Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Branislav Pecher, Jozef Mláka, Michal Kováčik,
Pavol Ondrejka a Daniel Novák

KILL-BILLS

Dokumentácia k riadeniu z predmetu Tímový Projekt

Vedúci tímu: Ing. Jakub Šimko, PhD.

Akademický rok: 2017/2018

Obsah

1 Big picture

1.1 Úvod

V každom jednom projekte, na ktorom robí viac ako jeden človek, je dôležité mať definované určité pravidlá. S narastajúcim počtom ľudí táto potreba iba ďalej narastá a rozširuje sa množstvo požiadaviek na riadenie, ktoré musia byť uspokojené. Či už sa jedná o spôsobe komunikácie medzi jednotlivými členmi ako aj mieste, kde takáto komunikácia prebieha alebo o spôsobe robenia prehliadok kódu či jeho verziovania, pravidlá musia byť stanovené.
Náš tím sa skladá z piatich členov a na aplikácii *Kill-Bills* pracujeme už druhý rok. V nasledujúcich častiach dokumentu sa budeme hlavne venovať rôznym pravidlám a metodikám, ktorými sa pri práci riadime.

1.2 Role členov tímu a podiel práce na dokumentácii

Je dôležité hneď na začiatku deklarovať, že tím sa riadi agilne. Konkrétne používame SCRUM. Ako napovedá samotné slovo *agile*, treba byť schopný sa prispôsobiť rôznym situáciám alebo problémom, ktoré sa počas vývoja softvéru môžu vyskytnúť. Každý člen tímu má svoju silnú stránku, a hlavný modul aplikácie, ktorému sa venuje, ale v rámci prejdenia na SCRUM padlo rozhodnutie *neškatuľkovať* jednotlivých členov tímu podľa ich silných stránok ale začať sa riadiť agílnejšie. V praxi to napríklad môže znamenať to, že ak bude treba viacerých vývojárov na jednom module aplikácie, tak ten skúsenejší ich zaučí a budú na danom module robiť viacerí.
Všetci členovia tímu sa podieľali na tvorbe dokumentácie rovnakým dielom.

1.3 Aplikácie manažmentov

Náš tím pracuje na aplikácii *Kill-Bills* už druhý rok a počas tohto obdobia prešiel viacerými zmenami. Najhlavnejšou bol prechod na agílny spôsob vývoja, čo malo dopad na niektoré oblasti manažmentu. V nasledujúcich sekciách si neprejdeme len to, ako sa tím riadi a aké pravidlá uplatňuje pri riadení teraz, ale aj ako sa riadil v minulosti.

1.3.1 Manažment komunikácie

Komunikácia je pri tímových projektoch nesmierne dôležitá. Preto sa pravidlá na komunikovanie v tíme dobre zadefinovali už pri jeho vzniku v rámci medzinárodnej súťaže *Imagine Cup*.
Ako hlavný spôsob komunikácie sa vždy používal *Slack*. Dajú sa v ňom vytvoriť

viaceré kanály komunikácie a integrovať ich s ďalšími aplikáciami, ktoré tím používa. Projekt máme uložený na Gitlabe. Vždy keď je potreba vykonať prehliadku kódu, alebo sa len pridá požiadavka o spojenie čiastkovej funkcionality do celkovej, Slack nás o tom informuje. Poskytuje taktiež možnosť označovať jednotlivé komentáre, ktoré môžu byť pre nás v budúcnosti dôležité a vieme ich neskôr ľahko vyhľadať. V prípade potreby riešiť komplexnejší problém sa môže použiť aj Skype. Výhodou tejto aplikácie veľakrát býva možnosť zdieľania obrazovky. Od prechodu na SCRUM sme začali používať aplikáciu Scrumdesk. Jej primárny účel nie je pomáhať s komunikáciou ako takou, ale vieme si tam pri jednotlivých úlohách nechávať krátke a výstižné komentáre/odkazy.

1.3.2 Manažment zaručenia kvality

Pri projekte, na ktorom pracuje viacero ľudí, treba mať jasne zadefinované kritéria akceptácie jednotlivých inkrementov, aby sa dodržala určitá kvalita inžinierskeho diela.
Členovia tímu musia písať kód čitateľne, dodržiavať stanovené metodiky a konvencie daného jazyka. Kód píšeme tak, aby sa dal ľahko škálovať a testovať. V minulosti nebolo zvykom písať testy, ani si navzájom robiť prehliadky kódu. Avšak to sa zmenilo a testy už začíname písať a prehliadky kódu sa robia vždy, keď sa napíše nová časť programu, ktorú chceme pridať do existujúcej aplikácie.

1.3.3 Manažment plánovania

Pred prejdením na SCRUM nebolo jasne určené, ako plánovať úlohy a ďalšie smerovanie projektu. Dvaja členovia používali Trello, ostatní si plánovali úplne sami. Každému bola pridelená samostatná práca. Jediné, čo sa robilo spolu, boli rozhrania.
SCRUM načrtáva určitý spôsob, ako manažovať plánovanie úloh. Ako tím sa s našim vedúcim pravidelne stretávame a debatujeme o vytváraní nových úloh. Týmto úlohám určujeme prioritu a pomocou metódy *Planning Poker* aj obtiažnosť. Úlohy s najväčšou prioritou sa dostanú do tzv. šprintov - periodických časových úsekov, kde sa dané úlohy pokúšame splniť. Po každom šprinte sa spraví retrospektíva, kde sa debatuje o tom, čo sa robilo dobre, čo zle a s čím treba začať. Na všetky tieto úkony, okrem retrospektívy, využívame aplikáciu Scrumdesk.
V minulosti sa nám nepodarilo dodať viacero úloh. Práve pomocou retrospektívy sme boli schopní identifikovať rôzne dôvody zlyhaní, napríklad sme nezobrali do úvahy odovzdania na iných predmetoch alebo to, že niektoré úlohy vie efektívne plniť iba jeden člen tímu. SCRUM nám pomáha zohľadniť tieto ale aj iné skutočnosti pri výbere úloh do nových šprintov.

1.3.4 Manažment dokumentácie

Počas minulého akademického roku si každý písal dokumentáciu sám, formou návodu na používanie. Do technickej špecifikácie bakalárskej práce sa potom dostala iba vysoko úrovňová abstrakcia toho, ako to celé funguje.
Dokumentácia je dôležitá preto, lebo ľuďom mimo tímu, čo na projekte pracujú, poskytuje jasný a ucelený pohľad na vytváraný softvér. Zároveň pomáha aj samotným členom jasne definovať, čo sa kde deje a aké pravidlá sa uplatňujú napríklad aj pri riadení. Preto sa každý člen tímu podieľa na vzniku tohto dokumentu zároveň počas práce na aplikácii. Jedna dezignovaná osoba potom jednotlivé vytvoréne časti spája do jedného celku a zodpovedá za kvalitu informácií.

1.3.5 Manažment prehliadok kódu

S prejdením na agílny vývoj sa zaviedli aj povinné prehliadky kódu, ktoré dovtedy neboli vykonávané žiadnym členom tímu - všetci sa starali iba o svoj kód. Avšak v rámci zaručenia kvality sú nevyhnutné. Žiadna časť kódu sa nemôže pridať do celku, pokiaľ nie je schválená iným členom tímu. Takto sa efektívne predchádza bugom, ktoré by sa tak mohli nepozorovane dostať na produkciu a zároveň jednotliví členovia tímu získavajú presnejšiu predstavu o fungovaní kódu ostatných. To je veľmi dôležité hlavne pri agile, kde chceme byť schopní v rámci potreby pracovať aj na iných častiach vyvíjaného softvéru.
Ak bola určitá časť kódu vytváraná za pomoci párového programovania, tak takáto explicitná prehliadka kódu sa stáva redundantnou, lebo je implicitne zahrnutá v procese párového programovania.

1.4 Sumarizácie šprintov

Zápisnica z prvého stretnutia

Dátum konania: 21.9.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Agenda

- aké nástroje budeme používať na manažovanie tímu
- diskusia a priblíženie Scrumu

Nástroje na manažovanie tímu

- *Jira* - konfigurácia nie je jednoduchá, nástroje od Atlassianu
    - kamaráti sa s Bitbucketom aj Slackom
- *Wiki* - na zhromažďovanie know how (na technické záležitosti, návody)
- *CID* - continuous integration delivery, automatické testovanie pre nové verzie
- *Slack* - bude sa používať na komunikáciu v rámci tímového projektu

SCRUM

- od budúceho týždňa šprinty
- treba zostaviť Backlog úloh do budúceho týždňa
    - pri opisoch nešetriť slovami
    - popísať funkcionalitu, ale aj čo treba vo vnútry spraviť
        - ako používateľ chcem upgradnuť svoj účet, aby som mal prístup k rozšírenej funkcionalite
- postupne sa budú *stories* vyberať z backlogu, následne sa ohodnotí ich zložitosť
    - stories sa budú dekomponovať na úlohy, ktoré sa popridelujú a budú sa postupne plniť
- šprint sa potom pomenuje
- postupne plynie *burnout chart*, ktorý treba splniť do konca šprintu
- dopredu sa treba dohodnúť, kto čo bude robiť, prípadne aj kedy, aby sme sa neblokovali
- na konci šprintu sa robí vyhodnotenie, kde sa rozhodne, či je naplnené *Definition of Done*
    - treba napísať *DoD*
    - deklaratívne vyhlásenie, nie viac ako pol strany *kvalitatívnych kritérií*
        - musí bežať
        - musí byť zaintegrovaná
        - musí mať napísané testy
        - musí byť predvedená zákazníkovi
    - môže ich byť viac, každá na iný typ takejto story
- každá úloha pozostáva z častí:
    - naimplementovanie
    - napísanie testov
    - code review (niekým iným) pri PR
- po skončení šprintu sa robí *retrospektíva*

    - reflektuje sa na skúsenosti predchádzajúceho šprintu
    - analyzujú sa dôvody zlyhania (hľadanie systémových chýb)
        - prvý kandidát je zlá komunikácia
- architektúra sa zvykne určiť ešte pred spustením SCRUMU

Zápisnica z druhého stretnutia

Dátum konania: 28.9.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher (prostredníctvom Skype)

Agenda

- Revízia product backlogu a tvorba nových user stories
- Scrumdesk namiesto Jiry
- Naučiť sa používať Scrumdesk
- Vytvoriť monorepozitár na gitlabe

Prešli sa všetky body agendy. Do budúceho týždňa sa Daniel naučí používať *Scrumdesk*. Jozef vytvorí monorepozitár. Michal začne pracovať na nastavovaní stagingu. Tím je o týždeň pripravený začať šprintovať.

Zápisnica z tretieho stretnutia

Dátum konania: 5.10.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Agenda

- Napísať generické *Definition of Done* pre vybrané typy stories
- Dobre opísať stories, ktoré sú prvé v poradí
- V prvom rade dokončiť MVP2.0
    - spending report
        - zoznam bločkov
        - itemy
        - ceny
- Vybrať stories, ktoré pôjdu do šprintu a ohodnotiť ich zložitosť
- Spustiť prvý šprint Beatrix Kiddo

Diskusia

- Robiť náhodné code reviews

- Nasadenie novej verzie nie je story!
- Keď sa opisujú chyby, treba zapísať ako sa chyby prejavujú (v čom spočíva), ideálne z pohľadu používateľa (zadám si vzor a on sa nevyparsuje)
- Chyby môžu byť:
  - critical
  - major
  - ...
- Veci treba rozoberať na drobné, nie opraviť chybu & pridať funkcionalitu, ale potom z toho treba urobiť 2 stories
- Nepoužívať slovesné podstatné mená, opäť ideálne z pohĺadu používateľa
- MVP musí zákaznikovi prinášať úžitok, nejde o to aby bol perfektný

- určili sme si 4 úlohy na najbližší šprint
- keď sa story dostane do šprintu, tak sa rozbije na úlohy (nakódiť, spraviť testy, demo, CI, nasadiť)
- potrebujeme zlúčiť repozitáre do jedného, aby sme sa vedeli zlúčiť do konzistentného stavu
  - súčasný stav je nevyhovujúci, lebo sa nevieme synchronizovať a povedať, že čo bola posledná stabilná verzia

Definition of done

- funkcionalita (demo)
- napísanie automatických testov je štandardná súčast story
- rails aplikáciu stačí nasadiť

TODO

- nastaviť staging prostredie
- v TP musíme napisať metodiky pre všetko čo je užitočné mať spísané (ako čo rozbehať, ako čo robiť)

Zápisnica zo 4 stretnutia

Dátum konania: 12.10.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Priebeh stretnutia

- prediskutovanie priebehu šprintu, kto ma s čím problémy, kto na čom robil
- pridanie nových user storis
- preusporiadať backlog, Palo si myslí že nie je dobre usporiadaný
- prediskutovanie nového API - aby sa predišlo veĺkej user story
- nadbehnúť si s odhadmi stories

Zápisnica z 5 stretnutia

Dátum konania: 19.10.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Priebeh stretnutia

- prediskutovanie priebehu šprintu, kto ma s čím problémy, kto na
- čom robil
- pridanie nových user storis
- preusporiadať backlog, Palo si myslí že nie je dobre usporiadaný
- prediskutovanie nového API - aby sa predišlo veĺkej user story
- nadbehnúť si s odhadmi stories

Záznam zo 6 stretnutia

Dátum konania: 26.10.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Agenda

- optimalizácia CI
- ujasniť ako používať waiting znak v scrumdesku
- návrh prezrieť dokument, upraviť - Kill Bills pre média
- metodika na git
- nadbehnutie s backlogom, preusporiadať, ohodnotiť niektoré stories
- pichnúť Kubovi s jeho Darth Vader prekvapkom na MIP (hudba, réžia, 31.10. o 08:00 rano, aula)

Priebeh stretnutia

- zhrnulo sa čo sa spravilo počas uplnynulého týždňa ("daily" scrum)

- rozhodlo sa, že bločky sa nebudú mazať keď sa uživateľ rozhodne terminovať svoj účet
    - nemazať uživateľa, len archivovať (zmeniť nejaký príznak)
- vždy zobraziť dve desatinné miesta pri cenách na portáli
- diskusia o waiting znaku, ako a či ho využívať
    - rozhodlo sa vytvoriť nový stav - *In review*
    - waiting znak bude od teraz znamenať, že sa čaká na ukončenie iného tasku
- diskusia o CI
    - použijeme cache, treba to ešte zanalyzovať
- git pomenovanie vetiev, nedávať *iba* číslo, treba aj doplňujúcu informáciu
    - fix-číslo_story-popis1_popis2_popis3
    - commit message
        - zatiaľ všetko v pohode, budeme riešiť až keď budú problémy
- diskusia o dĺžke code reviews
    - worst case: pauza
- diskusia o Kill Bills pre média
    - napísať autorke spätnú väzbu na článok, posunúť vydanie
    - pridávajú sa do textu komentáre, čo treba zmeniť
    - urobiť *landing page*, informácie o produkte, kde bude napr. link na beta verziu, možnosť registrovať
    - kúpiť doménu
    - vytvorili sme story na vytvorenie landing page-u
        - vytvoriť zberač emailov, zistiť koľko ľudí tam prišlo, nechalo mail, zbierať ďalšie štatistiky o návštevníkoch
- diskusia o backlogu, user stories, ohodnocovanie, preusporiadanie
    - Mišo sa musí ekologicky zbaviť pokazenej stoličky
    - pridali sme nový epic - Administrácia
        - admin účet story
        - chceme vidieť chronologicky pridávanie všetkých bločkov, zoznam používateľov (v nejakej tabuľke),

vedieť sa prekliknúť na ich činnosti

- prediskutovalo sa star wars vystúpenie, naplánované na prednášku MIP

Záznam zo 7 stretnutia

Dátum konania: 2.11.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik (prostredníctvom skype)
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Agenda

- "daily scrum"
- akceptovanie úloh product ownerom
- zhrnutie šprintu
- retrospektíva
- uzavretie šprintu *Bills*
- začatie šprintu

Priebeh stretnutia

- code review nie je iba zhodnotenie kódu, či spĺňa určité náležitosti ako metodiky, ale treba aj otestovať, či to naozauj funguje
- preusporiadanie backlogu

- treba urobiť dokumentáciu, dať na web, aktualizovať
- vybrať také user stories, čo sú nutné na release aplikácie
    - tie čo sa nestihli tento šprint, padanie aplikácie a výber vzoru
    - odkladanie dát, čo dostávame
    - landing page a článok
    - vyriešenie produkcie, aby bola stabilná, že nám nevyprší subscription
        - dá sa kreditka, budeme si platiť
- ohodnocovanie backlogu
- upravenie účtu aby neridirectovalo na landing page, ale nech obrazovka ostane niekde zmysluplnejšie, napríklad na profile
    - zrušiť update? aspoň na MVP
- spojenie taskov *zlepšiť výber vzoru* a *android api*
- Ukladanie všetkých dát
    - pair programming Palo a Mišo
    - zatiaľ iba posielanie tokenu, id usera a timestamp
- v novom šprinte je jedna story ohodnotená 8, s ktorou Jožko začne čo najskôr, je to riziková úloha

Záznam z 8 stretnutia

Dátum konania: 9.11.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Agenda

- daily scrum
- ohodnotenie ďalších user stories v backlogu
- diskusia o android app frontend inkremente, konkrétne o lište vo vyklikávaní bločku
- ukážeme si kde je čo poukladané na serveri
- prejdenie štruktúry dokumentácie
- prediskutovať čo so stagingom a produkciou
- či sa stíha android časť aplikácie

Priebeh stretnutia

- vykonali sme "daily scrum"
- je potrebné mať staging server na testovanie funkcionality

- v android časti aplikácie treba ešte pracovať na UI, možno sa stihne skoro
    - potom sa môže ísť robiť pripojenie na API, záleží od množstva zadaní
- prešla sa štruktúra dokumentácie a pridali sa dva nové manažmenty riadenia:
    - zaručenia kvality
    - plánovania
- dohodlo sa že kto musí dodať akú technickú dokumentáciu
- android lišta - užívateľ si vôbec nemusí všimnúť, že sa lišta môže vysunúť
- riešenia
    - pridať hamburger
    - nejakú šipku, že sa dá lišta rozšíriť
- otestovali sme nové UI na mobilnej časti aplikácie
- treba vytvoriť manuál ako používať aplikáciu
-

Foldre na serveri

- var/public/xml
    - keď OCR vyparsuje image, ide to tam
    - Všetky dáta sú v tvare: timestamp_userId
- var/public/uploads
    - Všetky tiff od Jožа sú tu uložené. Fotka sa volá rovnako ako xml.tiff
- var/public/unchecked_xmls
    - Keď sa prvýkrát označkuje bloček, ide sem
- var/public/marked_xmls

Záznam z 9 stretnutia

Dátum konania: 16.11.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Agenda

- "daily scrum"
- akceptácia úloh product ownerom
- diskusia o padnutej produkcii
- retrospektíva
- uzavretie šprintu *O-Ren Ishii*
- začatie šprintu *Vernita Green*

Priebeh stretnutia

- treba rozbehať produkciu, zistiť prečo padla
- redis nahodený na záložnej databáze
- padlo rozhodnutie na základe odporúčenia biznis mentora vykonať kvalitatívny výskum, za účelom validácie užitočnosti

vyvíjanej aplikácie

Retrospektíva šprintu *Beatrix Kiddo*

Dátum konania: 19.10.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Retrospektíva

S čím pokračovať

- Danovi sa páči programovanie vo dvojici
- Braňovi sa páči udržiavanie up-to-date stavu Scrumdesku
- Danovi sa páči, že Scrum master sa dopytuje počas týždňa ľudí, či s niečím nemajú problém

S čím treba prestať

- Palo navrhuje prestať drobiť pozornosť a skôr sa počas šprintu focusovať na jeden task.
    - dostáva sa to konfliktu so scrum metodikou, kde sa vyberajú stories z vrchu.
- Braňo navrhuje prestať s praktikou robiť viacero taskov.
    - dospeli sme k záveru, že nie s tým úplne prestať, ale skôr

obmedziť také praktiky
- Michal navrhuje prestať podceňovať riziká pri ohodnocovaní stories
- Venovať veľké množstvo času (20 hodín do týždňa)
- Braňo navrhuje posúvať do šprintu obrovské stories

S čím treba začať

- Jozef navrhuje explicitne vymenovať všetky riziká pre story a viacej ich brať do úvahy
  - uvádzať ich do popisu úloh
  - s rizikami je problém, že nie vždy ich môžeme vidieť
- Obrovské stories rozbiť do prototypov, pričom samotnú úlohu treba posunúť do ďalšieho šprintu, alebo to aspoň zvážiť.
- Začať tými najrizikovejšími stories, nie tými bez rizika.
- Dano navrhuje začať písať komentáre, resp. dokumentáciu ku kódu.
  - ako nováčik častokrát nerozumiem v širšej perspektíve tomu, čo sa v kóde deje.
  - buď písať TODO, písať ich sám do legacy kódu a komentovať nový kód
  - spísať nejaké návody ako pridávať features/prezentery, keďže na androide je najviac práce
  - spraviť kôlničku s nahratím komentovaného screencastu + pozrieť si videá
- Lepšie nazývať veci v gite

Retrospektíva šprintu *Bill*

Dátum konania: 2.11.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik (prostredníctvom skype)
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Retrospektíva

S čím pokračovať

- dobre a rýchlo robené code reviews (člen tímu)
- lepšie pomenovávanie vetiev a commitov
- ku každej úlohe Braňo vytvoril novú úlohu - test pre danú úlohu

S čím treba prestať

- dávať veľa user stories jedného typu do šprintu (napr. veľa Androidu)
- fetišizovať progres

S čím treba začať

- vyberať aj ľahšie android tasky, aby sa mohlo viacero členov dostať do androidu

- robiť aj na moduloch iných členov tímu, keď tam bude nevyvážený počet úloh
- pair programming na tom istom tasku, v rámci dohodnutej oblasti, kde sa treba vzdelávať, podľa neskoršej dohody
- pri code review sa snažiť pochopiť, čo kód robí, nie len dodržanie konvencií

Retrospektíva šprintu *O-Ren Ishii*

Dátum konania: 16.11.2017

Zúčastnení:

- Ing. Jakub Šimko, PhD.
- Bc. Jozef Mláka
- Bc. Michal Kováčik
- Bc. Pavol Ondrejka
- Bc. Daniel Novák
- Bc. Branislav Pecher

Retrospektíva

S čím pokračovať

- pokračovanie s párovým programovaním
- dobrá spolupráca (napríklad pri tvorbe dokumnetácie)

S čím treba prestať

- prestať robiť full android šprinty

S čím treba začať

- poctivé aktualizovanie scrumdesku
- keď sa revizitujú staré triedy, treba ich aj rovno aktualizovať (tlmiť technický dlh)
- dávať si väčší pozor na commitované súbory, občas sa commitujú zbytočnosti

1.5 Globálna retrospektíva ZS/LS

1.5.1 S čím pokračovať

- Párové programovanie
- Udržiavanie aplikácie scrumdesk v up-to-date stave
- Scrum master sa počas týždňa dopytuje na postup členov tímu a či nepotrebujú s niečim pomôcť
- Dobre a rýchlo robené prehliadky kódu
- Ku každej user story vytvoriť hneď aj úlohu - test pre danú user story

1.5.2 S čím treba prestať

- Prestať dávať do šprintu user stories ohodnotené zložitosťou 20 alebo viac
- Sústrediť sa radšej na jednu úlohu ako viacero naraz
- Prestať podceňovať rôzne riziká pri ohodnocovaní stories
- Dávať veľa user stories jedného typu/zamerania do šprintu

1.5.3 S čím treba začať

- Explicitne vymenovať a zamýšľať sa nad rizikami spätými s user stories
- V každom šprinte začať s najrizikovejšími stories
- Začať písať dokumentáciu ku kódu postupne
- Zlepšiť pomenovávanie vecí v gite
- Vyberať do šprintov aj ľahšie android user stories, aby sa mohlo viacero členov tímu zaučiť vývoju v androide
- Pri prehliadkach kódu sa snažiť pochopiť čo aj kód robí a potom ho aj otestovať, ak sa to dá

2 Motivačný dokument

Spoločný email: **kill-bills@googlegroups.com**

1. Tím

Náš tím tvoria piati študenti, z ktorých štyria počas predchádzajúceho roku štúdia pod vedením Ing. Jakuba Šimka, PhD. spolupracovali na projekte Kill-Bills a teda radi by sme v našom úsilí pokračovali aj ďalej na predmete Tímový projekt. Keďže však existujú ďalšie oblasti v ktorých by sa vedeli uplatniť aj ďalší ľudia, rozhodli sme sa doplniť náš tím o ďalších členov, ktorí majú predpoklady na to nám výrazne pomôcť posunúť sa s našim projektom.

Všetci členovia nášho tímu majú skúsenosti s prácou na väčších projektoch, ktoré nadobudli v rámci predchádzajúcich zamestnaní a tiež počas doterajšej práce na tomto projekte. Keďže jadro tímu spolu na projekte pracuje už viac ako jeden rok, tak nám nechýbajú základné zručnosti s manažmentom práce v rámci tímu a máme už aj skúsenosti s prezentovaním nášho projektu, vďaka účasti na akciách rôzneho druhu (Imagine Cup, Slovak University Startup Cup, Emerging Young Entrepreneurs).

Skúsenosti s fullstack vývojom webových aplikácií a ich kvalitným testovaní má náš tím hlavne v osobách Branislava Pecher a Michala Kováčika, ktoré nadobudli v rámci predchádzajúcich zamestnaní a predchádzajúcej práce na tomto projekte. Avšak s tvorbou jednoduchých webových rozhraní majú skúsenosti všetci členovia tímu.

Ďalšími skúsenosťami, ktoré by mohli byť prínosné pri nasledujúcej práci na tímovom projekte, sú skúsenosti s vývojom mobilných aplikácií. V tejto doméne je najviac zbehlý Jozef Mláka, ale do určitej miery aj Daniel Novák. Svoju prax nadobudli najmä v predchádzajúcich zamestnaniach a projektoch.

Všetci členovia tímu majú aj skúsenosti s databázami, pričom najväčšie skúsenosti v tejto oblasti má Pavol Ondrejka, najmä s jazykom PL/SQL a relačnými databázami, konkrétne Oracle.

Každý člen tímu nabral bohaté skúsenosti v oblasti, ktorej sa venoval počas práce na tomto projekte v rámci bakalárskeho projektu. K predmetom Ing. štúdia, ktoré majú členovia tímu aktuálne navolené a mali by byť prínosom k práci na projekte, patria hlavne Počítačové videnie, Vizualizácia dát a Pokročilé databázové technológie.

2. Motivácia

Nápad na skenovanie blokových dokladov a následné získavanie štruktúrovaných dát vznikol v rámci medzinárodnej súťaže Imagine Cup. Pôvodným nápadom bolo snímanie zdravotnej dokumentácie a jej prevodu do elektronickej podoby. Od tohoto zámeru sme však upustili, pretože zdravotná dokumentácia je kompilát z dokumentov písaných rukou povestným ”doktorským krasopisom”, pauzovacích papierov z písacieho stroja a rôznych iných neštruktúrovaných dát. Vtedy sme prišli s nápadom, že namiesto zdravotnej karty by sme mohli snímať bločky a následne z nich vytiahnuť štruktúrované dáta a nakoniec nad nimi vykonávať rôzne operácie. Uvedomili sme si, že pokladničné bločky obsahujú pre ľudí zaujímavé dáta, pričom však ich manuálna extrakcia z bločkov je časovo náročná. Vďaka týmto dátam by sme vedeli pomôcť ľudom viacerými spôsobmi.

Prvá z možností, ktorá nás napadla, ako využitie takejto platformy, bola aplikácia na rozpočítavanie výdavkov pre spoločné účty (napr. v baroch či reštauráciách). Používatelia by odfotili bloček a zaslali ho do cloudu, a nazad by dostali zoznam položiek s ich cenovým ohodnotením. Následne by každý účastník v tejto aplikácií jednoducho označil čo patrí jemu, a aplikácia by mu presne vypočítala, koľko má zaplatiť. Keby takouto sumou nedisponoval, môže to za neho zaplatiť iný účastník, a aplikácia si takýto dlh do budúcna zapamätá.

Ďalšie a hlavné využitie sme našli v sledovaní výdavkov. Používateľ namiesto ručného zapisovania všetkých výdavkov (jediný spôsob ako dosiahnuť evidenciu všetkých výdajov a po položkách, pretože nie všetky transakcie sa robia prostredníctvom platobnej karty, a u tých ostatných vidieť len celkovú sumu a nie parciálne výdavky), môže proste odfotiť blokový doklad svojím smartfónom a odoslať ho do cloudu. Potom si bude môcť cez webové rozhranie, alebo mobilnú aplikáciu pozerať správy o výdavkoch za ľubovoľné obdobie, výkyvy a pod. Popri overovaní a prezentovaní nášeho nápadu medzi ľudmi, sme sa dostali k názoru, že takéto zdieľanie výdavkov by bolo použiteľné aj v rámci biznisov.

3. Súčasný stav projektu

V súčasnej dobe je náš projekt vo fáze otvoreného beta testu. Pozostáva z troch častí:

- **Mobilnej aplikácie**
- **Cloudovej platformy Kill-Bills**
- **Webovej aplikácie Kill-Bills** dostupnej tu.

Platforma Kill-Bills, ktorú sme v projekte pre súťaž ImagineCup vytvorili, umožňuje používateľom jednoduchým spôsobom, prostredníctvom aplikácie v mobilnom telefóne, digitalizovať blokové doklady z registračných pokladníc, a ukladať ich ako štruktúrované dáta. Takto štruktúrované dáta sú potom dostupné, prostredníctom HTTP API, aj v našej webovej aplikácií na sledovanie výdavkov.

Aplikácia na sledovanie výdavkov sa, okrem jednoduchej prezentácie digitalizovaných bločkov používateľovi, venuje aj kategorizácií jednotlivých položiek a snaží sa ich priradiť do rozpočtových kapitol ako sú potraviny, oblečenie a iné. Vďaka tomu umožňuje používateľovi podrobný náhľad do jeho výdavkov, vizualizuje pomery výdavkov na jednotlivé kapitoly, a zobrazuje trendy v týchto kapitolách. V konečnom dôsledku mu teda umožňuje optimalizovať výdavky.

4. Plán na nasledujúce obdobie

1. Aplikácia na zdieľanie výdavkov

Keďže platforma umožňuje získavať údaje o jednotlivých položkách, chceme implementovať aplikáciu, ktorá umožní používateľom roztriediť položky z bločka medzi viacerých používateľov. Taký typický prípad použitia je rozpočítanie výdavkov za účet v reštaurácií, alebo bare, medzi konzumentov.

2. Nasadenie mobilnej aplikácie do produkcie

Po dokončení otvoreného beta testu plánujeme nasadiť mobilnú aplikáciu na Google Store pre platformu Android. Služba by bola dostupná pre bežných používateľov na sledovanie výdavkov a prípadné zdieľanie výdavkov, to znamená, že zatiaľ by aplikácia nebola dostupná pre small-business. Týmto počinom by sme získali prvých reálnych používateľov, od ktorých by sme boli schopní získať prvé dáta a podnety, na základe ktorých by sme boli schopní aplikáciu ďalej vyvíjať a zdokonalovať.

3. Zadefinovať ceny na základe trhu

S reálnymi používateľmi by sme boli schopní zistiť zátaž na náš systém a záujem zákazníkov o našu službu Kill-Bills. Taktiež by sme vedeli, ktoré s našich služieb sú najpoužívanejšie ako aj odhaliť tie, ktoré nemajú veľkú perspektívu. S týmito dátami

predpokladáme, že by sme boli schopní odhadnúť nacenenie našich produktov vzhľadom na spomenuté faktory pre bežných používateľov ako aj pre small-businesses.

4. Funkcionalita pre business

KillBills v súčasnosti umožňuje používateľovi nahrávať svoje bločky, sledovať výdavky a triediť ich do preddefinovaných kategorií. To ale nestačí. Aplikácia musí umožniť viacerým používateľom nahrávať bločky do jedného kontajnera (firma, domácnosť) a umožniť rozhodovať o odpisoch do nákladov, prípadne o oprávnenosti nákladu zamestnanca, napríklad pri služobnej ceste.

5. Implementovať API pre business

Firmy používajú rôzne účtovné systémy, ktoré im uľahčujú podnikanie. Takéto systémy častokrát umožnujú import dát z externých zdrojov, ako je excel, alebo XML. Možnosť vytvoriť takéto dokumenty z našich dát v našej aplikácií v klientom požadovanom, a na mieru mu nastavenom formáte, by preto mala byť samozrejmosť. Chceli by sme, aby sme nemuseli pre každého klienta takéto API vytvárať manuálne, ale aby si ho bol schopný vyklikať sám v používateľskom rozhraní.

6. Kompletný redizajn webovej aplikácie

UX testovanie nám odhalilo závažné chyby v dizajne našej webovej aplikácie na sledovanie výdavkov. Najväčšie problémy boli v neintuitívnom ovládaní tabuľky zobrazujúcej výdavky. Aplikácia takisto nebudí veľkú dôveryhodnosť svojím neuhladeným vzhľadom a neresponzívnosťou.

7. Mobile friendly aplikácia na sledovanie výdavkov

Naša prvá verzia aplikácie na sledovanie výdavkov, sa plnohodnotne a pohodlne dá používať iba z počítača. Na to, aby sme expandovali najmä medzi bežných ľudí, to však nepostačuje. Preto je potrebné nadizajnovať a implementovať aj mobilnú verziu, ktorá umožní plnohodnotný zážitok.

8. Bločková lotéria

Keďže aplikácia dokáže vyextrahovať všetky dáta, potrebné pre registráciu bločka do Národnej Bločkovej Lotérie (NBL), môžeme tento potenciál využiť na prilákanie súťažiacich. Týmto používateľom, by sme umožnili jednoducho a bez prepisovania registrovať bločky do NBL.

9. Úprava algoritmu na kategorizáciu položiek

Analyzovať, navrhnúť a implementovať riešenie, ktoré bude odolnejšie voči chybám čítania OCR, resp. odlišnostiam medzi jednotlivými obchodmi. V súčasnosti totiž používame naivné matchovanie, ktoré vyžaduje 100% zhodu reťazcov. V prvej iterácií, by sme chceli vyskúšať istú percentuálnu toleranciu využitím Levenshteinovej vzdialenosti.

10. Prepracovanie a zlepšenie jednotlivých komponentov

Je potrebné vylepšiť jednotlivé komponenty platformy. Parser častokrát nerozpozná už naučený vzor, OCR stále robí vysokú chybovosť, Androidová aplikácia stále nie je stabilná, častokrát mrzne, alebo padá, a webová aplikácia má jednak malú úspešnosť kategorizácie a druhak, veľmi neprehľadné UI.

11. Migrovanie platformy z Azure na inú platformu

Keďže subscription na BizSpark pre Microsoft Azure platformu, je len dočasný, a Azure je relatívne nákladný, budeme musieť platformu migrovať na inú platformu. Momentálne máme na to ešte čas, ale je treba to mať na pamäti a analyzovať existujúce možnosti migrácie (google cloud, aws, vps).

Príloha A

1. **KillBills**
2. **Recommerce**
3. **Zmluvy**
4. **PUB datasets**
5. **Smart parking**
6. **Virtual Assist**
7. **iBazar**
8. **Group**
9. **Vizreal**
10. **EDUvirtual**
11. **look-inside-me**
12. **FIITdo**
13. **Behametrics**
14. **Stressmonitor**
15. **IOT**
16. **ColabUI**
17. **MobUX**
18. **3D UML**
19. **Deep Search**
20. **SDN4futl**
21. **Brayslet2.0**
22. **Ontosec**
23. **Iweb**
24. **3D futbal**
25. **SDWN**
26. **Futuremod**
27. **Invest**

3 Metodiky

Methodology for Writing Code in the Kotlin Language

This methodology contains current coding style in the Kotlin language used by the Kill Bills team.

Naming Style

If in doubt, default to the Java Coding Conventions such as:

- use of camelCase for names (and avoid underscore in names)
- types start with upper case
- methods and properties start with lower case
- use 4 space indentation
- public functions should have documentation such that it appears in Kotlin Doc

Colon

There is a space before colon where colon separates type and supertype and there's no space where colon separaters instance and type:

```
interface Foo<out T : Any> : Bar {
    fun foo(a: Int): T
}
```

Lambdas

In lambda expressions, spaces should be used around the curly braces, as well as around the arrow which separates the parameters from the body. Whenever possible, a lambda should be passed outside of parentheses.

```
list.filter { it > 10 }.map { element -> element * 2
}
```

In lambdas which are short and not nested, it's recommended to use the it convention instead of declaring the parameter explicitly. In nested lambdas with parameters, parameters should be always declared explicitly.

Class header formatting

Classes with a few arguments can be written in a single line:

```
class Person(id: Int, name: String)
```

Classes with longer headers should be formatted so that each primary constructor argument is in a separate line with indentation. Also, the closing parenthesis should be on a new line. If we use inheritance, then the superclass constructor call or list of implemented interfaces should be located on the same line as the parenthesis:

```
class Person(
    id: Int,
    name: String,
```

```
    surname: String
) : Human(id, name) {
    // ...
}
```

For multiple interfaces, the superclass constructor call should be located first and then each interface should be located in a different line:

```
class Person(
    id: Int,
    name: String,
    surname: String
) : Human(id, name),
    KotlinMaker {
    // ...
}
```

Constructor parameters can use either the regular indent or the continuation indent (double the regular indent).

Unit

If a function returns Unit, the return type should be omitted:

```
fun foo() { // ": Unit" is omitted here

}
```

Functions vs Properties

In some cases functions with no arguments might be interchangeable with read-only properties. Although the semantics are similar, there are some stylistic conventions on when to prefer one to another.

Prefer a property over a function when the underlying algorithm:

- does not throw any Exceptions
- has a O(1) complexity
- is cheap to calculate (or cached on the first run)
- return the same result over invocations

Metodika pre vývoj RubyOnRails

Úvod

Cieľom tohto dokumentu je zadefinovanie základných postupov pre jednotnú tvorbu a komentovanie zdrojových kódov. Metodika zahŕňa konvencie písania zdrojového kódu, ktoré sú jeho tvorcovania povinní dodržiavať. Uvedená metodika je zameraná na programovací jazyk ruby a rámec (angl. framework) Ruby on Rails.

Zadefinovanie pojmov

V Tab. 1 sa nachádza zoznam použitých pojmov aj s ich vysvetlením.

Pojem	**Vysvetlenie**
meno	Meno autora (značky, kódu).
správa	Vyjadruje bližší opis značky (odôvodnenie).
snake_case	Kovencia, alebo tiež štýl písania viacslovných názvov, kde každé slovo začína malým písmenom a slová sú navzájom oddelené znakom '_' (podtržníkom).
CamelCase	Konvencia, alebo tiež štýl písania viacslovných názvov, kde každé slovo začína veľkým písmenom a jednotlivé slová nie sú navzájom oddelené.

SCREAMING_SNAKE_CASE	Konvencia, alebo tiež štýl písania viacslovných názvov, kde každé písmeno je písané veľkým písmom a jednotlivé slová sú navzájom oddelené znakom '_' (podtržníkom).

Postupy

Používaným jazykom pri písaní kódu ako aj komentárov je výhradne angličtina. Žiadne iné jazyky nie sú povolené.

Komentovanie a značkovanie zdrojového kódu

Komentáre

- V zdrojovom kóde nie sú povolené žiadne komentáre okrem dokumentačných komentárov. Z tohto dôvodu je potrebné písať kód tak, aby bol jasný na prvý pohľad.
- Každá zakomentovaná časť zdrojového kódu musí mať pri sebe značku (viď nižšie) s odôvodnením.
- Za značkou komentára nechávať jednu medzeru. Príklad:
  - #zle
  - # dobre

Značky

- Značka sa do kódu píše rovnako ako komentár v tomto tvare: {značka} ({meno}) {správa}.
- Značka má byť stručná, ale jasná.
- V správe k značke sa nahradzuje:
- zmazať kód (angl. remove code) skratkou rm,
- presunúť kód (angl. move code) skratkou mv.
- Povolené značky v kóde sú uvedené v Tab. 2.

Značka	Význam
TODO	Potrebné vyriešiť
FIX	Potrebné opraviť

Príklad:

```
# TODO (Pecher) rm
```

```
<!-- FIX (Kovacik) invalid html -->
```

Zdrojové kódy

Zdrojový kód v jazyku HTML

- Vždy musia byť použité " namiesto '.
- Odsadzuje sa dvoma medzerami.
- HTML musí byť ***validné***.
- Pravidlá pre rámec Twitter Bootstrap:
    - Element triedy row môže obsahovať len elementy triedy col.

Zdrojový kód v jazyku JavaScript

- používa sa jazyk CoffeeScript
- Odsadzuje sa dvoma medzerami.

Zdrojový kód v jazyku Ruby

Formátovanie zdrojového kódu

- Súbory musia byť kódované v UTF-8.

- Na konci riadkov vždy vložiť \n znak.
- Na začiatku riadka použiť odsadenie 2 medzery.
- Na odsadenie použiť 2 medzery. Tabulátor sa nesmie používať.
- Nikdy nepoužívať bodkočiarku.
- Vkladať medzeru okolo operandov a znaku rovná sa.
  - Príklad: x = 1 + 2
- Nepoužívať vnútorné medzery pri () a [].
- Nepoužívať vnútorné medzery okolo výrazov v reťazcoch "hello #{expression}".
- Vnútorné medzery používať okolo hash literálov alebo blokov { a: 1 }, s výnimkou v prípade ako { ... { ... } }, ktorý sa upravuje na { ... { ... }}.
- Vložiť voľný riadok medzi definíciami def, class, module a pod..
- Nedávať biely znak medzi funkciu a zoznam argumentov.
- Nenechať biele znaky na konci riadkov.
- Nenechávať dva a viac voľných riadkov po sebe.
- When vnoriť tak hlboko ako case, nie o úroveň hlbšie.
- Nepoužívať priame priradenie z jazykovej konštrukcie, t.j. nepoužívať result = if ... .
- Po priradení by mal ísť prázdny riadok.
- Pred return by mal ísť prázdny riadok.
- Používať voľné riadky okolo next.

Syntax zdrojového kódu

- Preferovať *each* oproti *for*
- Preferovať && a || oproti and a or kvôli prioritám operátorov.
- Pri definícii metód používať ( ), iba ak má metóda nejaké argumenty.
- Zátvorky ( ) použiť iba ak sprehľadnia kód alebo si to vyžaduje syntax jazyka.
- Preferovať ternárny operátor oproti if/then/else/end
  - Príklad:

```
# zle
if a < b
  then
    a
  else
    b

# dobre
a < b ? a : b
```

- Vhodne používať if/unless.
    - Príklad:

```
# zle
if !vyraz
# dobre
unless vyraz
```

- Nepoužívať unless spolu s else, pozitívny prípad bude vždy ako prvý.
- Používať radšej loop s break ako begin/end/until, resp. begin/end/while.
- Vždy používať {} pre:
    - jednoriadkové bloky,
    - bloky, na ktoré nadväzuje volanie.
        - Príklad:

```
{ “jednoriadkovy blok” }
# blok, na ktory nadvazuje volanie
{ … }.join
```

- Nepoužívať return pokiaľ to nie je nutné.

- Používať self pri volaní vlastných metód.
- Nepoužívať operátor ===
- Nepoužívať $ (globálne) premenné.
- Používať _ pre nepotrebné premenné
    - Príklad:

```
x = hash.map { |_, v| v + 1 }
```

- Používať Array.join a nie *.
- Použiť radšej (1000..2000).include?(x) alebo x.between?(1000,2000) namiesto x >= 1000 && x <= 2000
- Používať predikáty namiesto ==
    - Príklad:

```
# zle
x == nil
# dobre
x.nil?
```

- Vyhýbať sa používaniu vnorených podmienok pri riadení toku programu.
- Nepoužívať

```
# zle
if cond return x
else return y
# dobre
return x if cond
return y
```

- Nepoužívať

```
# zle
return nil if cond
# dobre
return x unless cond
```

Pomenovanie v zdrojovom kóde

- Pomenovať veci presne, ideálne jednoslovne.
    - Príklad:

```
# zle
trumpova_politika_prospieva_americkej_ekonomike =
false
# dobre
beneficial = false
```

- Nepoužívať skratky dlhšie ako jedno písmeno (jednopísmenové skratky ale používať opatrne, odporúča sa použiť ich iba v blokoch). Napríklad žiadne fld, ale celým slovom field alebo skratka f.
- Povolené viacpísmenové skratky sú:
    - args
    - params

Prehľad štýlov pomenovania jednotlivých konštrukcií jazyka je uvedený v Tab. 3

Značka	**Význam**
snake_case	Symboly
snake_case	Metódy

snake_case	Premenné
CamelCase	Triedy
CamelCase	Moduly
SCREAMING_SNAKE_CASE	Konštanty

- Predikáty nemajú prefix is, ale sufix ?.
    - Príklad:

```
# zle
user.is_single
#dobre
user.single?
```

- Databázove boolean stĺpce majú prefix is_, has_, can_ alebo check_
    - Príklad:

```
is_active
has_group_owner
check_invoices_sum
can_auto_assign
```

- Vlastné validačné metódy začínaju prefixom *validate_*
    - Príklad:

```
validates :validate_stuff
```

- Validácie namiesto do blokov, písať do metód

```
# zle
validates do
  ...
end

# dobre
validates :validate_stuff
```

Narábanie s výnimkami v zdrojovom kóde

- Používať fail alebo raise, pre vyhodenie výnimky
- Používať fail 'sprava' namiesto fail RuntimeError, 'sprava'
- Používať fail NejakaVynimka, 'sprava' namiesto fail NejakaVynimka.new('sprava')
- Nepoužívať return v ensure bloku.
- Nikdy nepotláčať výnimky.
- Zákaz používať výnimky pre riadenie toku programu.

Práca s množinami v zdrojovom kóde

- Používať [] a {} na vytvorenie poľa, resp. hash poľa namiesto Array.new a Hash.new.
- Používať first, second, ..., last namiesto [0], [1], .., [-1].
- Používať Set namiesto Array, ak je to vhodné.
- Používať ako kľúče symboly namiesto reťazcov.
- Vždy pokiaľ je to možné použiť { one: 1 } namiesto { :one => 1 }.

Práca s reťazcami v zdrojovom kóde

- Používať '' pre reťazce namiesto "".
- Pre spájanie reťazcov preferujte:

```
# dobre
"#{var1} nieco #{var2}"
# zle
var1 + " nieco " + var2
```

Definition of DONE

Typická User Story:

- Vytvorené testy na otestovanie pridávanej funkcionality (ak je to možné)
- Všetky testy zbehnú bez problémov - zelené testy
- Urobená prehliadka kódu - aspoň 1 človek
- Kód nasadený na produkčnej vetve - staging
- Otestovanie fungovania v produkčnom prostredí - na staging-u
- Funkcionalita predvedená a akceptovaná Product Owner-om

Analytická (Research) User Story:

- Spísaný dokument s výsledkami analýzy, umiestnený na dostupnom mieste
- Prehliadka dokumentu - aspoň jedným človekom

Technická User Story:

- Spísaný dokument s jasným opisom vykonaných zmien
    - Ak je základom zmena kódu, považuje sa kód za dokumentáciu
- Prehliadka dokumentu - aspoň jedným človekom
- Nasadenie zmien do vývojovej vetvy - development

4 Export evidencie úloh

ID	Type	Title	Status
238201	Bug	Opraviť kódovanie vzorov	DONE
238107	User story	Manuálne označenie bločka	IN PROGRESS
238089	User story	Zlepšiť výber vzoru pre bloček	IN PROGRESS
239605	Technical	Rozbehať CI pre projekt	DONE
239626	Technical	Nakonfigurovať Staging	DONE
238199	Bug	Opraviť preklady	DONE
238682	User story	Pridať link na stránku do aplikácie	DONE
239074	Bug	Opraviť odrezávanie znaku eura	DONE
238100	User story	Prezentačné stránky tímu	DONE
238103	User story	Tímový plagát	DONE

Obr. 1: Stav úloh po ukončení šprintu Beatrix Kiddo

ID	Type	Title	Status
242923	Research	Android workshop	IN PROGRESS
242924	User story	Pridať pripojenie na API do android aplikácie	IN PROGRESS
243058	Bug	SPLIT Opraviť kódovanie vzorov	IN PROGRESS
243059	User story	CLONE Manuálne označenie bločka	IN PROGRESS
243062	User story	CLONE Zlepšiť výber vzoru pre bloček	IN PROGRESS
241209	Bug	Pád mobilnej app po minimalizácií	DONE
238692	Bug	Aplikácia padne ak sa neodfotí nič alebo textová časť je moc malá	TODO
240085	Bug	Nefunguje mazanie používateľa a jeho aktualizácia	DONE
241227	Bug	Nastavenie default početnosti položiek na portáli	DONE
239060	Bug	Zaokrúhlenie čísiel zobrazovaných na portáli	DONE

Obr. 2: Stav úloh počas priebežného stretnutia šprintu Bil

ID	Type	Title	Status
242923	Research	Android workshop	DONE
242924	User story	Pridať pripojenie na API do android aplikácie	IN PROGRESS
243058	Bug	SPLIT Opraviť kódovanie vzorov	DONE
243059	User story	CLONE Manuálne označenie bločka	DONE
243062	User story	CLONE Zlepšiť výber vzoru pre bloček	IN PROGRESS
241209	Bug	Pád mobilnej app po minimalizácií	DONE
238692	Bug	Aplikácia padne ak sa neodfotí nič alebo textov...	TODO
240085	Bug	Nefunguje mazanie používateľa a jeho aktualizá...	DONE
241227	Bug	Nastavenie default početnosti položiek na portáli	DONE
239060	Bug	Zaokrúhlenie čísiel zobrazovaných na portáli	DONE

Obr. 3: Stav úloh po skončení šprintu Bil

ID	Type	Title	Status
245624	Bug	CLONE Aplikácia padne ak sa neodfotí nič ale...	TODO
245620	User story	CLONE Pridať pripojenie na API do android a...	IN PROGRESS
244271	Research	Review článku od Andrei	DONE
244270	User story	Upresniť landing page a kúpiť doménu	IN PROGRESS
245403	User story	Dokumentácia na prvé TP odovzdanie	IN PROGRESS
245630	Research	Treba zaistiť, že produkčný server nám neza...	DONE
244280	User story	Ukladanie všetkých dát	IN PROGRESS
245633	User story	Nastaviť dočasný support mail	DONE

Obr. 4: Stav úloh počas priebežného stretnutia šprintu O-Ren Ishii

ID	Type	Title	Status
245624	Bug	CLONE Aplikácia padne ak sa neodfotí nič alebo textová časť je moc malá	DONE
245620	User story	CLONE Pridať pripojenie na API do android aplikácie	DONE
244271	Research	Review článku od Andrei	DONE
244270	User story	Upresniť landing page a kúpiť doménu	DONE
245403	User story	Dokumentácia na prvé TP odovzdanie	DONE
245630	Research	Treba zaistiť, že produkčný server nám nezablokujú kvôli peniazom	DONE
244280	User story	Ukladanie všetkých dát	DONE
245633	User story	Nastaviť dočasný support mail	DONE

Obr. 5: Stav úloh po skončení šprintu O-Ren Ishii

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Branislav Pecher, Jozef Mláka, Michal Kováčik,
Pavol Ondrejka a Daniel Novák

KILL-BILLS

Dokumentácia k inžinierskemu dielu z predmetu Tímový Projekt

Vedúci tímu: Ing. Jakub Šimko, PhD.

Akademický rok: 2017/2018

5 Big picture

5.1 Úvod

Platforma Kill-Bills, ktorú sme v projekte pre súťaž ImagineCup vytvorili, umožňuje používateľom jednoduchým spôsobom, prostredníctvom aplikácie v mobilnom telefóne, digitalizovať blokové doklady z registračných pokladníc, a ukladať ich ako štruktúrované dáta. Takto štruktúrované dáta sú potom dostupné, prostredníctom HTTP API, aj v našej webovej aplikácií na sledovanie výdavkov.
Aplikácia na sledovanie výdavkov sa, okrem jednoduchej prezentácie digitalizovaných bločkov používateľovi, venuje aj kategorizácií jednotlivých položiek a snaží sa ich priradiť do rozpočtových kapitol ako sú potraviny, oblečenie a iné. Vďaka tomu umožňuje používateľovi podrobný náhľad do jeho výdavkov, vizualizuje pomery výdavkov na jednotlivé kapitoly, a zobrazuje trendy v týchto kapitolách. V konečnom dôsledku mu teda umožňuje optimalizovať výdavky.

5.2 Globálne ciele pre ZS/LS

Náš najdôležitejší cieľ je poskytnutie beta verzie aplikácie užívateľom a začať so zbieraním dôležitej spätnej väzby od tzv. *early adopters*, teda skorých záujemcov a uživateľov. Ak sa podarí dobre podchytiť týchto nadšencov pre našu technológiu, môžeme očakávať jej následné rýchle šírenie medzi ďalšími potenciálnymi používateľmi. Na dosiahnutie tohto cieľa sme si na začiatku semestra vytýčili splniť nasledovné kroky:

- Aplikácia na zdieľanie výdavkov
- Nasadenie mobilnej aplikácie do produkcie
- Zadefinovať ceny na základe trhu
- Funkcionalita pre business
- Implementovať API pre business
- Kompletný redizajn webovej aplikácie
- Mobile friendly aplikácia na sledovanie výdavkov
- Bločková lotéria
- Úprava algoritmu na kategorizáciu položiek
- Prepracovanie a zlepšenie jednotlivých komponentov

5.3 Princíp spracovania nasnímaného bločku

- **predspracovanie snímky** - snímka sa najskôr oreže. Orezanie sa sa vykonáva pomocou detekcie línií počítačovým videním, z ktorých sa algoritmus pokúša zostrojiť štvoruholník. Najväčší takýto identifikovaný štvoruholník je potom považovaný za blokový doklad a zo snímky sa vyreže. Následne sa vykoná transformácia z orezaného štvoruholníka na regulárny obdĺžnik (strany sú navzájom na seba kolmé). Nakoniec sa pomocou grafických filtrov zvýši kontrast medzi pozadím a samotným textom dokladu. Predspracovanie má za cieľ zvýšiť čitateľnosť snímky pre OCR a orezanie má za cieľ znížiť počet artefaktov. Takto pripravená snímka sa nahrá na cloudové úložisko a následne sa pošle do ďalšej časti dátovodu.

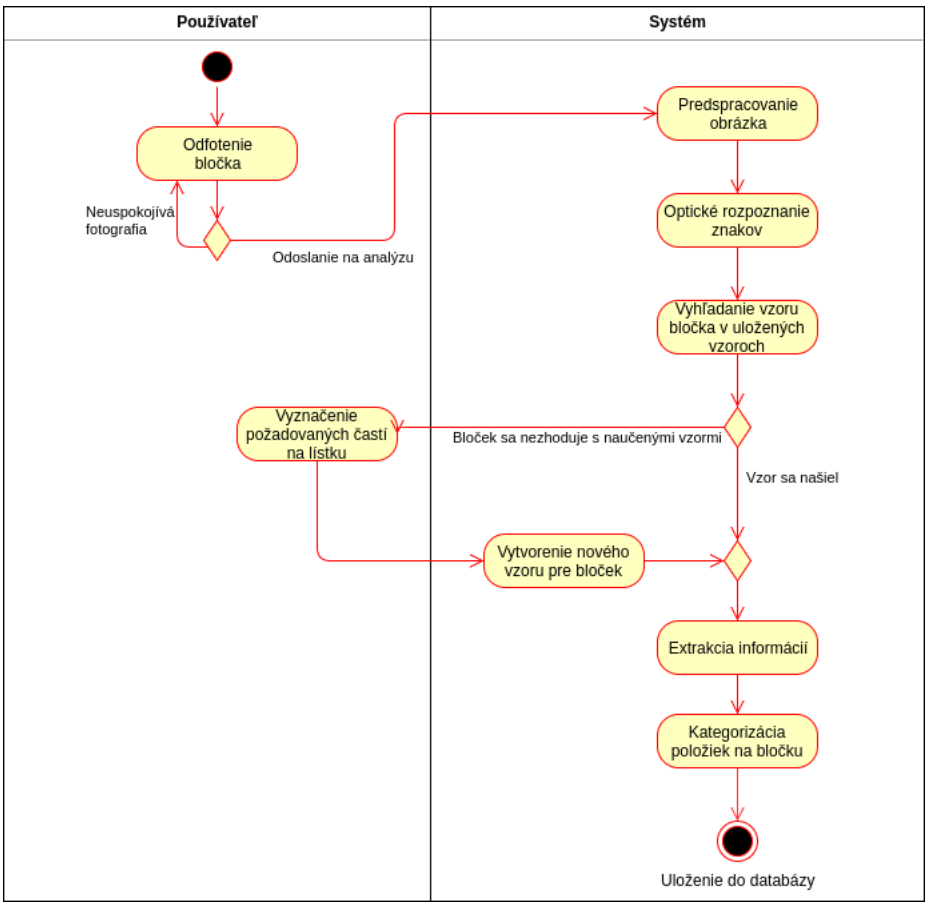
- **optické rozpoznanie znakov** - predpripravená snímka sa následne pošle na OCR. Výstupom z OCR je surový reťazec znakov obsahujúci prečítané znaky, vrátane artefaktov a chýb. Takýto reťazec sa posiela do ďalšej časti dátovodu.

- **parsovanie** - reťazec z OCR sa najskôr skontroluje na dĺžku (či je zmysluplne dlhý, lebo výstupom z OCR môže byť aj jeden znak, čo samozrejme nemá zmysel ďalej parsovať), potom sa v ňom vyhľadajú kľúčové slová (MF, DKP, IČO, a iné) a nakoniec sa doklad prevedie na štruktúrovaný JSON dokument.

- **uloženie do databázy** - štruktúrovaný dokument z parsera sa uloží do databázy, ktorá potom bude môcť vracať vyžiadané dáta do správ o spotrebe ako pre používateľa, tak aj pre predajcov.

- **klasifikácia položiek** - po uložení do databázy je ešte treba jednotlivé položky klasifikovať(potraviny, drogéria, elektronika a iné).

Dôvod prečo sme sa rozhodli pre dátovod s oddelenými časťami je ten, aby sme mohli fungovať podobne ako pásová výroba v továrni. Dátovod nám umožňuje jednotlivé požiadavky vykonávať prúdovo obsluhované naraz. Ďalším argumentom pre takúto architektúru je, že nám umožňuje nezávisle od seba vyvíjať jednotlivé časti dátovodu s tým, že máme vopred dohodnuté komunikačné rozhrania medzi susednými blokmi.



Obr. 6: Proces spracovania blokového dokladu v platforme KillBills

6 Moduly systému

6.1 Aplikácia na sledovanie výdavkov

Aplikácia je postavená na štandardnej MVC architektúre vo frameworku Ruby on Rails. Všetky prístupy do produkčných databáz, ako aj produkčné secret tokeny sa načítavajú z globálnych premenných shellu. Tieto sú umiestnené v Openshift konfigurácií v clusteri, takže po vybuildovaní novej verzie docker image-u sa netreba báť, že by bolo treba ešte niečo nastavovať.

6.1.1 Kontrolery

Kontrolery sú rozdelené do 3 namespaceov.

- **default namespace:**
  - *SiteController:* skontroluje čí je používateľ prihlásený a načíta Reactovský frontend.
  - *i18nController:* rieši jazykové mutácie obsahu stránky, ale momentálne nie je plne implementovaný.
- **user namespace:** nachádzajú sa to kontrolery vygenerované gemom devise, ktorý aplikácia používa na riešenie správy používateľov.
- **api namespace:** nachádza sa v ňom vnorený ďalší menný priestor označujúci verziu api. Avšak momentálne je len jeden „v1“
  - *ApiController:* Slúži ako báza pre ostatné kontrolery v priestore api. Oproti defaultnému ApplicationControlleru neodpovedá na html, ale na json requesty.
  - *BillItemsController:* Podporuje klasické cread, update a destroy akcie nad položkami bločka. Index ani show sa momentálne nejaví potreba implementovať, keďže aplikácia nemá taký view, ktorý by tieto dáta potreboval.
  - *BillsController:* Implementuje všetky operácie railsovskej resource. Pred všetkými operáciami kontroluje, či má prihlásený používať oprávnenie vyžiadať bločky, ktoré si vyžiadava. Pre operáciu index je možné nastaviť, prostredníctvom parametrov, filtre. Momentálne je len podporovaný filter date_filter. Pokiaľ tento nie je nastavený, defaultné nastavenie je 30 posledných dní.
  - *CategoriesController:* Obsahuje 4 verejné metódy. Prvé 3 menované metódy majú podporu pre date_filter:

    - INDEX: vráti root kategórie s informáciami pre tabuľku kategórií.
    - SHOW_CHILDREN: vráti kategórie, ktoré sú priamymi potomkami kategórie, ktorá je zadaná ako parameter požiadavky.
    - SHOW_ITEMS: vráti položky patriace do kategórie a do všetkých jej podkategórií.
    - ALL_SELECT: vráti zoznam všetkých kategórií. Používa sa pre selecty, ktoré doťahujú dáta just in time.
  - *UserController:* Slúži pre autentifikáciu používateľa v mobilnej aplikácií.
  - *OcrController:* Poskytuje rozhranie, ktoré využíva platforma KillBills na ukladanie a kategorizáciu bločkov. Má len jednu metódu, post_json, ktorá očakáva parameter user_id a json v takomto tvare: json "shop": "Terno real estate s.r.o", "date": "07.10.2016", "time": "9:04", "DKP": "1002120151011178", ïtems": ["name": "Rozok grahamovy 60g PBP", "quantity": "4", "price": "0.09", ämount": "0.36
    ], "sum": "0.36

6.1.2 Modely

Aplikácia obsahuje 5 tried modelu. Atribúty modelov sú popísané v /db/schema.rb.

- **Bill:** Model má implementované 2 scopy pre date_filter. Sú to from_date a to_date. Default ordering je nastavený podľa dátumu pridania, aby sa najnovšie naskenované bločky zobrazovali na vrchu tabuľky.

  Takisto má dôležité volanie na before_save metódu, v ktorej sa vypočíta atribút suma, sumovaním cez sumy položiek.

- **BillItem:** Model má implementované 4 scopy. Podobne ako Bill sú to from_date a to_date. Ďalej je to scope user, ktorá nájde bločky patriace používateľovi zadanému ako parameter. Posledný scope je belongs_to_children_of ktorý vytiahne položky ktoré patria kategórií a všetkým jej deťom.

  BillItem vykonáva 3 validácie. Kontroluje či atribúty suma, jednotková cena a kvantita sú číselné hodnoty.

  Obsahuje privátnu metódu categorize, ktorá sa volá before_save, pokiaľ nie je položka už zakategorizovaná.

- **Category:** Model má implementované 2 scopy pre date_filter a metódu, ktorá vytvára z entiry hashmapu, ktorá je vhodná pre zobrazenie v tabuľke.

- **Shop:** Nie je ničím zaujímavý.

- **User:** Model vygenerovaný devise gemom.

6.1.3 Obrazovky

Frontend je implementovaný vo frameworku React. Skladá sa zo 4 modulov, ktoré sú v priečinku /assets/javascript/components.

- **bills_table:** obsluhuje zobrazovanie bločkov v tabuľke a CRUD operácie nad nimi. Modul sa skladá z 3 komponentov:
  - *bill_form:* formulár na manuálne pridanie nového bločka.
  - *bills_table:* obsluhuje tabuľku, v ktorej sú zobrazované bločky.
  - *bills_table_row:* predstavuje jeden bloček. Obsluhuje in-line editovanie záznamov.
- **categories_table:** obsluhuje zobrazovanie kategórií v tabuľke a generuje príslušné vizualizácie prostredníctvom diagramov. Skladá sa zo 4 komponentov.
  - *categories_table:* obsahuje tabuľku, v ktorej sa zobrazujú kategórie a podľa módu, v ktorom sa nachádza (detail kategórie, alebo prehľad kategórií) rozhoduje, ktorý diagram vygenerovať.
  - *categories_table_row:* predstavuje jednu kategóriu.
  - *line_chart:* slúži na generovanie čiarového diagramu v detaile kategórie.
  - *pie_chart:* slúži na generovanie koláčového diagramu v prehľade kategórií.
- **filter:** slúži na filtrovanie záznamov, ktoré sa majú zobraziť. Skladá sa len z jedného komponentu filter, ktorý zobrazuje formulár.
- **items_table:** slúži pre zobrazovanie detailu bločku a kategórie. Zobrazuje položky z bločkov. Skladá sa z 3 komponentov.
  - *item_form:* formulár na pridávanie nových položiek do bločka. Je prístupný len zobrazení detailu bločka.
  - *items_table:* obsluhuje tabuľku, v ktorej sú zobrazované položky.
  - *item_table_row:* predstavuje jednu položku. Obsluhuje in-line editovanie.

Pohľady obsluhujúce prihlásenie a registráciu používateľa sú vygenerované prostredníctvom gemu devise a niesú implementované v Reacte, ale v .erb súboroch v priečinku /views.

6.2 Modul websocket-service

Modul je implementovaný vo frameworku Sinatra a používa websocket server z ruby gemu Faye. Faye sa stará o prihlasovanie nových klientov a rozosielanie správ. Sinatra sa stará o webserver, ktorý prijíma správy, ktoré treba preposlať prostredníctvom Faye. Správy sa posielajú HTTP POST requestom na metódu /socket s parametrami „channel“, čo je kanál, na ktorý správu poslať a „message“, ktorý predstavuje správu.

6.3 Modul work-dispatcher

Modul je implementovaný vo frameworku Sinatra. Obsahuje dve implementácie Sidekiq Workera (ImageUploader a PatternCreator), kde každý obsluhuje jednu z dvoch HTTP metód modulu. HTTP rozhranie modulu je nasledovné:

- **POST /image_upload** s parametrami file - predstavujúcim snímku bločku a user_id. Metóda uploadne obrázok, naplánuje jeho ďalšie spracovanie cez Sidekiq workera ImageUploader a odošle odpoveď o úspešnom uložení obrázka, spolu s názvom kanálu na websocket-service module, na ktorý sa má klient prihlásiť.
- **POST /send_marked** s parametrami file - predstavujúcim označkované xml a user_id. Metóda serializuje prijaté xml do súboru pre archiváciu a naplánuje ďalšie spracovanie prostredníctvom Sidekiq workera PatternCreator.

6.4 Modul na optické rozpoznávanie znakov

Modul je implementovaný v jazyku C++ a slúži na prevod obrázku na digitálny text. Ide o HTTP server komunikujúci cez REST API. Tento modul poskytuje nasledujúce HTTP rozhranie:

- **POST /file** so vstupným JSON parametrom, ktorý v premennej “filename“ obsahuje celú cestu k vstupnému obrázku. Táto metóda prevedie vstupný obrázok pomocou optického rozpoznávania znakov na digitálny text, pričom prevedie na tomto texte základnú korekciu slov. Takto prevedený text následne odošle naspäť ako odpoveď vo formáte XML správy (formát XML správy je definovaný v časti 6.4.2). Ak počas spracovávania obrázku nastal error, napríklad ak bola zadaná chybná cesta k vstupnému obrázku, vráti metóda chybový kód 400 spolu s popisom chyby, ktorá nastala.

6.4.1 Spôsob fungovania

Na prevod obrázku na digitálny text sa využíva Tesseract, ktorý sme upravili na naše potreby rozpoznávania textu na bločkoch. Obrázok je najprv rozdelený na jed-

notlivé riadky a slová, ktoré sú následne prevedené na digitálny text. Keďže sa však na obrázkoch bločkov môžu vyskytovať rôzne artefakty, musia byť niektoré slová opravené, poprípade úplne vymazané ak sa v skutočnosti na bločku nevyskytujú. Na tento účel definuje aplikácia 3 základné triedy:

- **Text** - táto trieda slúži na organizáciu všetkých tried typu “Line“ v programe. Zároveň slúži na vygenerovanie výsledného XML súboru. Aby mohla byť trieda typu “Line“ pridaná do zoznamu všetkých takýchto tried, musí obsahovať minimálne jednu triedu typu “Word“.

- **Line** - táto trieda reprezentuje všetky rozpoznané riadky textu na obrázku. Obsahuje zoznam všetkých tried typu “Word“, ktoré sa vyskytujú na danom riadku. Ďalší záznam do tohto zoznamu je možné pridať iba vtedy, ak daný záznam obsahuje text dĺžky väčšej ako 0. Táto trieda si zároveň drží údaj o poradí svojho výskytu a informáciu o pozícii na pôvodnom obrázku.

- **Word** - táto trieda reprezentuje všetky rozpoznané slová na obrázku. Drží si informáciu o poradí svojho výskytu a informáciu o pozícii na pôvodnom obrázku. Táto trieda taktiež implementuje základnú korekciu textu, ktorá kontroluje, či sa jedná o skutočné slová, alebo slová vzniknuté z nečistôt na obrázku. Korekcia je vykonaná hneď pri vytváraní tejto triedy, aby bolo možné odstraňovať nekorektné slová za behu.

6.4.2 Formát XML správ

Výsledný XML súbor môže obsahovať iba 3 typy elementov, ktoré viac menej kopírujú triedy v aplikácií. Ide o nasledujúce elementy:

- **Text** - jedná sa o koreňoví element v XML súbore. Môže obsahovať iba elementy “Line“.

- **Line** - predstavuje riadok textu. Obsahuje 5 atribútov - id reprezentujúce poradie riadku v texte a x-ové a y-ové koordináty ľavého horného rohu a pravého dolného rohu riadku na obrázku - spolu vytvárajú ohraničujúci štvoruholník pre daný riadok. Taktiež obsahuje vnorené elementy typu "Word", pričom vždy obsahuje minimálne jeden.

- **Word** - predstavuje slovo v texte. Obsahuje 5 atribútov, rovnakých ako pri elemente “Line“ - id a koordináty ohraničujúceho štvoruholníka. Taktiež tento element obsahuje text slova, ktorý musí mať dĺžku minimálne jedného znaku.

6.5 Modul na extrakciu dát

Modul slúži na extrakciu textových informácii z XML súborov vygenerovaných pomocou OCR modulu. Pre ľahké prototypovanie a jednoduché narábanie s textom je modul implementovaný v jazyku Python.

6.5.1 REST rozhranie

Modul je navrhnutý ako HTTP server komunikujúci cez RESTful rozhranie. Servis bol vytvorený pomocou Python frameworku Falcon, pričom beží na unixovom serveri Gunicorn. Rozhranie modulu vyzerá nasledovne:

- **GET /** bez tela správy slúži ako jednoduchý hello-request na zistenie, či server beží. Metóda zároveň vráti informácie o systéme a Python distribúcii.
- **POST /** - hello-request na zistenie či server beží. Telo požiadavky očakáva text, ktorý následne vráti ako odpoveď. Metóda zároveň vráti informácie o systéme a Python distribúcii.
- **POST `/parse`** s XML dokumentom ako telom požiadavky, ktorý bol vrátený OCR modulom. Metóda vydoluje z XML dáta na základe doposiaľ vytvorených vzorov (6.5.3), pomocou metódy `/create_pattern` a vráti ich vo formáte JSON dokumentu. V prípade, že nebol nájdený vhodný vzor, aplikácia vráti HTTP kód 206 a čiastočne vyparsované dáta v JSON dokumente. Ak nebol doposiaľ vytvorený žiadny vzor, aplikácia vráti HTTP kód 204.
- **POST `/check_items`** s XML dokumentom ako telom požiadavky, ktorý obsahuje informácie o príslušnosti riadkov a slov k jednotlivým entitám pomocou atribútu `tag`. XML by malo obsahovať riadky označené hodnotou atribútu ako `items` a tie by mali mať označené slová reprezentujúce práve jednu položku. Odpoveďou je XML s riadkami pôvodne označenými ako `items`, označenými ako `item_main_*` alebo `item_ignored_*`, kde `*` predstavuje unikátne číslo pre každý ignorovaný riadok alebo riadky, čo obsahujú položku. Metóda vráti HTTP kód 206 v prípade, že boli pridané označenia s hodnotou `item_ignored_*`. HTTP kód 417 je vrátený, keď v XML dokumente nebolo nájdené označenie riadku `items` s označenými slovami.
- **POST `/create_pattern`** s formulárom obsahujúcim parametre `file` a `data_identification_token`. Metóda na základe informácii v označenom XML dokumente, ktorý obsahuje parameter `file`, vytvorí vzor bločku, uloží ho s názvom korešpondujúcim s parametrom `data_identification_token` a vráti označené dáta v JSON dokumente. Ak XML neobsahuje žiadny riadok označený ako `item_main_*`, vráti HTTP kód 417.

6.5.2 Vlastné knižnice/balíky

- **`parser_lib`** - balík obsahujúci API na parsovanie XML súborov z OCR modulu a vytváranie vzorov reprezentujúcich štruktúru pokladničných bločkov. Model vzorov je bližšie popísaný v sekcii Model vzorov 6.5.3.
- **`util`** - balík obsahujúci všeobecné pomocné funkcie na serializáciu a informácie o systéme.

6.5.3 Model vzorov

Triedy reprezentujúce vzor štruktúry bločku ho rekurzívne popisujú, a preto je diagram tried 7 inšpirovaný vzorom Composite. Rozhranie `PatternClass` je pomyselné rozhranie, podľa ktorého je možné volať všetky triedy popisujúce vzor. Trieda `State` slúži ako médium na uchovávanie informácií pri parsovaní. Trieda `Line` predstavuje list a všetky ostatné triedy sú kompozitné.

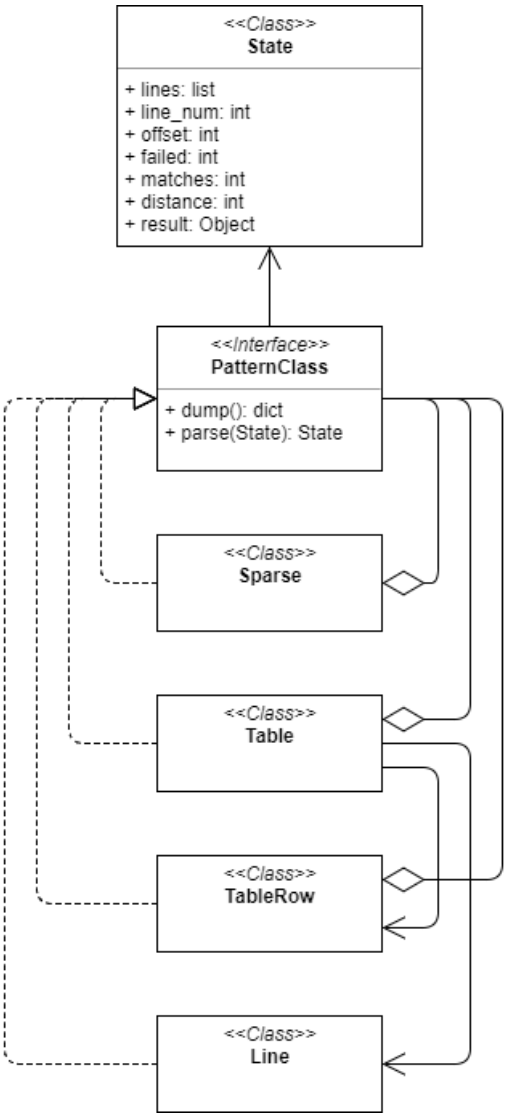
- **`Sparse`** - trieda zaznamenáva, na ktorých riadkoch (relatívna pozícia) sa nachádzajú cieľový text alebo ďalší objekt
- **`Table`** - trieda popisujúca tabuľku, čiže viacero rovnako vyzerajúcich riadkov-objektov, ktorých môže byť viac typov, čo zastrešuje trieda `TableRow`
- **`TableRow`** - trieda definujúca jeden riadok v tabuľke
- **`Line`** - trieda extrahujúca informácie priamo z textového riadku

6.5.4 Závislosti modulu

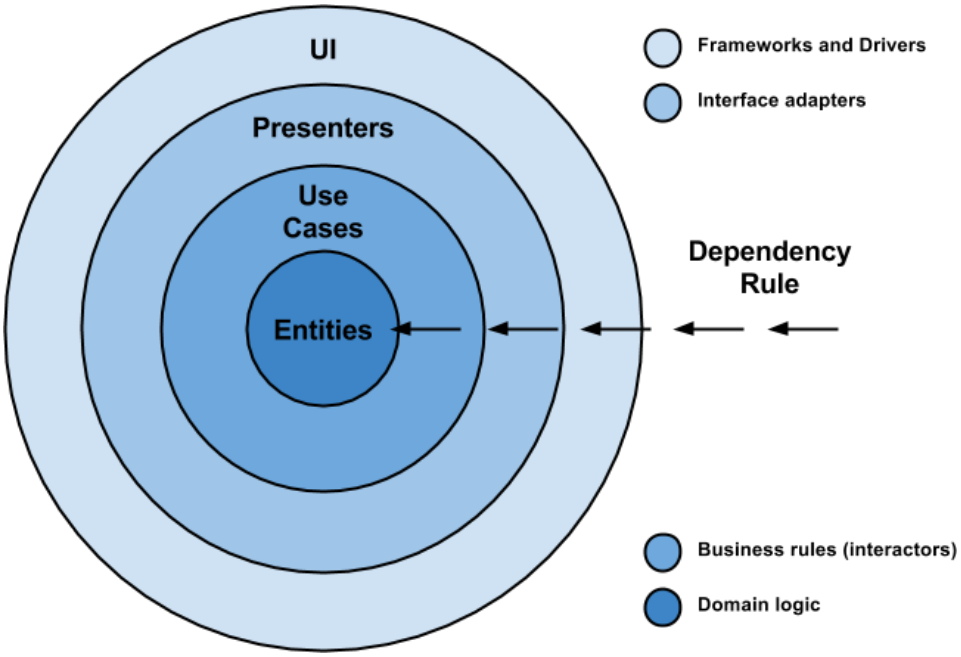
- libxml2-dev & libxslt-dev
- lxml
- falcon
- falcon-multipart
- gunicorn

6.6 Android klient

Architektúrou aplikácie je upravená ”Clean Architecture”, ktorá rozdeľuje aplikáciu na štyri vrstvy: frameworky a ovládače, biznis logiku, prípady použitia a entity. Architektúra bola zvolená z viacerých dôvodov, hlavným z nich je nezávislosť jednotlivých modulov a vysoká testovateľnosť. Na obrázku 8 je zobrazená schéma závislostí medzi vyššími vrstvami.



Obr. 7: Diagram tried zobrazujúci závislosti medzi triedami reprezentujúcimi vzor štruktúry pokladničného bločku.



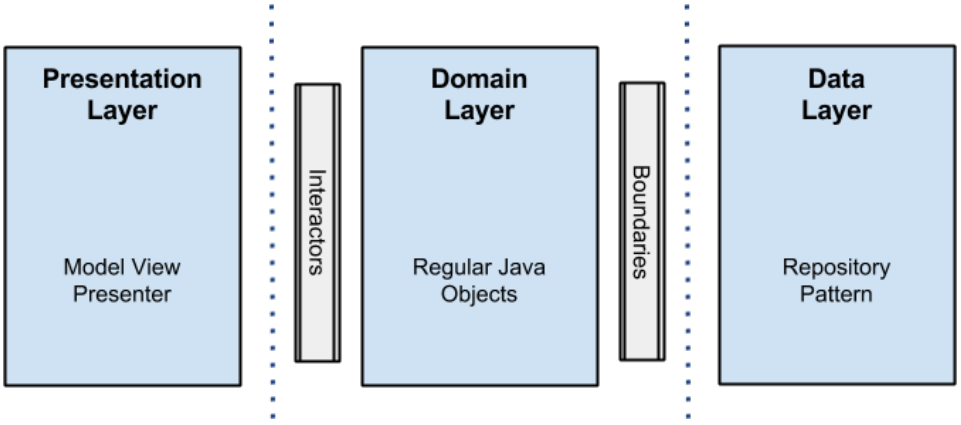
Obr. 8: Schéma závislostí medzi vrstvami. Vnútorné vrstvy by nemali vedieť o existencii vyšších vrstiev.

Aplikácia je rozdelená do troch vrstiev: prezentačná, doménová a dátová. Na obrázku 8 sú prezentačná a dátová vrstva umiestnené na vonkajších dvoch vrstvách (UI a Presenters) a doménová na vnútorných (Use cases a entities). Schéma 9 zobrazuje komunikáciu medzi týmito vrstvami. Boundaries sú implementované návrhovým vzorom repository a ”Interactors” sú v implementácii označené ako ”Use case”. Ideálne by mali všetky vrstvy komunikovať iba cez doménovú vrstvu.

6.6.1 Prezentačná vrstva

Táto vrstva je android vrstva, ktorá obsahuje hlavne používateľské rozhrania a interakcie s doménovou vrstvou. V projekte je táto vrstva umiestnená v najvyšom module ”presentation”. Používa sa tu štýl MVP, ktorý ďalej rozdeľuje túto vrstvu na ďalšie tri vrstvy:

- **model** - obsahuje modely výhradne pre rozhranie, teda iba určené na prezentáciu, umiestnené v balíku ”model”
- **view** - obsahuje všetku funkcionalitu, ktorá obsahuje vykresľovanie používateľského rozhrania, v aplikácii sa za view považujú android aktivity, fragmenty, vlastne definované view komponenty, xml definície rozhrania a tiež rozhrania pre prácu s rozhraním, ktoré používa presenter. Sú umiestné v balíku ”view”.



Obr. 9: Základná schéma architektúry. ”Boundaries” a ”Interactors” sú jediným spôsobom komunikácie medzi jednotlivými vrstvami. Ideálne by všetky ostatné moduly mali komunikovať výhradne cez doménovú vrstvu

- **presenter** - obsahuje všetku logiku, pri ktorej sa interaguje s viewom. Vrstva komunikuje výhradne cez vyššie spomenuté rozhrania, pričom udalosti z používateľského rozhrania sa posielajú priamo vyvolaním metód presentera, keďže view obsahuje inštancie presentera. V presenteroch sa tiež vyvoláva biznis logika z vyššej doménovej vrstvy.

V tejto vrstve je na vytváranie a vkladanie závislostí používa dependency injection prostredníctvom knižnice Dagger. Moduly a komponenty sú všetky umiestnené v balíku ”di”.

6.6.2 Doménová vrstva

Vrstva je umiestnená v moduli ”domain”. Je nezávislá na androide, teda je znovupoužiteľná aj pre iné platformy. Obsahuje biznis logiku, ktorá je implementovaná prípadmi použitia (use cases). Prezentačná vrstva môže vytvoriť inštanciu a zavolať metódu ”execute”, ktorá asynchrónne vráti výsledok operácie.

Vrstva používa vzor repository na perzistenciu entít. Repozitáre sú definované ako rozhrania a slúžia ako výstup pre prípady použitia, ktoré potrebujú CRUD operácie nad entitami. Repozitáre sú implementované v dátovej vrstve.

Vrstva je rozdelená do 4 balíkov:

- **executor** - obsahuje definície rozhraní pre asynchrónne vykonávanie prípadov použitia
- **interactor** - obsahuje prípady použitia
- **model** - modely špecifické pre doménu
- **repository** - definuje repository rozhrania

6.6.3 Dátová vrstva

Táto vrstva slúži na získavanie, transformáciu a ukladanie dát. Je to android modul, ktorý implementuje rozhrania definované v doménovej vrstve. Pracuje s čistými dátami a používa nástroje špecifické pre platformu, ktorými môžu byť databáza, pripojenie na vzdialené API alebo prístup k lokálnym súborom. Je umiestnená v module ”data”.