

Prehľad funkčných požiadaviek kladených na systém zachytáva diagram prípadu užitia na obrázku 9.

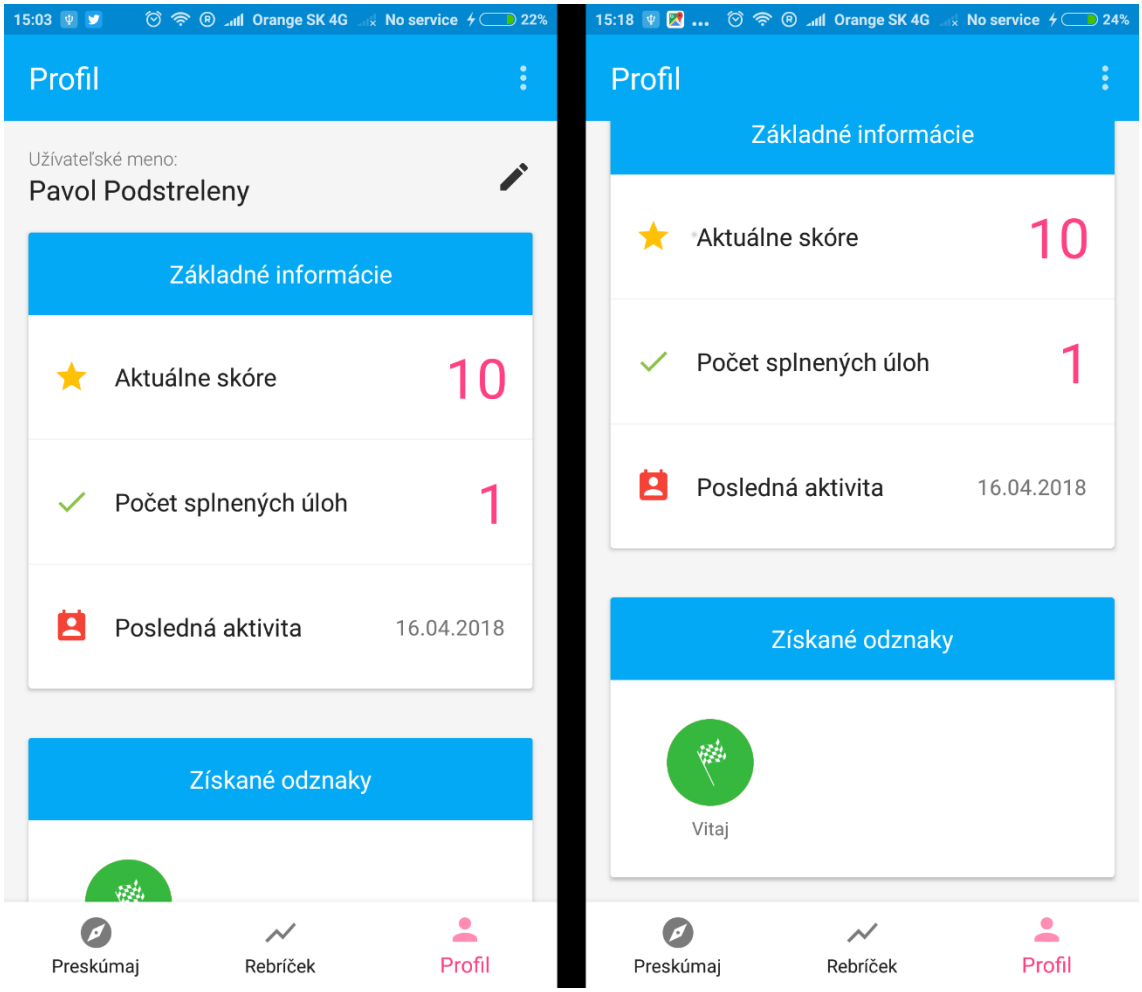


Obrázok 10: Diagram prípadu užitia

Zdroj autor

6.2 Využitie gamifikácie v aplikáci

V aplikácií sú využívané herné mechaniky (body, odznaky, rebríček) slúžiace na povzbudenie záujmu a motivácie užívateľa. Užívateľ získava body za každý správne vyriešený príklad. Pri každom opakovanom vyriešení toho istého príkladu sa bodové hodnotenie znižuje, čo zabraňuje tomu, aby užívateľ riešil dookola ten istý príklad a získaval rovnaký počet bodov. Po dosiahnutí určitého počtu bodov získava užívateľ automaticky odznak. Tieto odznaky, aktuálne nahraté skóre či počet splnených úloh si môže užívateľ zobraziť v sekcii „Profil“ aplikácie.

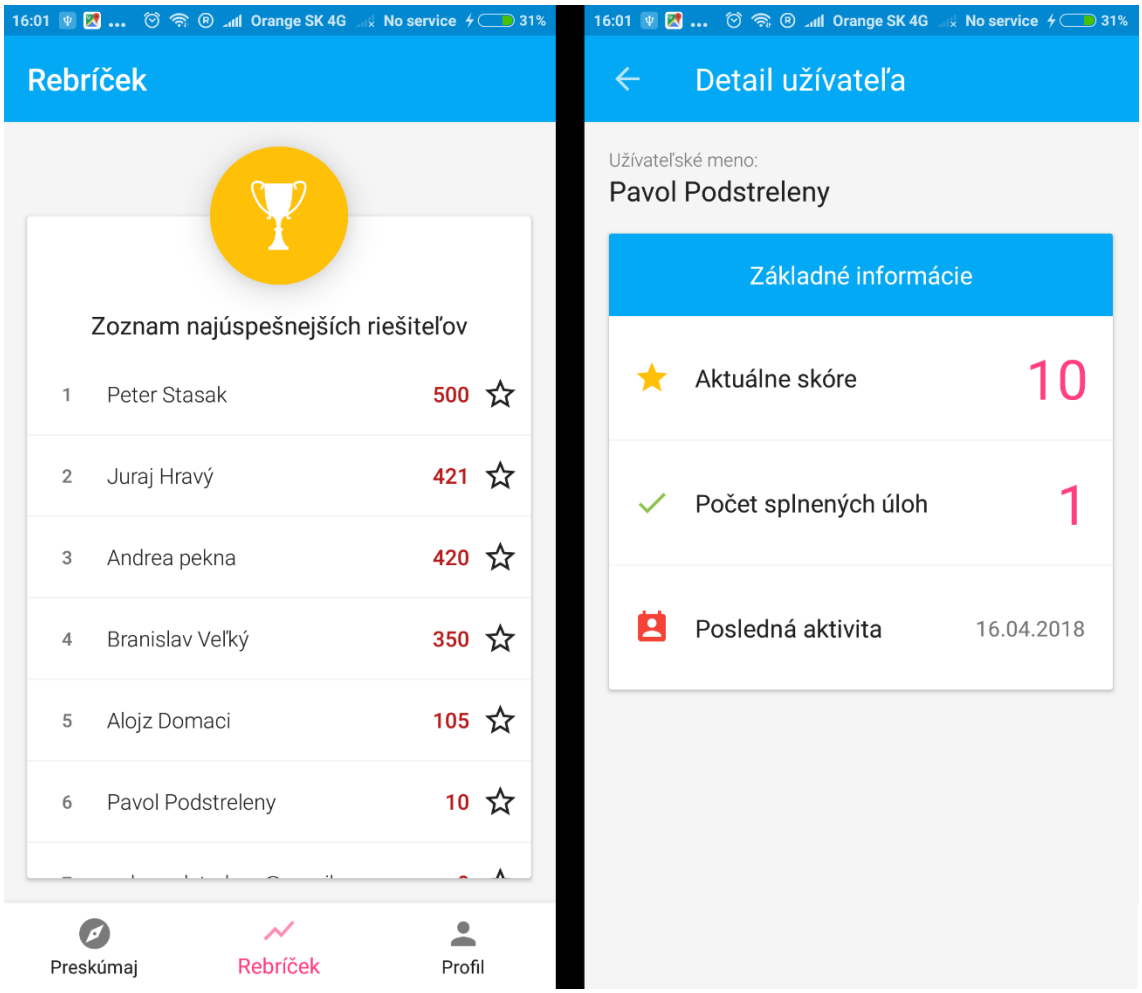


Obrázok 11: Užívateľský profil aplikácie

Zdroj autor

Užívateľ si taktiež môže porovnať svoje aktuálne bodové ohodnotenie v sekcii „Rebríček” s najlepšími riešiteľmi. Táto sekcia zobrazuje rebríček desiatich najúspešnejších riešiteľov zostupne na základe dosiahnutého skóre. Po kliknutí na

užívateľa v danom rebríčku je zobrazený „Detail užívateľa“ s jeho základnými informáciami.



Obrázok 12: Rebríček úspešných riešiteľov a detail užívateľa

Zdroj autor

6.3 Použité technológie

Správne použitie technológií môže mať kladný, ale aj záporný dopad pri vývoji aplikácie. V nasledujúcom texte je popísaná služba Firebase a programovací jazyk Kotlin, ktoré zásadne prispeli k rýchlejšiemu a kvalitnejšiemu vývoju spomínanej aplikácie.

6.3.1 Kotlin

V roku 2017 sa na I/O konferencii stal programovací jazyk Kotlin oficiálne podporovaným jazykom pre vývoj Android aplikácií. Kotlin je staticky typovaný

programovací jazyk zameraný na Java platformu. Medzi jeho najväčšie výhody patrí interoperabilita s Javou, čo zaručuje ich vzájomnú spoluprácu.

Taktiež je kompatibilný so všetkými Java knižnicami a framework-ami. Kotlin je v dnešnej dobe najčastejšie využívaný pri vývoji webových a mobilných aplikácií spúšťaných na Android zariadeniach. Plne podporuje objektovo orientované, ale aj funkcionálne programovanie. Okrem toho so sebou prináša programovanie v tomto jazyku radu výhod, medzi ktoré patrí jednoduchá syntax, čitateľnosť kódu, chytré automatické pretypovanie, predvolené či pomenované argumenty v metódach a mnoho ďalších výhod.

V aplikácií bol Kotlin využitý hlavne kvôli jednoduchej syntaxi, kompatibilite a asynchrónnemu programovaniu, ktoré so sebou prináša.

Pre demonštráciu jednoduchej syntaxe je v nasledujúcom texte definovaná trieda `Person` v programovacom jazyku Kotlin a zároveň v jazyku Java.

```
class Person(var name: String, var age: Int)
```

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age){ this.name = name;  this.age = age;}
    public String getName(){ return name;}
    public void setName(String name){ this.name = name;}
    public int getAge(){ return age; }
    public void setAge(int age){
        this.age = age
    }
}
```

6.3.2 Firebase

Skoro každý vývojár mobilných aplikácií okrem samotného vývoja aplikácie musí vyriešiť problémy spojené s komunikáciou aplikácie so serverom a ukladaním dát do databáze. Pokiaľ chce ukladať dáta o jednotlivých užívateľoch musí vytvoriť bezpečný prihlasovací systém, pomocou ktorého sa užívatelia prihlásia do aplikácie. Neskôr sa môže stretnúť s náhlym nárastom užívateľov, čo znamená zvyšovanie nárokov na škálovateľnosť serverov a databáz. Postupom času je dobré vyhodnocovať užívateľskú angažovanosť, aktivitu a nájsť spôsoby, ako rozširovať užívateľskú základňu. Vývojár sa tak dostáva do situácie, keď musí riešiť vyššie spomenuté problémy a vytvárať rôzne nástroje, ktoré sú veľmi dôležité, ale netýkajú sa samotnej aplikácie.

Jedným z riešením, ako sa vysporiadať s vyššie spomenutými problémami, je využitie Firebase. Firebase poskytuje všetky nástroje pre vývoj úspešných aplikácií. Pomáha získať nových užívateľov, udržiavať ich angažovanosť, napomáha speňaženiu aplikácie a spravuje podrobnosti back-end infraštruktúry. Vývojár tak môže upriamiť pozornosť na vývoj aplikácie a nemusí sa starať o vytváranie nástrojov, ktoré Firebase poskytuje.

Praktická časť bakalárskej práce využíva niekoľko firebase služieb. V nasledujúcom texte sú spomenuté tie najzaujímavejšie, ktoré sa v aplikácií vyskytujú.

6.3.2.1.1 Autentifikácie užívateľa

Firebase poskytuje autentifikačný systém, ktorý je jednoduchý na implementáciu v Android OS a ponúka rôzne metódy a poskytovateľov prihlásenia (Google, Facebook, Twitter, Github, telefón, email/heslo). Týchto poskytovateľov prihlásenia je možné povoliť poprípade zakázať cez webové rozhranie Firebase. Toto rozhranie taktiež poskytuje jednoduchú správu všetkých užívateľov. V praktickej časti bakalárskej práce je využívané prihlásenie cez email/heslo.

6.3.2.1.2 Firebase Realtime Database

V aplikácií je využívaná vzdialená Firebase Realtime Database. Dáta sú v tejto databáze ukladané do JSON-u a synchronizované v reálnom čase so všetkými

pripojenými zariadeniami. Okrem toho je táto databáza optimalizovaná pre offline využitie. Pri strate pripojenia k internetu, SDK databáza využíva lokálne cache na získanie dát a ukladania zmien. Po následnom pripojení k internetu sú lokálne dáta automaticky synchronizované.

Pre ochranu dát sú využívané ochranné pravidlá, ktoré určujú, kto má prístup k daným dátam a taktiež akým spôsobom majú byť dáta štruktúrované.

Obrázok 13: NoSQL Databáza aplikácie

Zdroj autor

6.3.2.1.3 Firebase storage

Ďalšou službou využitou v aplikácii je Firebase storage (úložisko). Toto úložisko slúži na ukladanie obrázkov pre odznaky, kategórie či ostatné dynamicky generované obrázky.

6.3.2.1.4 Cloud functions for Firebase

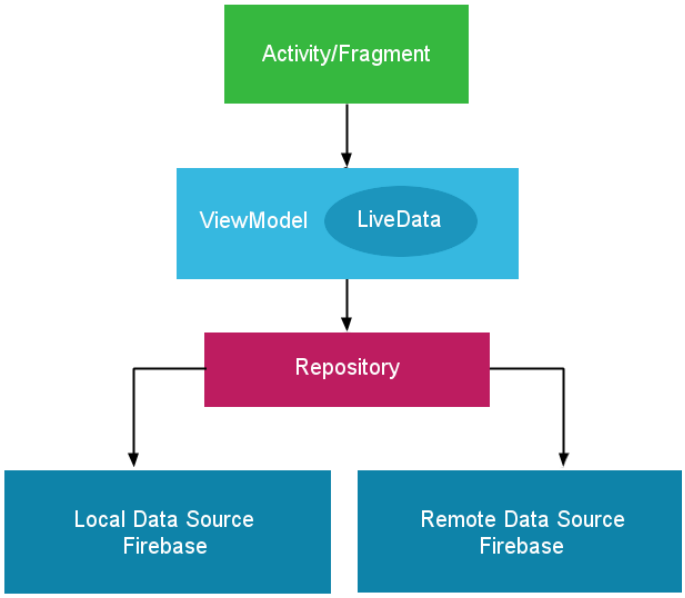
Poskytujú možnosť definovania vlastných funkcií, ktoré sú písané pomocou programovacieho jazyka JavaScript pre prostredie Node.js. Tieto funkcie môžu byť následne odoslané na firebase server (pomocou príkazového riadku), kde sú aplikované.

V praktickej časti bakalárskej práce je definovaných niekoľko vlastných funkcií, ktoré reagujú na rôzne udalosti spôsobené zmenou v databáze. V nasledujúcom texte je predstavenie definovanej funkcie uloženia užívateľa do databázy pri vytvorení účtu.

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');

admin.initializeApp();

//CreteUserProfile
exports.createUserProfile = functions.auth.user().onCreate((user) => {
   return admin.database().ref('/users/'+user.uid).set({
      name: user.email,
      email: user.email,
      score: 0
   }).then(
      admin.database().ref('user_badges'+user.uid).update({
        'welcome' : true,
   }));
});
```

6.4 Architektúra aplikácie

Architektúra aplikácie je založená na android architecture components. Každý UI komponent (`Activity`, `Fragment`) aplikácie komunikuje s `ViewModel`-om, ktorý poskytuje pre tieto komponenty dáta a komunikuje s biznis logikou aplikácie. `LiveData` slúži na informovanie UI komponentov o zmenách vo `ViewModel`-i. Repository vrstva predstavuje sprostredkovateľa, ktorý dodáva dáta zo vzdialenej a lokálnej Firebase databáze.



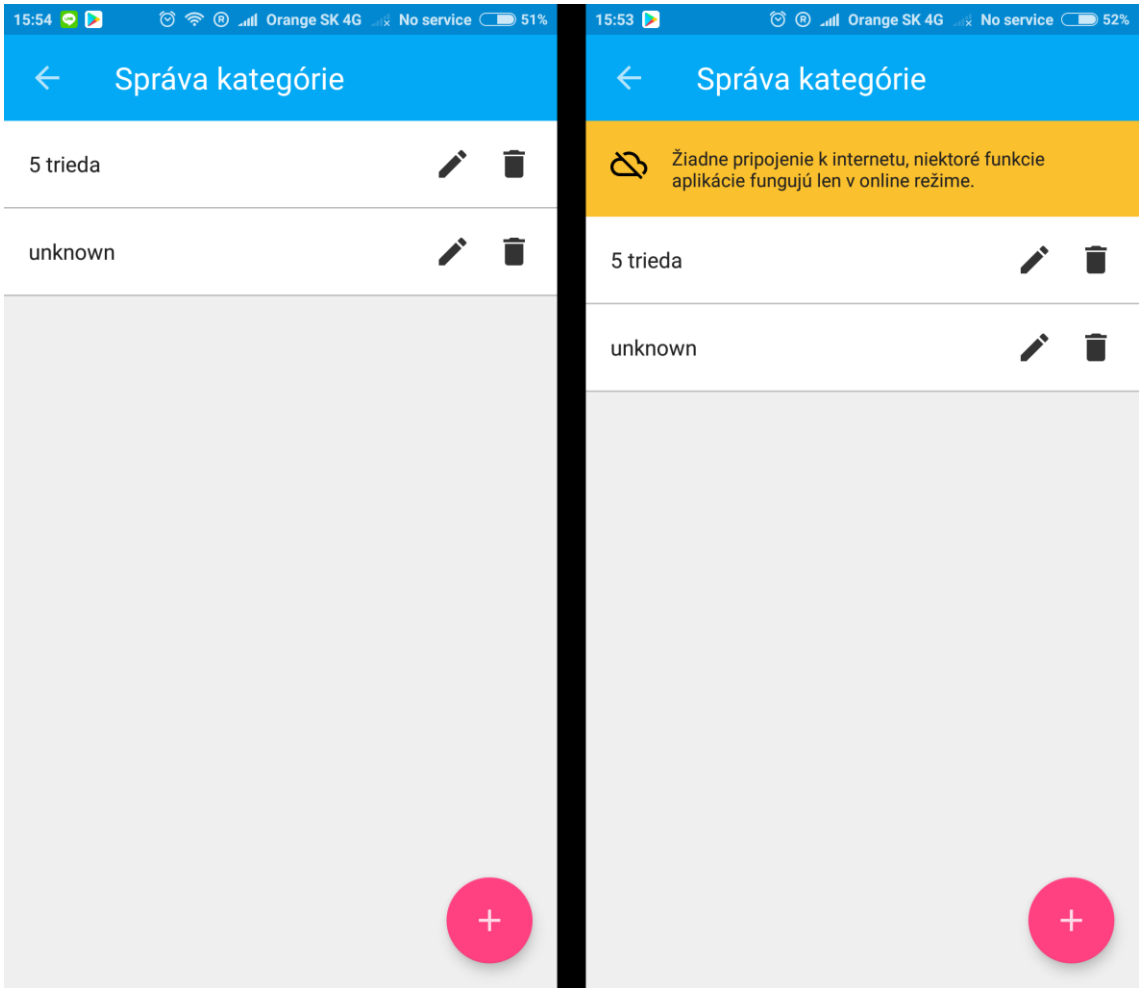
Obrázok 14: Architektúra aplikácie

Zdroj autor

Výhody plynúce z využitia android architecture komponentov sú predstavené v kapitole 4. Tieto komponenty napomohli vyriešiť niekoľko zložitejších problémov v aplikácii . Napríklad za pomoci využitia `LifecycleObserver` rozhrania je v aplikácií definovaný `ConnectionObserver`, ktorý slúži na pozorovanie životného cyklu aktivity či fragmentu. Pri udalostiach `ON_RESUME` registruje a pri `ON_PAUSE` odregistruje `ConnectionReceiver` predstavujúci broadcast receiver slúžiaci na monitorovanie zmien pripojenia k internetu.

```
class ConnectionObserver : LifecycleObserver {
    private val filter = "android.net.conn.CONNECTIVITY_CHANGE"
    private val receiver: ConnectionReceiver
    private val lifecycleOwner: LifecycleOwner
    private val context: Context

    constructor(receiver: ConnectionReceiver,
                lifecycleOwner: LifecycleOwner,
                context: Context){
        this.receiver = receiver
        this.lifecycleOwner = lifecycleOwner
        this.context = context
        lifecycleOwner.lifecycle.addObserver(this)
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    fun start(){
        lifecycleOwner.lifecycle.addObserver(this)
        context.registerReceiver(receiver, IntentFilter(filter))
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    fun stop(){
        context.unregisterReceiver(receiver)
        lifecycleOwner.lifecycle.removeObserver(this)
    }

}
```

Akonáhle dôjde k zmene pripojenia k internetu, automaticky sa zavolá metóda onReceive, definovaná v ConnectionReceiver, ktorá skontroluje, či došlo k pripojeniu či odpojeniu k internetu a na základe toho zobrazí, poprípade skryje, správu pre užívateľa.

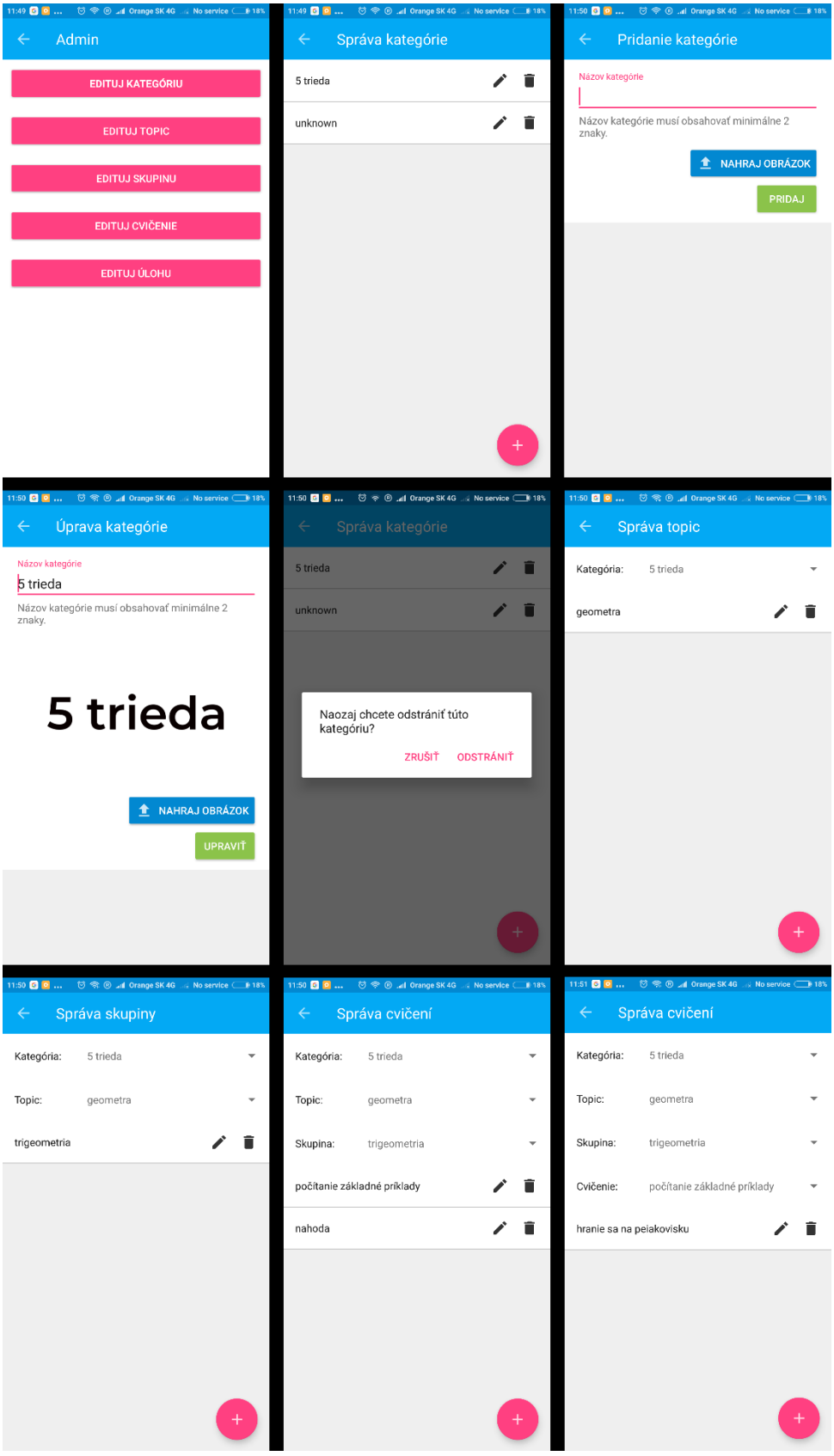


Obrázok 15: Aktivita s pripojením a bez pripojenia k internetu

Zdroj autor

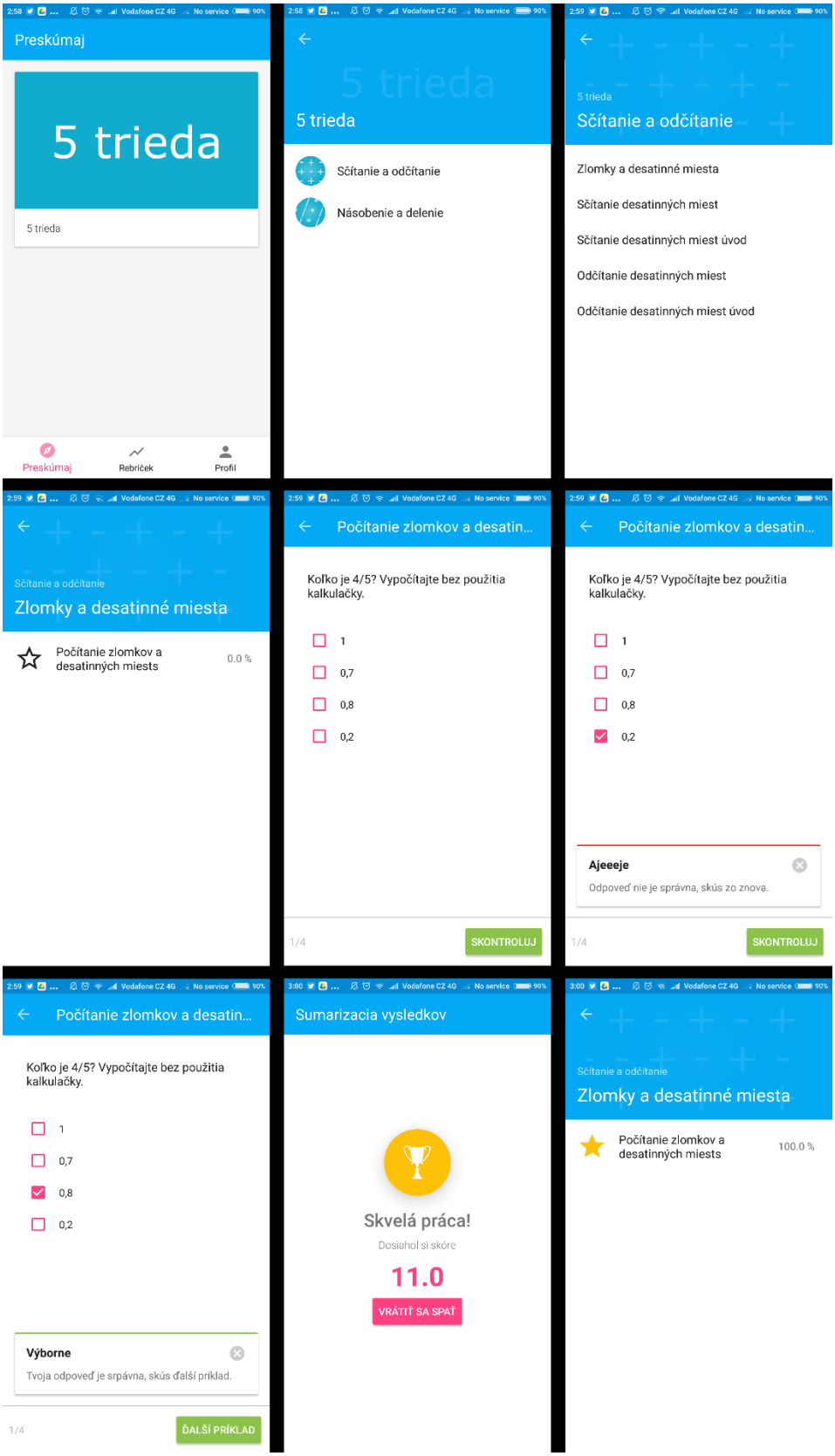
6.5 Zobrazenie aktivít aplikácie

V tejto časti práce je zobrazený zoznam aktivít, s ktorými sa užívateľ môže stretnúť v aplikácii.



Obrázok 16: Administrátorské aktivity

Zdroj autor



Obrázok 17: Aktivity bežného užívateľa

Zdroj autor

7 Zhrnutie výsledkov

Hlavným cieľom bakalárskej práce bolo navrhnúť a implementovať mobilnú aplikáciu s prvkami gamifikácie. Základom vývoja tejto aplikácie bolo prvotné pochopenie teoretickej časti gamifikácie a neskôr aplikovanie získaných informácií do praxe. V teoretickej časti práce v kapitole o gamifikácii bol kladený dôraz na stručné oboznámenie sa s týmto pojmom a taktiež bolo poukázané na fakt, že pre správne aplikovanie gamifikácie do systému je dôležité pochopiť, aká motivácia vedie hráčov k hraniu hier a taktiež poukázať na to, že nároky a očakávanie na gamifikovaný systém sa u ľudí odlišujú. Dôležitou časťou tejto sekcie bolo predstavenie rozdelenia ľudí do štyroch základných skupín hráčov, ktorú definoval Richard Bartl. Toto rozdelenie napomáha k zisteniu, akí zákazníci gamifikovaný systém využívajú a na základe toho reagovať na úpravu tohto systému. V neposlednom rade boli uvedené herné mechaniky, ktoré možno definovať ako sériu nástrojov napomáhajúcich k motivácií a k dosiahnutiu želaných emocionálnych reakcií hráčov pri interakcii s gamifikovaným systémom. Tieto získané teoretické poznatky pomohli pri aplikovaní gamifikovaných elementov v mobilnej aplikácií.

Pri vývoji mobilnej aplikácie hrá veľkú úlohu zvolenie správneho architektonického vzoru. V teoretickej časti práce bola venovaná celá kapitola tejto problematike. Postupne boli predstavené štyri architektonické vzory využívané v OS Android. U každého vzoru bol spomenutý význam a použitie jednotlivých vrstiev. Taktiež bolo poukázané na výhody a nevýhody použitia týchto vzorov. Na základe tohto porovnania bolo rozhodnuté, že architektúra aplikácie bude založená na MVVM architektúre, ktorá bude využívať android architecture components.

Google v roku 2017 na I/O konferencii predstavil oficiálnu príručku k návrhu android aplikácií, ktorá je založená na android architecture components. V teoretickej časti bola venovaná samostatná kapitola tomuto návrhu, ktorá poukazuje na dodržiavanie niekoľkých princípov pri návrhu aplikácie a predstavuje spôsoby riešenia bežných situácií pri vývoji aplikácie. Mobilná aplikácia sa snaží držať princípov a zásad, ktoré sú spomenuté v tejto teoretickej časti.

Posledná časť práce sa venuje popisu účelu tejto aplikácie, využitiu gamifikácie, a v neposlednom rade použitým technológiám, ktoré zásadne uľahčili a zrýchlili vývoj tejto aplikácie.

Výsledkom tejto snahy viedlo k vytvoreniu mobilnej aplikácie s prvkami gamifikácie pre žiakov základných škôl v predmete matematika.

8 Závery a odporúčania

Výsledkom tejto práce bolo vytvorenie mobilnej aplikácie s prvkami gamifikácie pre základnú školu v predmete matematika. Vývoj tejto aplikácie vychádzal zo základov gamifikácie, architektúry a návrhu aplikácie, ktoré boli predstavené v teoretickej časti bakalárskej práce. Veľký vplyv na vývoj aplikácie malo použitie android architecture components a príručka správneho návrhu aplikácií, ktoré boli predstavené spoločnosťou Google v roku 2017. Tieto komponenty a príručka zásadne ovplyvnili kvalitu vývoja spomínanej aplikácie.

Mobilná aplikácia síce spĺňa funkčné požiadavky, no na druhej strane nie je možné v tomto štádiu zistiť, či boli prvky gamifikácie správne implementované.

Pre zistenie správnosti by bolo nutné túto aplikáciu predstaviť a používať určitý čas na základnej škole. Na základe tohto by bolo možné skúmať interakciu žiaka so systémom a štatisticky vyhodnotiť úspešnosť tejto aplikácie. Počas skúmania by mohlo dôjsť k predbežnému zozbieraniu a vyhodnoteniu dát, čo by mohlo prispieť a napomôcť k úpravám tohto systému.

Okrem toho, aplikácia obsahuje iba matematické príklady slúžiace na demonštráciu funkčnosti aplikácie. To je spôsobené tým, že za obsah príkladov z predmetu matematika by mal byť pridávaný a upravovaný iba kompetentnou osobou (napríklad učiteľ základnej školy).

Po technickej stránke došlo k úspešnému vývoju aplikácie s prvkami gamifikácie. To však neznamená, že aplikácia bude mať úspech v školskom prostredí a bude dostatočne motivujúca pre žiaka. Tomu je potrebné venovať ďalšie štúdium a výskum.

9 Zoznam použitej literatúry

BARTLE, Richard. 1996. HEARTS, CLUBS, DIAMONDS, SPADES: PLAYERS WHO SUIT MUDS. In: *Richard A. Bartle's consultancy web site.*[online]. 1996 [cit. 2018-04-22]. Dostupné z: http://mud.co.uk/richard/hcds.htm

CHURCHILL, Elizabeth a Judd ANTIN. 2011. Badges in Social Media: A Social Psychological Perspective. In: *Gamification Research Network* [online]. 12. 05. 2011 [cit. 2018-04-22]. Dostupné z: http://gamification-research.org/wp-content/uploads/2011/04/03-Antin-Churchill.pdf

CLERON, Mike, Yigit BOYAR a Lukas BERGSTROM. 2017. Architecture Components - Introduction (Google I/O '17). *Youtube* [online video]. California: Google, 2017, 17 May 2017 [cit. 2018-01-30]. Dostupné z: https://www.youtube.com/watch?v=FrteWKKVyzI

FUJIWARA, Lyla. 2017a. Architecture Components: Improve Your App's Design. *Youtube* [online video]. Google, 2017, 17 May 2017 [cit. 2018-01-30]. Dostupné z: https://www.youtube.com/watch?time_continue=1&v=vOJCrbr144o

FUJIWARA, Lyla. 2017b, Architecture Components: LiveData and Lifecycle. Youtube [online video]. Google, 2017, 10 Nov 2017 [cit. 2018-01-30]. Dostupné z: https://www.youtube.com/watch?v=jCw5ib0r9wg

FUJIWARA, Lyla. 2017c. Architecture Components: ViewModel. *Youtube* [online video]. Google, 2017, 8 Nov 2017 [cit. 2018-01-30]. Dostupné z: https://www.youtube.com/watch?v=c9-057jC1ZA

FUJIWARA, Lyla. 2017d. Droidcon NYC 2017 - ViewModels, LiveData and Lifecycles, oh my!. *Youtube* [online video]. Google, 2017, 31 Oct 2017 [cit. 2018-01-30]. Dostupné z: https://www.youtube.com/watch?v=SlZVYkhoSq8

GOOGLE. 2017a. Handling Lifecycles with Lifecycle-Aware Components. *Android Developers* [online]. Google, 2017 [cit. 2018-01-30]. Dostupné z: https://developer.android.com/topic/libraries/architecture/lifecycle.html

GOOGLE. 2017b. Lifecycle. *Android Developers* [online]. Google, 2017 [cit. 2018-01-30]. Dostupné z: https://developer.android.com/topic/libraries/architecture/lifecycle.html#lc

GOOGLE. 2017c. LifecycleOwner. *Android Developers* [online]. Google, 2017 [cit. 2018-01-30]. Dostupné z: https://developer.android.com/topic/libraries/architecture/lifecycle.html#lco

GOOGLE. 2017d. ViewModel. *Android Developers* [online]. Google, 2017 [cit. 2018-01-30]. Dostupné z: https://developer.android.com/topic/libraries/architecture/viewmodel.html

GOOGLE. 2017e. LiveData. *Android developers* [online]. Google, 2017 [cit. 2018-01-30]. Dostupné z: https://developer.android.com/topic/libraries/architecture/livedata.html

GOOGLE. 2017f. Saving Data Using the Room Persistence Library. *Android Developers* [online]. Google, 2017 [cit. 2018-01-30]. Dostupné z: https://developer.android.com/training/data-storage/room/index.html

GOOGLE. 2017g. Common architectural principles. Android Developers[online]. Google, 2017 [cit 2018-03-28]. Dostupné z: https://developer.android.com/topic/libraries/architecture/guide.html#common_architectural_principles

GOOGLE. 2017h. Recommended app architecture. Android Developers[online]. Google, 2017 [cit 2018-03-28]. Dostupné z: https://developer.android.com/topic/libraries/architecture/guide.html#recommended_app_architecture

GOOGLE DEVELOPERS. 2016. Google I/O 2016 - Keynote. In: *Youtube* [online]. 2016 [cit. 2018-04-22]. Dostupné z: https://youtu.be/862r3XS2YB0?t=4m57s

GOSSMAN, John. 2005. Introduction to Model/View/ViewModel pattern for building WPF apps. *Microsoft* [online]. Redmond: Microsoft, 2005, 8 Oct 2005 [cit. 2018-01-30]. Dostupné z: https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/

HUNICKE, Robin, Marc LEBLANC a Robert ZUBEK. MDA: A Formal Approach to Game Design and Game Research. In: *Northwestern University* [online]. 2004 [cit. 2018-04-22]. Dostupné z: http://www.cs.northwestern.edu/~hunicke/MDA.pdf

KARPOUZIS, Thanos. 2015. Android Architecture. *Medium* [online]. San Francisco: Medium Corporation, 2015, Aug 24 2015 [cit. 2018-01-30]. Dostupné z: https://android.jlelse.eu/android-architecture-2f12e1c7d4db

LAZZARO, Nicole. 2004. Why We Play Games: Four Keys to More Emotion Without Story. In: *XeoDesign* [online]. Oakland, California, 2004 [cit. 2018-04-22]. Dostupné z: https://xeodesign.com/xeodesign_whyweplaygames.pdf

MUNTENESCU, Florina. 2016a. Android Architecture Patterns Part 1: Model-View-Controller. *Medium* [online]. San Francisco: Medium Corporation, 2016, 1 Nov 2016 [cit. 2018-01-30]. Dostupné z: https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6

MUNTENESCU, Florina. 2016b. Android Architecture Patterns Part 2: Model-View-Presenter. *Upday* [online]. Berlin, 2016, 8 Oct 2016 [cit. 2018-01-30]. Dostupné z: https://upday.github.io/blog/model-view-presenter/

MUNTENESCU, Florina. 2016c. Android Architecture Patterns Part 3: Model-View-ViewModel. *Medium* [online]. San Francisco: Medium Corporation, 2016, 4 Nov 2016 [cit. 2018-01-30]. Dostupné z: https://medium.com/upday-devs/android-architecture-patterns-part-3-model-view-viewmodel-e7eeee76b73b

MUNTENESCU, Florina. 2017d. Architecture Components - Use Cases (GDD India '17). *Youtube* [online video]. Google, 2017, 1 Dec 2017 [cit. 2018-01-30]. Dostupné z: https://www.youtube.com/watch?v=4VGmFztUF6g

MUNTENESCU, Florina. 2017e. Architecture Components: Room. *Youtube* [online video]. Google, 2017, 6 Nov 2017 [cit. 2018-01-30]. Dostupné z: https://www.youtube.com/watch?v=H7I3zs-L-1w

POTEL, Mike. 2011. MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. In: *ResearchGate* [online]. 2011 [cit. 2018-04-22]. Dostupné z: https://www.researchgate.net/publication/255616200_MVP_Model-View-Presenter_The_Taligent_Programming_Model_for_C_and_Java_Taligent_Inc

WERBACH, Kevin a Dan HUNTER. 2012. *For the win: how game thinking can revolutionize your business*. Philadelphia, PA: Wharton, 2012. s. 15-120. ISBN 978-1-61363-023-5.

WERBACH, Kevin a Dan HUNTER. 2015. *Mechanics, and Components for the Win*. Philadelphia, PA: Wharton, 2015. s. 15-24. ISBN 978-1-61363-069-3

ZICHERMANN, Gabe a Christopher CUNNINGHAM. 2011. *Gamification by design: implementing game mechanics in web and mobile apps*. Sebastopol, Calif.: O'Reilly Media, 2011. s. 11-70. ISBN 978-1-449-39767-8.

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

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

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Podstrelený Pavol	Chočská 1526/6, Dolný Kubín	I14126

TÉMA ČESKY:

Vývoj mobilní aplikace s prvky gamifikace

TÉMA ANGLICKY:

Development of mobile application with gamification elements

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cieľ bakalárskej práce

Cieľom práce je navrhnúť, implementovať a vyhodnotiť mobilnú aplikáciu s prvkami gamifikace v školskom prostredí. Aplikácia bude zameraná na žiakov základných škôl v predmete matematika.

Osnova

Úvod
Gamifikace
Vývoj mobilných aplikácií
Návrh a implementácii mobilnej aplikácie s prvkami gamifikace
Záver
Zdroje (Literatúra)

SEZNAM DOPORUČENÉ LITERATURY:

Google Scholar
The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education, ISBN-13: 978-1118096345
Gamification by Design Implementing Game Mechanics in Web and Mobile Apps, ISBN-10: 1449397670
The Building for Billions Playbook (for developers): A Companion Guide to The Secrets to App Success on Google Play

Podpis studenta: Podpisový ..... Datum: 25.09.2017

Podpis vedoucího práce: ..... Datum: .....